Jeffrey D. Ullman – Jennifer Widom

ADATBÁZIS-RENDSZEREK

Alapvetés

Második, átdolgozott kiadás

www.panem.hu

Jeffrey D. Ullman – Jennifer Widom

Adatbázisrendszerek

Alapvetés

Jeffrey D. Ullman – Jennifer Widom

Adatbázisrendszerek

Alapvetés

Második, átdolgozott kiadás

Panem

A mű eredeti címe: A First Course in Database Systems
Third Edition by Jeffrey D. Ullman, Jennifer Widom

Második, átdolgozott kiadás

ISBN 978-963-545-481-5

Szerkesztette: dr. Benczúr András

Fordította: Ács Zoltán, Buza Antal, Cserges Enikő, Csizmazia Balázs, Gyenizse Pál, Hajas Csilla, Kónya László, Kovács György, Nikovits Tibor, Nyitrai Erika, Varga Balázs
Lektorálta: dr. Márkus Tibor
Felelős szerkesztő: Szabó Ildikó
Borítóterv: Tóth Attila
Nyomdai előkészítés: Gerner József. Szabó Ildikó

A kiadásért felel a Panem Könyvkiadó Kft. ügyvezetője, Budapest, 2009

panem@panem.hu
www.panem.hu

Nyomta és kötötte: a Dürer Nyomda Kft. Gyulán
Felelős vezető: Kovács János

Tartalomjegyzék

Előszó a magyar kiadáshoz

A Stanford Egyetem Adatbázisrendszerek csoportja által készített és használt könyvpáros – *A First Course in Database Systems* (1997) és *Database System Implementation* (2000) – világszerte az adatbázis-kezelés legnépszerűbb tankönyvei közé tartoznak az informatikusképzésben. A két kötet *Adatbázisrendszerek – Alapvetés* (1998), illetve *Adatbázisrendszerek megvalósítása* (2001) címmel az amerikai megjelenést követő években magyarul is megjelent a Panem Könyvkiadó gondozásában.

Az első kötet az adatbázisrendszereknek azt a két legfontosabb oldalát hangsúlyozza, amelyek a legtöbb informatikus hallgató számára a leghasznosabbak: az adatbázis-tervezést és az adatbázis-programozást. A második kötet az adatbázis-kezelő rendszerek megvalósítását tekinti át, nevezetesen a tárolási szerkezeteket, a kérdésfeldolgozást és a tranzakciókezelést.

Az első kiadás kiválasztásakor és fordításakor még nem volt a könyvvel kapcsolatban oktatási tapasztalatunk. Azonban tíz éves oktatási háttér alapján most már bizonyosak lehetünk abban, hogy jól választottunk.

Nyugodtan kijelenthető, hogy a könyv bármely felsőfokú informatikai képzéshez az adatbázisok témakörben az elvárható alapismereteket foglalja egységbe. A megjelenése után rövid időn belül a hazai felsőoktatásban a fentiek visszaigazolásaként alaptankönyvvé vált. A felsőoktatás új, kétlépcsős rendszerében a három informatikus szak mindegyikén az adatbázis-kezelés tantárgyakhoz ezt a könyvet az oktatási intézmények szinte kivétel nélkül kötelező irodalomként adták meg. Ezért a könyv legújabb kiadásának magyar nyelvű megjelentetésére igen nagy szükség volt.

Időközben a Pearson Education még egy átdolgozott kiadást jelentetett meg, a jelen kötet fordítása pedig már a harmadik, 2008-ban napvilágot látott angol nyelvű kiadás alapján készült. Ebben a kiadásban igen jelentős változtatást hajtottak végre a szerzők, a könyv legalább ötven százalékban újnak tekinthető. Ez kerülhet most az olvasó kezébe magyar nyelven, mégpedig – és ezt hangsúlyoznunk kell – az amerikai kiadás megjelenésének évében.

A könyv felépítése az oktatás célszerűsége érdekében lényegesen változott. A relációs modellel való első megismerkedés került az elejére, beleértve a relációs algebra bevezetését, a sémák tervezését és az elemi kulcs- és hivatkozási megszorításokat. Ezt a korábbi kiadásoktól eltérően nem a magas szintű, fogalmi adatmodellezés követi, hanem a relációs adatbázisok tervezésének, a függőségek-

nek és normalizálásnak a korábbinál egzaktabb, kicsit részletesebb tárgyalása. A magas szintű adatmodellezés az E/K, az UML és végül az ODL rövidített bemutatása mindjárt a relációs modellre való leképezéssel együtt történhet. A relációs modellel és programozásával való mélyebb ismerkedés az 5. fejezetben a relációs algebrának mint lekérdező nyelvnek részletesebb bemutatásával kezdődik, és a logikai alapú nyelvek közül a Datalog rekurziót nem tartalmazó tárgyalásával folytatódik. A hazai tipikus tantárgyi tematikákban szerepel még a predikátum-kalkulushoz hasonló relációs kalkulus, amit szükség esetén ennek a fejezetnek tárgyalása után célszerű megtartani. A 6., 7. és 8. fejezetek teljes mértékben az SQL lehetőségeinek bemutatását tartalmazzák.

Megújult az SQL használatát és programozását lehetővé tevő környezetek tárgyalása; a háromrétegű architektúra, a tárolt eljárások (PSM), a C nyelv hívásszintű interfésze (CLI), a Java-környezetből az SQL használatát biztosító JDBC, és végül az SQL PHP-ben való használata; mind újdonság az első kiadáshoz képest. A 10. fejezetben szereplő haladóbb témakörök közül a titkosítás és a jogosultságok kezelése, valamint a rekurzió az SQL-99-ben már szerepelt az első kiadásban is. Azonban az objektumrelációs modell és az SQL-99-ben szabványosított implementálása új tárgyalásban szerepel itt.

Legjelentősebb újdonság a két új fejezet, amelyekben a relációs modell mellett másik meghatározó modellé vált félig-strukturált adatmodell, illetve szabványosodó változata, az XML adatmodellezési lehetőségei (DTD, XML Schema) és programozási, lekérdezési nyelvei (XPath, XQuery és XSLT) kerülnek bemutatásra.

A kötet a bevezető tankönyvek minden követelményét kielégíti. A képzettebb, a formális definíciókkal könnyebben boldoguló olvasó kihagyhatja a definíciók, használati lehetőségek példákon keresztül történő részletes bemutatását, illusztrálását, míg annak, aki a példákon keresztül érti meg igazán a még szokatlan témakört, didaktikusan tervezett példarendszer is rendelkezésére áll. A könyv tárgyalásmódja mind a megértéshez, mind a megjegyzéshez elegendő szabadságot biztosít az olvasó számára.

Köszönet illeti a Panem Könyvkiadót, hogy támogatás nélkül is vállalkoztak a második magyar kiadás megjelentetésére, megértették, hogy milyen azonnali igény van a könyvre, és mi teszi sürgőssé az új kiadás magyar nyelvű megjelentetését. Ezzel újra rendelkezésre áll az informatikus szakos hallgatók számára az adatbázis-kezelés kiemelkedő minőségű tankönyve. A fordítást végző csapat most is az Eötvös Loránd Tudományegyetem Információs Rendszerek Tanszékének oktatóiból, volt és jelenlegi doktoranduszaiból szerveződött. Köszönet illeti a fordítókat és a lektort az áldozatvállalásért, amivel rövid határidőre elkészítették a fordítást, és így az új tanév kezdetére megjelenhetett az új könyv.

Budapest, 2008. augusztus

Dr. Benczúr András

Előszó

Stanfordban negyedéves, quarter (három hónapos ciklusú) rendszer szerint folyik az oktatás, ennek következtében a bevezető adatbázis-kezelés tantárgyát két kurzusba osztottuk. Az 1. kurzust olyan hallgatók számára terveztük, akik majd használnak adatbázisokat, de nem szükségszerű, hogy adatbázis-kezelő rendszerek megvalósítása területen is fognak munkát vállalni. Ez a kurzus előfeltétele a 2. kurzusnak, ami bevezet az adatbázisok implementálásába. Ezután azok a hallgatók, akik messzebbre akarnak jutni adatbázis-ismeretekben, felvehetik a 3. Speciális témák, a 4. DBMS-megvalósítási projekt és az 5. Tranzakciófeldolgozás és osztott adatbázisok kurzusokat.

1997-ben kezdtük meg egy könyvpár kiadását. Az első, *A First Course in Database Systems*[1] az 1. kurzus számára, a második, *Database Sytem Implementation*[2] a 2. és a 3. egy részéhez készült.

A harmadik kiadásban több új témát is bevezettünk, és bizonyos mértékig az általános szemléleten is változtattunk. Manapság az adatbázis-rendszerekhez két fontos adatmodell emelhető ki: a relációs és a félig-strukturált (XML). Ezért úgy döntöttünk, hogy háttérbe szorítjuk az objektumorientált adatbázisokat, a tervezéssel és az objektumrelációs rendszerekkel kapcsolatos összefüggésük kivételével.

A harmadik kiadás újdonságai

Az első fejezetben található rövid bevezetés után a 2–4. fejezetekben a relációs modellezést fedjük le. A 4. fejezetet a magas szintű modellezésre szántuk. Ebben az E/K-modellhez hozzáfűzve tárgyaljuk az UML-t (Unified Modeling Language) is. Szintén a 4. fejezetbe helyeztük át az ODL-ről írt korábbi anyagok rövidebb változatát úgy tárgyalva, mint a relációs adatbázisséma tervezésének egyik nyelvét. Az ODL és OQL korábbi részletesebb tárgyalását az érdeklődők elérhetik a könyv weblapján angol nyelven.

A funkcionális és többértékű függőségek anyagát módosítottuk, és a 3. fejezetben maradtak. Változtattunk nézőpontunkon úgy, hogy a funkcionális füg-

[1] A könyv magyar fordítása *Adatbázisrendszerek – Alapvetés* címmel jelent meg a Panem kiadó gondozásában 1998-ban.

[2] A könyv magyar fordítása *Adatbázisrendszerek megvalósítása* címmel jelent meg a Panem kiadó gondozásában 2001-ben.

gőségek jobb oldalán attribútumok halmaza szerepel. Bizonyos algoritmusokat explicit módon megadtunk, közöttük a „chase"-t, amelyek lehetővé teszik a függőségek kezelését. Kiegészítettük a harmadik normálforma tárgyalását a 3NF-szintetizáló algoritmussal, és világosabbá tettük, mi az előnye és hátránya a 3NF-nek a BCNF-fel szemben.

Az 5. fejezet a relációs algebra áttekintését tartalmazza az előző kiadásból átvéve, hozzávéve még a régi 10. fejezetből a Datalog tárgyalását (pontosabban egy részét). A Datalogban szereplő rekurzió a könyv weblapjába került át, illetve a rekurzív SQL ennek a könyvnek a 10. fejezetében szereplő tárgyalásával kombinálódott.

A 6–10. fejezeteket az SQL programozási lehetőségeinek bemutatására szántuk, és a korábbi könyv 6., 7., 8. és részben a 10. fejezetének átrendezésének és kiegészítésének tekinthetők. Az indexek és nézettáblák (view-k) ismertetése egy saját 8. fejezetbe került, és ezt az anyagot kiegészítettük olyan fontos témák tárgyalásával, mint a materializált nézettáblák és az indexek automatikus választása.

Az új 9. fejezet a régi 8. fejezetre (beágyazott SQL) épül. Új bevezető szakasszal kezdődik, amely ismerteti a háromrétegű architektúrákat is. Úgyszintén tartalmazza a JDBC kibővített tárgyalását és a PHP új áttekintését.

A 10. fejezetben összegyűjtöttük az SQL több haladó témáját. A jogosultságok tárgyalása a régi 8. fejezetből került ide, míg a rekurzív SQL a régi 10. fejezetből. A fejezet túlnyomó részét a beágyazott relációknak (a régi 4. fejezetből) és az objektumrelációs SQL lehetőségeknek (a régi 9. fejezetből) szenteltük.

Végül a 11. és 12. fejezet az XML-t és az XML-alapú rendszereket tartalmazza. A régi 4. fejezet végén lévő résztől eltekintve, ami a 11. fejezetbe került át, ez az anyag teljesen új. A 11. fejezet a modellezést fedi le a DTD részletes tárgyalásával és az XML-séma (XML Schema) teljesen új anyagával. A 12. fejezetet a programozásra szántuk, és az XPath, XQuery és XSLT bemutatását tartalmazó szakaszokból áll.

A könyv használata

A könyv egy egyszemeszteres, adatbázisrendszerek modellezése és használata témájú kurzus számára igen alkalmas anyagot tartalmaz. Egy negyedéves kurzus esetében, mint az említett 1. kurzusunk, valószínűleg el kell hagyni néhány témakört. A 2–7. fejezeteket tartjuk a kurzus központi anyagának. A megmaradó öt fejezet olyan témákat tartalmaz, amiből biztonsággal lehet tetszés szerint válogatni, bár az a meggyőződésünk, hogy minden hallgatót szembesíteni kell egy kicsit azzal, amit az SQL beágyazása jelent egy-egy szabványos befogadó nyelvbe, amivel a 9. fejezet szakaszai foglalkoznak.

Amennyiben a projektkészítés is része a kurzusnak, átrendezhető a tananyag olyan módon, hogy az SQL-utasítások előbb kerüljenek bevezetésre, mint ahogy a könyvben szerepelnek. Későbbre tehetők olyan témák, mint a függőségek, bár a tervezéshez a hallgatóknak szüksége van a normalizálásra.

Előismeretek

A könyvet közbülső kurzusokon használtuk, amelyet egyaránt felvehettek alapképzésben (BSc) és kezdeti mesterképzésben (MSc) részt vevő hallgatók. A tárgy felvételének formális követelménye a következő másodéves kurzusok teljesítéséből áll: (1) Adatstruktúrák, algoritmusok és diszkrét matematika, (2) Szoftverrendszerek, szoftvertervezés és programozási nyelvek. Az anyag megértéséhez igen fontos, hogy a hallgatók legalább alapjaikban ismerjék a következő témaköröket: algebrai kifejezések és szabályok, logika, elemi adatstruktúrák és keresőfák, objektumorientált programozás alapfogalmai és programozási környezetek. Hiszünk abban, hogy az első év végére a számítástudományi szakot végző hallgatók biztosan elsajátítják a szükséges alapokat.

Feladatok

A könyv terjedelmes feladatrendszert tartalmaz, szinte minden szakasz után szerepel néhány feladat. A nehezebb feladatokat felkiáltójel, a legnehezebbeket pedig kettős felkiáltójel jelöli.

Online gyakorlatok a Gradience rendszerben

Létezik egy online házi feladatokat tartalmazó gyűjtemény (angol nyelven), amely a Gradiance cég által kifejlesztett technológiát használja. A feladatokra beadott megoldásokat a rendszer gyűjti, és nem megfelelő válasz esetén specifikus tanácsokat vagy visszacsatolást ad, ami segít a helyes válasz megadásában.

A Gradience szolgáltatás *megvásárolható* a

```
http://www.prenhall.com/goal
```

webcímen. Azok az oktatók, akik kurzusaikhoz használni kívánják a rendszert, vegyék fel a kapcsolatot Prentice-Hall képviselőjével.

Támogatás a World Wide Weben

A könyv honlapjának címe:

```
http://www-db.stanford.edu/~ullman/fcdb.html
```

Itt további háttéranyagok találhatók angol nyelven. Elérhetővé tesszük minden meghirdetésénél az 1. kurzushoz tartozó aktuális jegyzeteket, beleértve a házi feladatokat, a projekteket és a vizsgakérdéseket. Ugyancsak itt tesszük elérhetővé a második kiadás olyan szakaszait, amelyek kimaradtak a harmadik kiadásból.

Köszönetnyilvánítás

Szeretnénk köszönetet mondani Donald Kossmannak a hasznos megbeszélésekért, különösen amelyek az XML-t és a hozzá kapcsolódó programozási rendszereket érintették. Ugyanúgy Bobbie Cochrane-nak, aki a korábbi verzióban a triggerek szemantikájának megértésében segített.

Igen sokan nyújtottak segítséget számunkra a könyvnek és elődeinek tartalmi és szövegellenőrzésében, illetve kapcsolatba lépve velünk hibák közlésével, amit a könyvekben, illetve a webalapú anyagokban találtak. Örömünkre szolgál, hogy itt mondjunk köszönetet mindnyájuknak:

Marc Abromowitz, Joseph H. Adamski, Brad Adelberg, Gleb Ashimov, Donald Aingworth, Teresa Almeida, Brian Babcock, Bruce Baker, Yunfan Bao, Jonathan Becker, Margaret Benitez, Eberhard Bertsch, Larry Bonham, Phillip Bonnet, David Brokaw, Ed Burns, Alex Butler, Karen Butler, Mike Carey, Christopher Chan, Sudarshan Chawathe, Per Christensen, Ed Chang, Surajit Chaudhuri, Ken Chen, Rada Chirkova, Nitin Chopra, Lewis Church, Jr., Bobbie Cochrane, Michael Cole, Alissa Cooper, Arturo Crespo, Linda DeMichiel, Matthew F. Dennis, Tom Dienstbier, Pearl D'Souza, Oliver Duschka, Xavier Faz, Greg Fichtenholtz, Bart Fisher, Simon Frettloeh, Jarl Friis, John Fry, Chiping Fu, Tracy Fujieda, Prasanna Ganesan, Suzanne Garcia, Mark Gjol, Manish Godara, Seth Goldberg, Jeff Goldblat, Meredith Goldsmith, Luis Gravano, Gerard Guillemette, Himanshu Gupta, Petri Gynther, Jon Heggland, Rafael Hernandez, Masanori Higashihara, Antti Hjelt, Ben Holtzman, Steve Huntsberry, Sajid Hussain, Leonard Jacobson, Thulasiraman Jeyaraman, Dwight Joe, Brian Jorgensen, Mathew P. Johnson, Sameh Kamel, Seth Katz, Pedram Keyani, Victor Kimeli, Ed Knorr, Yeong-Ping Koh, David Koller, Gyorgy Kovacs, Phillip Koza, Brian Kulman, Bill Labiosa, Sang Ho Lee, Younghan Lee, Miguel Licona, Olivier Lobry, Chao-Jun Lu, Waynn Lue, John Manz, Arun Marathe, Philip Minami, Le-Wei Mo, Fabian Modoux, Peter Mork, Mark Mortensen, Ramprakash Narayanaswami, Hankyung Na, Mor Naaman, Mayur Naik, Marie Nilsson, Torbjorn Norbye, Chang-Min Oh, Mehul Patel, Soren Peen, Jian Pei, Xiaobo Peng, Bert Porter, Limbek Reka, Prahash Ramanan, Nisheeth Ranjan, Suzanne Rivoire, Ken Ross, Tim Roughgarten, Mema Roussopoulos, Richard Scherl, Loren Shevitz, June Yoshiko Sison, Man Cho A. So, Elizabeth Stinson, Qi Su, Ed Swierk, Catherine Tornabene, Anders Uhl, Jonathan Ullman, Mayank Upadhyay, Anatoly Varakin, Vassilis Vassalos, Krishna Venuturimilli, Vikram Vijayaraghavan, Terje Viken, Qiang Wang, Mike Wiacek, Kristian Widjaja, Janet Wu, Sundar Yamunachari, Takeshi Yokukawa, Bing Yu, Min-Sig Yun, Torben Zahle, Sandy Zhang.

A megmaradó hibák, természetesen, a mieink.

Stanford, Kalifornia, 2007. július

Jeffrey D. Ullman, Jennifer Widom

1. fejezet

Az adatbázisrendszerek világa

Az adatbázisok ma már alapvető fontosságúak minden szakterületen. Amikor meglátogatjuk valamelyik nagyobb weboldalt, mint amilyen a Google, a Yahoo, az Amazon.com, vagy a kisebb információszolgáltató oldalak százait, a színfalak mögött mindig van egy adatbázis, ami az általunk kért információt kezeli. A cégek szintén adatbázisban tárolják a fontos információikat. Az adatbázisok alapvető eszközként jelennek meg számos tudományos kutatási területen is. Adatbázisban tárolják az összegyűjtött adatokat a csillagászok, az emberi génállomány kutatói, a proteinek orvosi tulajdonságait vizsgáló biokémikusok, valamint számos más tudomány képviselői. Az adatbázisok igazi ereje abból a tudásbeli és technológiai fejlesztésből származik, aminek eredményei az utóbbi évtizedek során egy speciális szoftverben öltöttek testet, amit *adatbázis-kezelő rendszer*nek vagy röviden *ABKR*-nek, illetve egyszerűen csak *adatbázisrendszer*nek hívunk. Egy ABKR hatékony eszköz nagy mennyiségű adatok létrehozására és kezelésére, továbbá lehetővé teszi az adatok hosszú időn át való biztonságos tárolását. Ezek a rendszerek a legbonyolultabb szoftvertípusok közé tartoznak. Ebben a könyvben azt fogjuk bemutatni, hogy hogyan kell az adatbázisokat megtervezni, hogyan lehet különböző nyelveken programokat írni, amelyek kezelik az adatbázist, illetve azt, hogy hogyan kell magát az adatbázis-kezelő rendszert megvalósítani.

1.1. Az adatbázisrendszerek fejlődése

Mi is egy adatbázis? Lényegében egy adatbázis nem más, mint hosszú ideig – gyakran évekig – meglévő információk gyűjteménye. Hétköznapi nyelven az *adatbázis* szó olyan adatok együttesére utal, amelyeket egy *adatbázis-kezelő rendszer* kezel. Egy adatbázis-kezelő rendszerrel szemben a következő elvárásaink vannak:

1. Tegye lehetővé a felhasználók számára, hogy új adatbázisokat hozhassanak létre, és azok *sémáját*, vagyis az adatok logikai struktúráját egy speciális nyelven adhassák meg. Ezt a speciális nyelvet *adatdefiníciós nyelv*nek nevezzük.

2. Engedje meg a felhasználóknak, hogy az adatokat egy megfelelő nyelv segítségével *lekérdezhessék* és módosíthassák. Ezt a nyelvet szokás *lekérdező nyelv*nek vagy *adatmanipulációs nyelv*nek nevezni.

3. Támogassa nagyon nagy mennyiségű (terabájtnyi vagy még több) adat hosszú időn keresztül való tárolását, és tegye lehetővé a hatékony adathozzáférést a lekérdezések és az adatbázis-módosítások számára.

4. Biztosítsa a *tartósság*ot, vagyis az adatbázis helyreállíthatóságát különböző hibák és meghibásodások, valamint szándékos rongálás esetén.

5. Felügyelje a több felhasználó által egy időben történő adathozzáféréseket úgy, hogy az egyes felhasználók műveletei ne legyenek hatással a többi felhasználóra, és az egyidejű adathozzáférések ne vezethessenek az adatok hibássá vagy következetlenné válásához.

1.1.1. Az első adatbázis-kezelő rendszerek

Az első megvásárolható adatbázis-kezelő rendszerek a 60-as évek vége felé kezdtek megjelenni. Ezek a fájlkezelő rendszerekből alakultak ki. A fájlkezelő rendszerek a fenti 3. pontnak részben megfelelnek, mert lehetővé teszik nagy mennyiségű adat hosszú időn keresztül való tárolását. A fájlkezelő rendszerek azonban általában nem garantálják, hogy az adatok nem vesznek el, hacsak nem készítünk róluk biztonsági másolatot, és az adatok hatékony elérését sem támogatják olyan esetekben, amikor az adatelem elhelyezkedése egy állományon belül nem ismert.

A fájlkezelő rendszerek nem támogatják közvetlenül a 2. pontban megfogalmazott elvárást sem, vagyis nem biztosítanak lekérdező nyelvet az adatokhoz. Az adatséma megadására vonatkozó lehetőségeik (1. pont) az állományokat tartalmazó könyvtárszerkezet megadására korlátozódnak. A 5. pontban megfogalmazott elvárást nem mindig támogatja a fájlrendszer, így biztonsági mentés nélkül az adatok elveszhetnek. Végül pedig a 4. pont elvárásait sem elégítik ki a fájlrendszerek. Ha ugyanis megengedik több felhasználó (vagy felhasználói folyamat) konkurens hozzáférését az állományokhoz, általában akkor sem akadályozzák meg az olyan helyzeteket, amikor két felhasználó ugyanazt az állományt körülbelül egy időben módosítja, és így az egyikük által végrehajtott módosítások nem jelennek meg az állományban.

Az adatbázis-kezelő rendszerek első jelentős alkalmazási területei azok a rendszerek voltak, amelyekben sok kis adatelem szerepelt és a rendszerben sok lekérdezés és módosítás volt. A következőkben felsorolunk néhány példát ezek közül:

1. Banki rendszerek: Ezek kezelik a hozzáféréseket, és gondoskodnak arról is, hogy a rendszerhibák ne vezethessenek pénzösszegek eltűnéséhez.

2. Repülőgép-helyfoglalási rendszerek: Ezeknél, hasonlóan a banki rendszerekhez, szintén követelmény annak a biztosítása, hogy adat ne vesszen el, illetve, hogy a vásárlók által kezdeményezett kisebb műveleteket nagyobb mennyiségben elfogadják.

3. Vállalati nyilvántartások: Tárolják a munkavállalói és az adó rekordokat, a készleteiket, az eladási adatokat és még sokféle egyéb információt is. Ezen adatok többsége kritikus jelentőségű.

A korai ABKR-ekben a programozónak kellett az adatokat ábrázolni aszerint, ahogyan azok tárolva voltak. Ezek a rendszerek különböző adatmodelleket használtak az adatbázisban tárolt információk szerkezetének ábrázolásához. A legfontosabb ezen modellek közül a hierarchikus adatmodell – amely egy fa szerkezettel ábrázolta az adatokat –, és a hálós adatmodell, amely egy gráffal ábrázolta az adatokat. Ez utóbbit a 60-as évek végén szabványosították egy CODASYL-jelentésben (Committee on Data Systems and Languages – Adatrendszerekkel és Nyelvekkel Foglalkozó Bizottság)[1].

Ezekkel a korai modellekkel és rendszerekkel az volt a gond, hogy nem támogattak semmilyen magas szintű lekérdező nyelvet. Például a CODASYL lekérdező nyelvnek olyan utasításai voltak, amelyek csak azt engedték meg a felhasználónak, hogy adatelemről adatelemre mozogjon az elemek között meglévő mutatókból álló gráf mentén. Az ilyen programok megírása meglehetősen nagy erőfeszítést igényelt még egyszerű lekérdezések esetén is.

1.1.2. Relációsadatbázis-kezelő rendszerek

Ted Codd 1970-ben publikált egy híres cikket[2], amelynek megjelenése után az adatbázisrendszerek jelentősen megváltoztak. Codd azt javasolta, hogy az adatbázisrendszereknek a felhasználó felé az adatokat táblázatok formájában kellene megjeleníteniük, ezeket a táblákat nevezzük *relációk*nak. A háttérben persze egy bonyolult adatstruktúra is lehet, amelyik lehetővé teszi, hogy a rendszer gyorsan adjon választ a legkülönbözőbb kérdésekre, de ellentétben a korábbi rendszerekkel, egy relációs rendszer felhasználójának nem kell törődnie az adatok tárolási struktúrájával. A lekérdezések egy olyan magas szintű nyelv segítségével fejezhetők ki, amelynek használata jelentős mértékben növeli az adatbázis-programozók hatékonyságát. A könyv nagy részében a relációs modell kérdéseivel fogunk foglalkozni. Az SQL nyelvet (Structured Query Language – Strukturált Lekérdező Nnyelv), a relációs modell relációs algebrán alapuló legfontosabb lekérdező nyelvét, alaposan végignézzük.

[1] *CODASYL Data Base Task Group April 1971 Report*, ACM, New York.

[2] Codd, E. F., „A relational model for large shared data banks," *Comm. ACM*, **13**:6, pp. 377–387, 1970.

1990-re az adatbázisrendszerek közül a relációsadatbázis-kezelők váltak egyeduralkodóvá. Ennek ellenére az adatbázis-kezelés területe továbbra is folyamatosan fejlődik, új kérdések és megközelítések újra meg újra felbukkannak. Az objektumorientált környezetek hatással voltak a relációs modellre. A nagyobb adatbázisok egy része a relációs módszert használóktól jelentősen eltérő módon van megszervezve. E fejezet hátralevő részében áttekintünk néhány mai irányzatot az adatbázisrendszerek témaköréből.

1.1.3. Egyre kisebb rendszerek

Az első adatbázis-kezelők terjedelmes és drága szoftverrendszerek voltak, amelyek nagyméretű gépeken futottak. A méret szükségszerű volt, mert egy gigabájtnyi adat tárolásához nagyméretű számítógépre volt szükség. Manapság sok száz gigabájt ráfér egyetlen lemezre, és az is lehetséges, hogy egy adatbázis-kezelőt személyi számítógépen futtassunk. Így a relációs modellen alapuló adatbázisrendszerek ma már egészen kis számítógépekre is megvásárolhatók. Mára ezek a rendszerek ugyanolyan elterjedt számítógépes eszközökké kezdenek válni, mint korábban a táblázatkezelők és a szövegszerkesztők.

Egy másik meghatározó irányvonal a dokumentumok használata, ez gyakran címkéket használó XML (eXtensible Markup Language – Kiterjeszthető Leíró Nyelv) dokumentumokat jelent. Kisebb dokumentumoknak egy nagyobb méretű gyűjteménye is szolgálhat adatbázisként. A lekérdezési és a módosítási módszerek pedig eltérnek a relációs modellnél megszokottaktól.

1.1.4. Egyre nagyobb rendszerek

Másik oldalról szemlélve a dolgot, egy gigabájtnyi adat nem is olyan sok. A vállalati adatbázisok gyakran több terabájtnyi (10^{12} bájt) helyet foglalnak el. Mostanában már a petabájt (10^{15} bájt) méretű adatmennyiségeket is tárolnak, és ilyen méretekben is dolgozzák fel ezeket. Tekintsünk át néhány lényeges példát:

1. A *Google* több petabájtnyi adatot gyűjtött össze a *webről*. Ezeket az adatokat pedig nem egy hagyományos ABKR-ben tárolják, hanem a keresőmotor lekérdezéseire optimalizált speciális struktúrákban.

2. A műholdak petabájtnyi adatokat küldenek egy specializált rendszernek letárolásra.

3. Egy kép most már többet mond ezer szónál. Míg ezer szót öt- vagy hatezer bájton tudunk tárolni, addig egy képnek a tárolása általában ennél jóval nagyobb helyet igényel. Az olyan tárházak, mint a *Flickr*, több millió képet tárolnak, és lehetővé teszik a keresést is a képek között. Még egy, az *Amazon*hoz hasonló adatbázis is a termékek képeinek millióit szolgáltatja.

4. Sőt, ha a képek ennyi területet használnak fel, akkor a filmek még ennél is többet. Egy egyórás videó legalább egy gigabájtot foglal el. A *YouTube*-hoz hasonló oldalak filmek százezreinek vagy millióinak tárolását és könnyű elérését teszik lehetővé.

5. A *peer-to-peer* fájlmegosztó rendszerek hagyományos számítógépek nagy hálózatait használják az adatok tároláshoz és többféle megosztásához. Habár egy hálózati csúcs csak néhány száz gigabájtnyi adat tárolását teszi lehetővé, a többi csúccsal együtt viszont már hatalmas mennyiséget tárolhatnak.

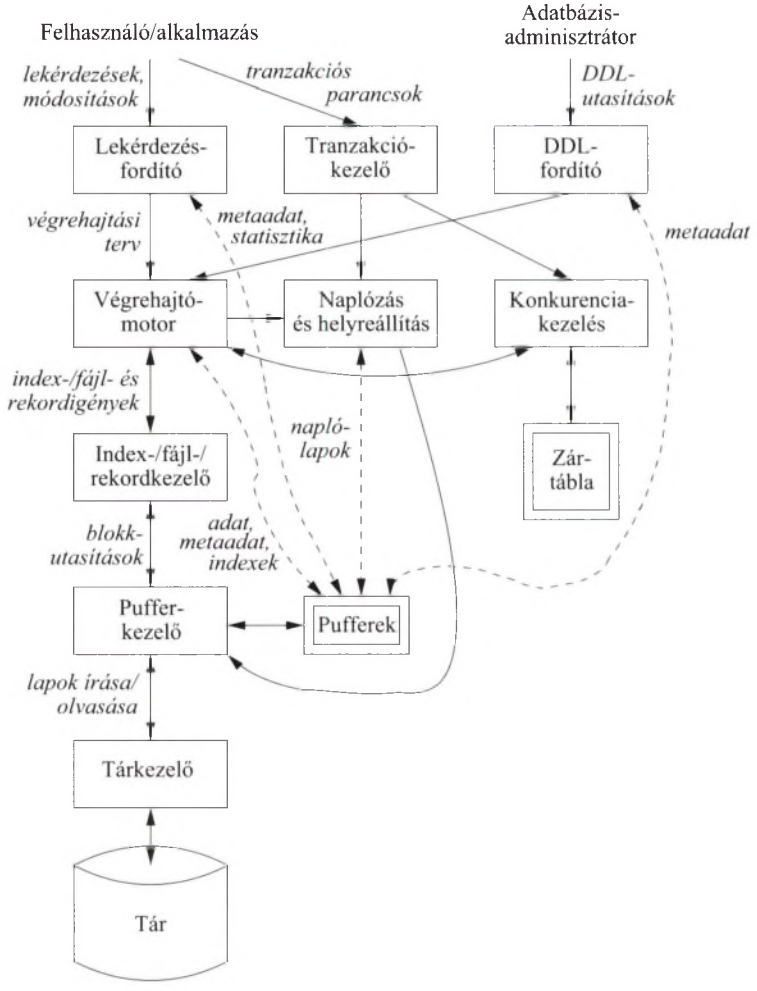
1.1.5. Információk egyesítése

Ahogy az információ szerepe egyre alapvetőbbé válik a munkában és a szórakozásban, azt figyelhetjük meg, hogy a meglevő információforrásokat egyre többféle módon hasznosítjuk. Például egy nagyvállalatnak sok részlege van, amelyek mindegyike külön adatbázissal rendelkezik a saját termékeiről, illetve alkalmazottairól. Természetesen néhány ilyen részleg maga is független vállalat, amelynek megvan a saját módszere a dolgok intézésére. Ezek a részlegek akár különböző ABKR-t, illetve adatstruktúrát is használhatnak. Használhatnak eltérő kifejezéseket ugyanazon dolognak a leírására, vagy azonos kifejezéseket különböző dolgok leírására. A helyzetet tovább rontja az olyan örökölt alkalmazások létezése, melyek az összes adatbázist használják. Így lehetetlenné válik ezen adatbázisok elvetése.

Ennek az eredményeként egyre gyakrabban szükségessé vált egy, a létező adatbázisok feletti struktúra kiépítése. Ezeknek az a célja, hogy a megosztott információkat egységesített formára hozzák. Az egyik ilyen népszerű megközelítés az *adattárház*ak létrehozása, amelyekben a több adatbázisból örökölt információkat a megfelelő átalakítások használatával időszakosan kimásoljuk egy központi adatbázisba. Egy másik megközelítés egy olyan *közvetítő* (vagy *„middleware"*) megvalósítása, aminek a feladata a különböző adatbázisokból származó adatok egyesített modelljének támogatása ezen modell és az egyes adatbázisok aktuális modellje közötti átfordítással.

1.2. Az adatbázis-kezelő rendszerek áttekintése

Az 1.1. ábrán egy ABKR vázlatát láthatjuk. Az egyvonalas téglalapok a rendszer komponenseit jelölik, a kétvonalasok pedig memóriabeli adatszerkezeteket. A folyamatos vonalak a vezérléseket és az adatáramlásokat jelölik, a szaggatott vonalak pedig azt, hogy az adott irányban csak adatok áramlanak. Mivel az ábra meglehetősen bonyolult, a részleteket több lépésben vizsgáljuk. Először tekintsük az ábra tetején az ABKR felé kiadható utasításokat, amelyek két különböző helyről érkezhetnek:



1.1. ábra. Egy adatbázis-kezelő rendszer részei

1. Hagyományos felhasználóktól és alkalmazói programoktól, amelyek lekérdezik vagy módosítják az adatokat.

2. *Adatbázis-adminisztrátor*tól. Ez olyan személyt vagy személyeket jelent, akik felelősek az adatbázis szerkezetéért vagy más szóval *sémá*jáért.

1.2.1. Adatdefiníciós nyelvi utasítások

A második fajta utasításokat egyszerűbb feldolgozni, először az ilyen utasítások feldolgozási útvonalát mutatjuk be az 1.1. ábra jobb felső sarkából kiindulva. Tegyük fel például, hogy egy egyetemi nyilvántartó adatbázis-adminisztrátora vagy röviden *DBA*-ja (a DBA az angol Database Administratorból származó rövidítés) elhatározza, hogy létre kell hozni egy olyan táblát, amelynek oszlopaiban a hallgatót, a hallgató által felvett kurzusokat és a kurzuson szerzett érdemjegyet tároljuk. A DBA azt is eldöntheti, hogy a lehetséges érdemjegyek köre A, B, C, D és F. Ezek a szerkezetre és megszorításra vonatkozó információk mind az adatbázis sémájának részét képezik. Azt láthatjuk az 1.1. ábrán, ahogyan a DBA elküldi az utasításokat a rendszernek. A DBA-nak speciális jogosultságokkal kell rendelkeznie ahhoz, hogy a sémamódosító utasításokat végrehajthassa, hiszen ezek alapvető hatással lehetnek az adatbázisra. Ezeket a sémamódosító utasításokat DDL-utasításoknak nevezzük (a rövidítés az angol Data Definition Language-re utal, amely adatdefiníciós nyelvet jelent). Egy DDL-feldolgozó elemzi őket, majd továbbítja a végrehajtómotorhoz, ahonnan az index-/fájl-/rekordkezelőhöz kerülnek, majd ezután megtörténik a *metaadatok* módosítása, amely információk az adatbázis sémájának leírására szolgálnak.

1.2.2. A lekérdezések végrehajtásának áttekintése

Az ABKR-nek küldött utasítások többsége az 1.1. ábra bal oldalán látható útvonal mentén hajtódik végre. Egy felhasználó vagy egy alkalmazói program művelet kezdeményezésére az adatmanipulációs nyelvet (vagy az angol Data Manipulation Language alapján a DML-t) használják. Ez az utasítás nem érinti az adatbázis sémáját, de érintheti a tartalmát (amennyiben módosító utasításról van szó), vagy adatot kér az adatbázisból (lekérdező utasítás esetén). A DML-utasításokat két különálló alrendszer kezeli, amelyeket a következőkben mutatunk be.

Lekérdezések megválaszolása

A lekérdezéseket a *lekérdezésfordító* elemzi és optimalizálja. Az eredményül kapott *végrehajtási terv*, ami az ABKR által végrehajtandó elemi műveletek sorozata, a végrehajtómotorhoz kerül. A végrehajtómotor olyan utasítások sorozatát adja ki az erőforrás-kezelő felé, amelyek kis adatmennyiségekre, általában a relációk néhány sorára vonatkoznak. Az erőforrás-kezelő ismeri a relációkat tároló *adatfájl*okat, a fájlokon belüli rekordok méretét és formátumát, valamint az *indexfájl*okat, amelyek segítségével az adatfájlok elemei gyorsan megtalálhatók.

Az adatkéréseket a *pufferkezelő* kapja meg. A pufferkezelő szerepe az, hogy a másodlagos tárolón (ez általában a lemez) tartósan tárolt adatokat a memóriabeli pufferekbe megfelelő adagokban beolvassa. Általában lapnak vagy „lemezblokknak" nevezzük a lemez és a memóriapufferek között mozgatott adatok mennyiségi egységét.

A pufferkezelő a tárkezelővel kommunikál az adatok lemezről való beolvasásának céljából. A tárkezelő kiadhat operációs rendszer szintű utasításokat is, de gyakoribb megoldás, hogy az ABKR közvetlenül ad utasításokat a lemezvezérlőnek.

Tranzakciók feldolgozása

A lekérdezések és más DML-műveletek *tranzakciók*ba vannak csoportosítva. Ezek olyan egységeket jelölnek, amelyeket atomosan és egymástól elkülönítve kell végrehajtani. Bármely lekérdezés vagy módosító művelet önmagában alkot egy tranzakciót. Ezenkívül a tranzakciók végrehajtásának *tartós*nak kell lennie, ami annyit jelent, hogy egy befejezett tranzakció eredményének meg kell őrződnie még abban az esetben is, ha a tranzakció befejeződése után közvetlenül valamiféle rendszerhiba fordul elő. A tranzakciófeldolgozót két nagyobb részre oszthatjuk, amelyek a következők:

1. *konkurenciakezelő* vagy más néven *ütemező*, amely a tranzakciók atomosságáért és elkülönítéséért felelős, valamint

2. *naplózás- és helyreállítás-kezelő*, amely a tranzakciók tartósságáért felelős.

1.2.3. A tárkezelő és a pufferkezelő

Az adatbázis adatai általában másodlagos tárolón találhatók, ami a mai számítógépes rendszerek esetében többnyire mágneslemezt jelent. Ahhoz azonban, hogy az adatokon bármilyen műveletet végezhessünk, azoknak a memóriában kell lenniük. A *tárkezelő* feladata az adatok lemezen való elhelyezkedésének, valamint a lemez és memória közötti adatmozgatásoknak a felügyelete.

Egy egyszerű adatbázisrendszerben a tárkezelő lehetne az operációs rendszer fájlkezelő része is, de a hatékonyság érdekében az adatbázis-kezelők általában közvetlenül felügyelik az adatok lemezen való tárolását, legalábbis bizonyos körülmények között. A *tárkezelő* tartja nyilván a fájlok lemezen való elhelyezkedését, és a pufferkezelő igénye szerint ő állítja elő egy adott fájl blokkját vagy blokkjait.

A *pufferkezelő* feladata a memóriának pufferekre való felosztása. A pufferek lapméretű tartományok, amelyekbe a lemezblokkok beolvashatók. Így az ABKR-nek minden komponense, amelynek lemezen lévő információra van szüksége, kapcsolatba kerül a pufferekkel és a pufferkezelővel, vagy közvetlenül, vagy a végrehajtómotoron keresztül. A különböző komponenseknek a következő típusú információkra lehet szükségük:

1. *Adat:* Magának az adatbázisnak a tartalma.

2. *Metaadat:* Az adatbázis sémája, ami leírja az adatbázis szerkezetét, valamint a rajta definiált megszorításokat.

3. *Napló rekordok:* Az adatbázison végzett módosításokról szóló információkat tartalmazzák. Az adatbázis tartósságának megőrzését támogatják ezekkel.

4. *Statisztikák:* Az ABKR által összegyűjtött és tárolt információk az adatok tulajdonságairól, például a relációk méretéről és a bennük előforduló értékekről, valamint az adatbázis további komponenseiről.

5. *Indexek:* Olyan adatszerkezetek, amelyek a hatékony adatelérést támogatják.

1.2.4. Tranzakciók feldolgozása

Teljesen általános dolog, hogy egy vagy több adatbázis-műveletet *tranzakció*ba csoportosítunk, ami olyan egységet alkot, amit a rendszernek atomosan kell végrehajtania, valamint az egyes tranzakciókat látszólag egymástól függetlenül kell futtatnia. Ezenkívül az ABKR garantálja még a tartósságot, vagyis azt, hogy egy befejezett tranzakció eredménye nem fog elveszni. A *tranzakciókezelő* tehát fogadja az alkalmazástól a tranzakció utasításait, ami megmondja, hogy egy tranzakció mikor kezdődik és mikor ér véget, továbbá információkat kap az alkalmazás egyéb elvárásairól (előfordulhat például, hogy nem igényli az atomosságot). A tranzakciókezelő a következő feladatokat végzi:

1. *Naplózás:* A tartósság biztosításának érdekében minden adatbázismódosítás külön naplózódik a lemezen. A *naplókezelő* egy olyan eljárást használ, amelynek segítségével bármikor következzék is be egy rendszerhiba, a *helyreállítás-kezelő* képes lesz a változások naplóbejegyzéseinek vizsgálatával az adatbázist egy konzisztens állapotba visszaállítani. A naplókezelő először a bejegyzéseket pufferekbe írja, majd a pufferkezelővel egyezteti, hogy a pufferek a megfelelő időpontokban lemezre legyenek írva (ahol túlélik a rendszerösszeomlást).

2. *Konkurenciakezelés:* A tranzakcióknak úgy kell tűnniük, mintha elkülönítve futnának. A legtöbb rendszerben azonban a valóságban sok tranzakció hajtódik végre egy időben. Ezért az ütemezőnek (konkurenciakezelőnek) kell azt biztosítania, hogy a tranzakciók egyedi műveletei olyan sorrendben hajtódjanak végre, hogy azok összhatása ugyanaz legyen, mintha a tranzakciók valóban egymás után futottak volna le. Az ütemezők ezt a feladatot jellemzően úgy végzik el, hogy az adatbázis bizonyos részeire *zár*akat helyeznek el. A zárak megakadályozzák azt, hogy két tranzakció úgy férjen hozzá egy adatelemhez, ahogyan az nem megengedett. A zárak

A tranzakcióktól elvárt tulajdonságok[a]

A helyesen megvalósított tranzakcióktól elvárjuk, hogy teljesítsék az alábbi követelményeket:

- *Atomosság:* Ez azt jelenti, hogy a tranzakciónak vagy az összes utasítása hajtódjon végre, vagy egyetlen utasítása se hajtódjon végre.
- *Elkülönítés:* Ez azt jelenti, hogy az egyes tranzakcióknak látszólag úgy kell végrehajtódniuk, mintha a lefutásuk ideje alatt egyetlen más tranzakció sem futna.
- *Tartósság:* Ez azt jelenti, hogy ha egy tranzakció befejeződött, akkor az általa végrehajtott utasítások eredménye már semmilyen körülmények között nem veszhet el.

Szokás még egy tulajdonságot említeni, nevezetesen a „*konzisztenciát*". Minden adatbázisban vannak konzisztenciára vonatkozó megszorítások, amelyek az adatelemek közötti kapcsolatokra vonatkozó elvárásainkat fejezik ki. Ilyen lehet például, hogy a számlaegyenlegek ne lehessenek negatívak. A tranzakcióktól elvárjuk, hogy ezeket a konzisztenciafeltételeket megőrizzék.

[a] Az angol szakirodalom ezt ACID-tulajdonságoknak nevezi (A = atomicity, C = consistency, I = isolation, D = durability). *(A szerkesztő megjegyzése.)*

általában a memóriában tárolódnak egy úgynevezett *zártábl*ában, ahogyan azt az 1.1. ábrán is láthatjuk. Az ütemező oly módon befolyásolja a lekérdezések és egyéb adatbázis-műveletek végrehajtását, hogy nem engedi a végrehajtómotor számára a hozzáférést az adatbázis zárolt részeihez.

3. *Holtpontfeloldás:* Miközben a tranzakciók az ütemező által biztosított zárak segítségével versengenek az erőforrásokért, előállhat egy olyan helyzet, hogy egyik tranzakció sem tud továbblépni, mert olyan erőforrásra van szüksége, amit valamelyik másik tranzakció zárol. Ilyenkor a tranzakciókezelő feladata, hogy közbelépjen és megszakítson egy vagy több tranzakciót annak érdekében, hogy a többiek tovább tudjanak dolgozni.

1.2.5. A lekérdezésfeldolgozó

Az ABKR-nek az a része, amelyik a legnagyobb hatással van a felhasználó által érzékelhető hatékonyságra, a lekérdezésfeldolgozó. Az 1.1. ábrán a lekérdezésfeldolgozó két összetevőjét ábrázoltuk:

1. A *lekérdezésfordító* fordítja le a lekérdezést egy belső formára, amit végrehajtási tervnek nevezünk. Ez a terv az adatokon végrehajtandó műveletek sorozata. Általában ezek a műveletek a „relációs algebra" műveleteinek megvalósításai, amely műveletekről majd a 2.4. szakaszban lesz szó. A lekérdezésfordító három főbb részből áll:

   a) egy *lekérdezéselemző*ből, ami a lekérdezés szövegéből egy fa struktúrát épít fel;

   b) egy *lekérdezés-előfeldolgozó*ból, ami szemantikai ellenőrzéseket végez a lekérdezésen (például ellenőrzi, hogy a lekérdezésben előforduló relációk valóban léteznek-e), majd átalakításokat végez az elemzőfán úgy, hogy abból egy algebrai műveleteket tartalmazó kezdeti végrehajtási terv legyen;

   c) egy *lekérdezésoptimalizáló*ból, ami a kezdeti végrehajtási tervet átalakítja a lehető legjobb tervvé, vagyis műveletsorozattá, a tényleges adatok figyelembevételével.

   A lekérdezésfordító a várhatóan leggyorsabb műveletsorozat megtalálásához az adatokra vonatkozó metaadatokat és statisztikákat használja. Például egy *index* létezése az egyik tervet sokkal gyorsabbá teheti egy másik tervnél. Az index egy speciális adatszerkezet, ami lehetővé teszi az adatok gyors elérését, feltéve, hogy egyes adatelemeket ismerünk.

2. A *végrehajtómotor* felelős a kiválasztott végrehajtási terv egyes lépéseinek végrehajtásáért. A végrehajtómotor az ABKR legtöbb komponensével valamilyen kapcsolatba lép, vagy közvetlenül, vagy a pufferekeren keresztül. Ahhoz, hogy bármit tehessen az adatokkal, azokat az adatbázisból a pufferekbe kell bekérnie. Egyeztetnie kell az ütemezővel, nehogy zárolt adatokhoz férjen hozzá, továbbá a naplókezelővel, hogy biztos lehessen benne, hogy minden adatbázisbeli változás megfelelően naplózva lett.

1.3. Adatbázisrendszerekkel kapcsolatos ismeretek áttekintése

Az adatbázisok tanulmányozását öt fő részre osztottuk.[3] Ez az alfejezet áttekintést ad arról, hogy mikre lehet számítani az egyes részeknél:

1. rész. Relációs adatbázisok modellezése

A relációs modell elhanyagolhatatlan jelentőségű az adatbázisrendszerek vizsgálatánál. Az alapvető fogalmak vizsgálata után elmélyedünk a relációs adatbázisok elméletében. Ez a tanulmány magában foglalja a *funkcionális függő-*

[3] A 4. és az 5. rész tárgyalása a következő könyvben található: H. Garcia-Molina, J. D. Ullman, J. Widom: *Adatbázisrendszerek megvalósítása*, Panem, Budapest, 2001, 2008.

*ség*eket, azaz egy adatnak egy másikkal történő egyértelmű meghatározásának kifejezésére szolgáló formális módszert. Emellett tartalmazza a *normalizálás*t is, vagyis egy olyan eljárást, amellyel tökéletesíthető egy relációs adatbázis tervezete a funkcionális függőségek és egyéb formális függőségek felhasználásával. Áttekintünk magasabb szintű tervezési jelöléseket is. Ezek a módszerek tartalmazzák: az egyed-kapcsolat (E/K) modellt, az Egységes Modellező Nyelvet (vagyis az UML-t), illetve az Objektum Definíciós Nyelvet (azaz az ODL-t). Ezeknek a módszereknek az a célja, hogy a tervezési kérdések informális vizsgálatát lehetővé tegyék, mielőtt a tervet egy relációs ABKR felhasználásával megvalósítanánk.

2. rész. Relációs adatbázisok programozása

Ezután rátérünk a relációs adatbázisok lekérdezésének és módosításának miként jére. Az algebrán (*relációs algebrán*) és a logikán (*Datalog*on) alapuló absztrakt programozási nyelvek bevezetése után a relációs adatbázisok szabvány szerinti nyelvével, az SQL-lel fogunk foglalkozni. Tanulmányozzuk az alapokat és olyan fontos speciális témákat is, mint: a megszorítások specifikálását és a *trigger*eket (azaz aktív adatbáziselemeket), az indexeket és egyéb teljesítménynövelő struktúrákat, az SQL tranzakcióba osztását, illetve az adatbiztonságot és titkosságot az SQL-ben.

Megtárgyaljuk azt is, hogy hogyan használható az SQL kész rendszerekben. Gyakran az SQL-t kombinálják valamilyen hagyományos vagy befogadó nyelvvel azért, hogy adatot lehessen cserélni SQL-hívásokkal az adatbázisok és a hagyományos programok között. Több módszert is áttekintünk az ilyen kapcsolatok létrehozására, ezeken belül a beágyazott SQL-t, a folyamatosan tárolt eljárásokat (PSM), a hívásszintű felületet (CLI), a Java adatbázis kapcsolódását (JDBC) és a PHP-t.

3. rész. Félig-strukturált adatok modellezése és programozása

A *web* egyik hatása eredményezte a hierarchikusan strukturált adatoknak a kezelését, ugyanis a *web* szabványa beágyazott címkeelemeken (*félig-strukturált adat*okon) alapul. Bevezetjük az XML-t és a hozzátartozó sémadefiníciós jelöléseket, vagyis a dokumentumtípus definíciót (DTD), és az XML-sémát. Az XML három lekérdező nyelvét tekintjük át: az XPATH-t, az XQueryt és a kiterjeszthető nyelvi átalakítást (XSLT).

4. rész. Adatbázisrendszerek megvalósítása

A *tároláskezelés* vizsgálatával kezdünk: hogyan szervezhető meg az adat hatékony hozzáférése egy merevlemez-központú tárolással. Ismertetjük az adatok kezelésére leggyakrabban használt B-fákat, azaz a merevlemez blokkjainak egy kiegyensúlyozott fáját, illetve más, a többdimenziós adatok kezelésére alkalmas speciális sémákat.

Ezt követően a *lekérdezés feldolgozására* fektetjük a hangsúlyt. Ennek a vizsgálatnak két része lesz. Először meg kell ismernünk a lekérdezés végrehajtását, vagyis az olyan algoritmusokat, amelyeket a lekérdezéseket felépítő műveletek megvalósításánál használnak. Mivel az adat általában a merevlemezen van, így az algoritmusok is el fognak térni azon elvárásoktól, amelyek ugyanezen problémáknak a memóriában lévő adatokra történő elemezésénél vannak jelen. A második lépés lesz a *lekérdezésfordítás*. Itt tanulmányozni fogjuk, hogy hogyan választhatunk ki egy olyan hatékony lekérdezési tervet a sok közül, amellyel az adott lekérdezés végrehajtható.

Ezután a *tranzakciófeldolgozás* vizsgálatával foglalkozunk. Itt több szálon is el lehet indulni. Az egyik ezek közül a *naplózás*, amely az ABKR működésével kapcsolatos megbízható rekordok kezelésével foglalkozik a rendszer leállást okozó események utáni *visszaállíthatóság* céljából. Egy másik ilyen szál az *ütemezés*, vagyis az események tranzakciókra bontásának a vezérlése, amire az ACID-tulajdonságok megőrzése miatt van szükség. Foglalkoznunk kell még a holtpontok kezelésének miként jével, illetve módosítanunk kell azon algoritmusainkat, amelyek akkor szükségesek, amikor egy tranzakciót szét akarunk osztani több, egymástól független helyre.

5. rész. Modern adatbázisrendszerek kérdései

Ebben a részben megnézünk egy pár olyan adatbázis-technológiát, amely a hagyományos, relációs ABKR-ek világában fontos szerepet játszik. Megnézzük, hogy hogyan működnek a *keresőmotorok* és azok a speciális adatstruktúrák, amelyek lehetővé teszik a műveletek elvégzését. Áttekintjük az információ egyesítését, illetve az adatbázisok közötti láthatatlan adatmegosztásnak a metodikáját. Az *adatbányászat* egy olyan vizsgálat, amely jó néhány érdekes és fontos algoritmust tartalmaz nagy mennyiségű adat összetett feldolgozására nézve. Az *adatfolyamrendszerek* olyan adattal foglalkoznak, amely folyamatosan érkezik a rendszerbe, és amelynek a lekérdezéseire folyamatosan, időtől függő módon érkezik a válasz. A *peer-to-peer* rendszerek sok kihívást állítanak elénk a független kiszolgálókon tárolt adat megosztásának kezelésére nézve.

1.4. Irodalomjegyzék

Ma már az online kereshető irodalomjegyzékekben lényegében minden újabb cikket megtalálunk, ami az adatbázisrendszerekről szól. Ezért a könyvben nem törekszünk majd a hivatkozások teljes felsorolására, hanem elsősorban a történetileg jelentős cikkeket fogjuk megemlíteni, valamint a legfontosabb áttekintő munkákat. Az egyik adatbázisokról szóló cikkeket tartalmazó kereshető tárgymutatót Michael Ley hozta létre [5]. Alf-Christian Achilles szintén karbantart egy ilyen könyvtárat, amelyben számos, a témába vágó szócikk található [3].

Noha az adatbázisrendszereknek számos prototípus-megvalósítása járult hozzá a technológiai fejlődéshez, a két legismertebb ezek közül a System R projekt volt az IBM almadeni kutatóközpontjában [4], valamint az INGRES

projekt a Berkeley egyetemen [7]. Mindkettő egy korai relációs rendszer volt, amelyek segítettek abban, hogy ez a technológia vált egyeduralkodóvá az adatbázisok körében. Számos olyan cikket találhatunk [6]-ban, amelyek alapvetően meghatározták e terület irányát.

Az adatbázisrendszerekkel kapcsolatos kutatásokról és irányzatokról szóló tanulmányok sorában a 2003-as „Lowell-tanulmány" [1] tekinthető a legújabbnak. Ebben számos hivatkozást találunk korábbi, hasonló jellegű tanulmányokra.

Az adatbázisrendszerek elméletéről további, e könyvben nem érintett ismeretek találhatók [2]-ben és [8]-ban.

[1] S. Abiteboul et al., „The Lowell database research self-assessment," *Comm. ACM* **48**:5 (2005), pp. 111–118.
`http://research.microsoft.com/~gray/lowell/LowellDatabaseResearchSelfAssessment.htm`

[2] S. Abiteboul, R. Hull, V. Vianu, *Foundations of Databases*, Addison-Wesley, Reading, MA, 1995.

[3] `http://liinwww.ira.uka.de/bibliography/Database`

[4] M. M. Astrahan et al., „System R: a relational approach to database management," *ACM Trans. on Database Systems* **1**:2, pp. 97–137, 1976.

[5] `http://www.informatik.uni-trier.de/~ley/db/index.html`
Ugyanez a következő tükrözött oldalon is megtalálható:
`http://www.acm.org/sigmod/dblp/db/index.html`

[6] M. Stonebraker, J. M. Hellerstein (eds.), *Readings in Database Systems*, Morgan-Kaufmann, San Francisco, 1998.

[7] M. Stonebraker, E. Wong, P. Kreps, G. Held, „The design and implementation of INGRES," *ACM Trans. on Database Systems* **1**:3, pp. 189–222, 1976.

[8] J. D. Ullman, *Principles of Database and Knowledge-Base Systems, Volumes I and II*, Computer Science Press, New York, 1988, 1989.

I. rész

Relációs adatbázisok modellezése

2. fejezet

A relációs adatmodell

A fejezet során bevezetjük a legfontosabb adatmodellt: a kétdimenziós táblát vagy más néven a „relációt". A tárgyalást az adatmodellek általános áttekintésével kezdjük. Megadjuk a relációk legfontosabb fogalmait és jelöléseit és bemutatjuk, hogy hogyan használható a modell a legismertebb adatformák leírására. Mindezek után bemutatjuk az SQL nyelv azon részét, amely a relációk, illetve a relációk struktúrájának megfogalmazására szolgál. A fejezetet a relációs algebra tárgyalásával zárjuk. Látni fogjuk, hogy ez a jelölés egyaránt alkalmas lesz mind egy lekérdező nyelvnek, azaz az adatmodell olyan aspektusának, mely kérdések megfogalmazását teszi lehetővé számunkra az adatokon, mind egy megszorításos nyelvnek, vagyis egy olyan adatmodell-szemléletnek, amellyel többféle módon is megszorításokat tehetünk az adatbázisban szereplő adatokra.

2.1. Adatmodellek áttekintése

Az adatbázisrendszerek vizsgálata során az egyik legfontosabb fogalom az „adatmodell". A fogalom rövid összefoglalása folyamán definiálunk néhány alapvető terminológiát, és megemlítjük a legfontosabb adatmodelleket is.

2.1.1. Mi az adatmodell?

Az *adatmodell* információ vagy adatok leírására szolgáló jelölés. A leírás általában az alábbi három részből áll:

1. *Az adat struktúrája.* Nagy valószínűséggel az olvasó már találkozott ilyen eszközökkel valamilyen a C-hez vagy Javához hasonló programozási nyelvben íródott program által használt adatstruktúrák leírásánál, mint például a tömbök, a („struct" kulcsszóval megadott) struktúrák vagy az objektumok. Az adat számítógépen történő implementálása során használt adatstruktúrákat az adatbázisrendszerek világában *fizikai adatmodell*nek nevezik, jóllehet valójában távol állnak az elektronok és kapuk világától,

amelyek az adat valódi fizikai implementációját szolgálnák. Az adatbázisok világában az adatmodellek magasabb szinten vannak, mint az adatstruktúrák. Éppen ezért gyakran *fogalmi modell*ként hivatkoznak rájuk a megkülönböztethetőség miatt. Hamarosan példákat is mutatunk.

2. *Az adaton végezhető műveletek.* A programozási nyelvekben az adatokon végezhető műveletek általában bármit jelenthetnek, ami programozással kivitelezhető. Az adatbázis adatmodelljeinél viszont leggyakrabban egy véges műveletalmazt jelentenek, amely a végrehajtható műveleteket tartalmazza. Általában *lekérdezés*ek (információkat visszaadó műveletek) és *módosítás*ok (az adatbázist megváltoztató műveletek) egy korlátos számú halmazának végrehajtása engedélyezett. Ez a korlátozás inkább erősségnek tekinthető, mint gyengeségnek, hiszen a műveletek korlátozása olyan magas szintű adatbázis-operációk megfogalmazását teszi lehetővé a programozó részére, amelyeket az adatbázis kezelőrendszere már eleve hatékonyan implementált. Az összehasonlíthatóság kedvéért, a legelterjedtebb programozási nyelvekben (mint például a C) nem mindig lehetséges optimalizálni egy programot. Például egy nem túl hatékony algoritmus (mint például a buborékrendezés) esetén a programot át kell írnunk egy hatékonyabbra (például gyorsrendezésre).

3. *Az adatra tett megszorítások.* Az adatbázis adatmodelljei általában lehetővé teszik, hogy megszorításokat fogalmazhassunk meg arra nézve, hogy milyen adatokat engedélyezünk. Ezek a megszorítások a legegyszerűbb intervallumoktól az összetettebb korlátozásokig bármik lehetnek. (Például: „a hét napjai 1 és 7 közötti egész számok" vagy „egy filmnek csak egy címe lehet".) Ezekkel részletesen foglalkozunk a 7.4., illetve a 7.5. alfejezetek során.

2.1.2. Fontos adatmodellek

Jelenleg a két legnagyobb jelentőséggel bíró adatmodell a következő:

1. A relációs modell, amely magában foglalja az objektumrelációs kiterjesztést is.

2. A félig-strukturált adatmodell, amely tartalmazza az XML-t és a hozzá tartozó szabványokat is.

Az első pontban szereplő jelen fejezetnek a tárgya, és a forgalomban lévő összes adatbázis-kezelő rendszer ezt használja. A félig-strukturált modell, amelynek a fő megjelenési formája az XML, csak egy hozzáadott környezet a legtöbb relációs ABKR-ben, és így több különböző megvalósítási formája létezik. Ezt az adatmodellt a 11. fejezetben tárgyaljuk részleteiben.

2.1.3. A relációs modell vázlatosan

A relációs modell táblákon (például lásd a 2.1. ábrán) alapul. Ezt a modellt a 2.2. alfejezetben tárgyaljuk majd. A példában szereplő reláció vagy tábla filmeket ír le: a címüket, a gyártási évüket, a perc alapú hosszukat és a műfajukat. Jelen esetben három sor látható, de feltehető, hogy ennél sokkal több sor szerepel benne. Képzeljük el, hogy minden valaha elkészített filmhez egy sorbejegyzés tartozik.

filmcím	*év*	*hossz*	*műfaj*
Elfújta a szél	1939	231	dráma
Csillagok háborúja	1977	124	sci-fi
Wayne világa	1992	95	vígjáték

2.1. ábra. Egy példareláció

A relációs modell egy része hasonlít a C programozási nyelv olyan tömbstruktúrájához, amelyben az oszlopnevek a mezők neveinek felelnek meg, és amelyben minden sor a tömb egy struktúrájának az értékeit reprezentálja. Hangsúlyoznunk kell viszont, hogy ez utóbbi fizikai megvalósítás csupán egy lehetséges mód a tábla fizikai szerkezetének megvalósítására. Valójában nem így szokták reprezentálni a relációkat, és az adatbázisrendszerek tanulmányozásának egyik része pontosan az ilyen táblák helyes ábrázolásáról szól. A különbségek nagy részben a reláció természetéből fakadnak: normál esetben nem a fő memória struktúrájaként vannak megvalósítva, illetve a megfelelő fizikai megvalósításuk során azt is figyelembe kell vennünk, hogy olyan nagy méretű relációk hozzáférését kell lehetővé tenniük, amelyek a merevlemezen vannak tárolva.

A műveletek főként a (2.4. alfejezetben tárgyalásra kerülő) „relációs algebrán" alapuló relációs modellel vannak kapcsolatban. Ezek a műveletek táblaorientáltak. Megkérdezhetjük például egy reláció összes soráról, hogy rendelkeznek-e egy adott értékkel valamely adott oszlopukban. Például a 2.1. ábra táblájára lekérdezhetjük az összes „vígjáték" műfajú sort.

A relációs adatmodell megszorításokkal foglalkozó részét érintjük a 2.5. alfejezetben, és később a 7. fejezetben részleteiben is tárgyalni fogjuk. Egy rövid példa lehet viszont az általában használt megszorítások fajtáira, hogy ha a filmműfajokat egy rögzített lista elemeiből akarjuk venni, akkor az utolsó oszlop értékei csak ebből a listából kerülhetnek majd ki. Vagy azt is eldönthetjük (helytelenül, mint látni fogjuk), hogy két filmnek nem lehet ugyanaz a címe. Ez utóbbi megszorítás azt jelenti a táblára nézve, hogy semelyik két sornak nem lehet azonos az első komponensben lévő karaktersorozata.

2.1.4. A félig-strukturált modell vázlatosan

A félig-strukturált adat inkább fákhoz, illetve gráfokhoz hasonlítható, mint táblákhoz vagy tömbökhöz. Manapság ennek a nézőpontnak az elsődleges felhasználási területe az XML, amely egy hierarchikusan beágyazott, címkézett elemeket jellemző adatábrázolási módszert jelent. A címkék (vagy *tag*ek) – a HTML nyelvben használtakhoz hasonlóan – a különböző adatok szerepét jelölik ugyanúgy, mint ahogyan a relációs adatmodell tábláinál az oszlopok fejlécei. Például a 2.1. ábrán lévő adatok lehetnének egy XML-dokumentumban is. Ez látható a 2.2. ábrán.

```
<Filmek>
    <Film cím="Elfújta a szél">
        <Év>1939</Év>
        <Hossz>231</Hossz>
        <Műfaj>dráma</Műfaj>
    </Film>
    <Film cím="Csillagok háborúja">
        <Év>1977</Év>
        <Hossz>124</Hossz>
        <Műfaj>sci-fi</Műfaj>
    </Film>
    <Film cím="Wayne világa">
        <Év>1992</Év>
        <Hossz>95</Hossz>
        <Műfaj>vígjáték</Műfaj>
    </Film>
</Filmek>
```

2.2. ábra. A filmek XML-dokumentuma

A félig-strukturált adaton végezhető műveletek általában a megfelelő fa egy eleme és a fa egy vagy több másik beágyazott részeleme közötti rákövetkező utakkal dolgoznak, majd a részelemek beágyazott elemeivel folytatják a feldolgozást és így tovább. Például, ha a legkülső `<Filmek>` elemmel kezdünk (a teljes XML-dokumentum a 2.2. ábrán látható), akkor átléphetünk bármelyik beágyazott `<Film>` elembe, ahol minden elem egy `<Film>` és a neki megfelelő `</Film>` címke között van. Majd bármely `<Film>` elemből átléphetünk a beágyazott `<Műfaj>` elembe, ahol már látni fogjuk, hogy melyik film lesz „vígjáték".

Ennek az adatmodellnek az adatstuktúráján tett megszorítások gyakran egy címkéhez tartozó adattípus értékeire vonatkoznak. Például: „A `<Hossz>` címkéhez tartozó értékek egész számok vagy tetszőleges karakterláncok legyenek?". Egy másik megszorítással azt határozhatjuk meg, hogy egy címke melyik másik címkének lehet a beágyazott tagja. Például: „Minden `<Film>` elemnek kell hogy

legyen egy beágyazott <Hossz> eleme?", „A 2.2. ábra mely egyéb címkéjét kell használnunk a <Film> elemeken belül?" vagy „Egy filmnek lehet több műfaja is?". Ezeket és más egyéb problémákat is megtárgyalunk a 11.2. alfejezetben.

2.1.5. Egyéb adatmodellek

Több, az eddigiektől eltérő modellt is kapcsolatba hoznak, illetve hoztak az ABKR-ekkel. Egy modern irányvonala ezeknek, hogy a relációs adatmodellt kiegészítik ojektumorientált környezetekkel is. Az objektumorientáltság hatása a relációkra a következő két pontban foglalható össze:

1. Az értékeknek struktúrája is lehet, azaz nem csak olyan elemi típusokat enged meg, mint a 2.1. ábrán szereplő karakersorozat, illetve egész szám.

2. A relációkhoz tartozhatnak metódusok is.

Ezen kiterjesztésnek, amelyet *objektumrelációs modell*nek nevezünk, a szelleme nagyon hasonlít ahhoz a módhoz, ahogyan a C struktúráit objektumokká terjesztették ki a C++-ban. Az objektumrelációs modellt a 10.3. alfejezetben vezetjük be.

Vannak tisztán objektumorientált adatbázismodellek is. Ezekben a reláció már nem az adatstruktúra legfontosabb fogalma, hanem csak egy lehetséges struktúra a sok közül. Az objektumorientált adatbázisokat a 4.9. alfejezetben tárgyaljuk.

Több olyan, eddigiektől eltérő modellt is használtak az ABKR korai időszakában, amelyek mára kimentek a divatból. A *hierarchikus modell* egy faorientált modell volt, akárcsak a félig-strukturált adat. A hátránya, hogy a modern modellektől eltérően, ez a fizikai szintet befolyásolta, és így lehetetlenné tette a programozók számára egy megfelelően magas szinten megfogalmazott kód megírását. A *hálós modell* egy másik ilyen típusú, amely egy gráforientált, fizikai szintű modell volt. Valójában a hierarchikus, valamint a mostanában használatos félig-strukturált modellek teljes gráfstruktúrákat is engednek, és nem korlátoznak csupán a faszerkezetekre. A hálós modellben azonban a gráfok voltak alapszerkezetként közvetlenül beépítve ahelyett, hogy a fákat részesítették volna előnyben, mint ahogy a többi említett modell teszi.

2.1.6. A modellezési megközelítések összehasonlítása

Még a jelenlegi vázlatos példáinknál is látható, hogy a félig-strukturált modellek rugalmasabbak, mint a relációk. Ez a különbség még nyilvánvalóbbá válik majd, mikor azt tárgyaljuk, hogy teljes gráfszerkezetek hogyan illeszthetőek be a faszerű, félig-strukturált modellekbe. A relációs modellt azonban még mindig előnyben részesítik, és meg szeretnénk magyarázni, hogy miért. Most ennek a rövid áttekintése következik.

Az adatbázisok nagy mérete miatt az adathoz történő hozzáférés és a velük végzett módosításoknak a hatékonysága nagyon fontos szerepet kap. Fontos

még a kényelmes használat is, vagy másképpen szólva az adatot feldolgozó programozók termelékenysége. Meglepő módon egy modell, nevezetesen a relációs modell teljesíti mindkét célkitűzést, azaz:

1. Egy egyszerű, korlátozott megközelítést ad az adatok strukturálására, ami ésszerűen sokoldalú. Azaz bármi modellezhető vele.

2. Egy korlátozott, ám hasznos, az adatokon végrehajtható műveletgyűjteményt tesz lehetővé.

Ez a két korlátozás együtt ad egy környezetet, és lehetővé teszi nyelvek megvalósítását, mint például az SQL-t, ami biztosítja a programozóknak, hogy kérdéseiket nagyon magas szinten megfogalmazhassák. Egy néhány soros SQL-utasítás elérheti ugyanazt az eredményt, mint egy több ezer soros C-program, vagy mint egy több száz soros kód, amelyet a korábbi modellekben (mint a hierarchikus vagy a hálós) írtak az adatokhoz történő hozzáférésre. Mivel a használható műveletek halmaza erősen korlátozott, a rövid SQL-programok még optimalizálhatóak is azért, hogy gyorsan vagy más választható nyelvben írt kódoknál gyorsabban futhassanak.

2.2. A relációs modell alapjai

A relációs modellben az adatok egyszerűen reprezentálhatók: kétdimenziós táblákban, ún. *relációk*ban. A 2.1. ábrán egy példát mutattunk relációra, amelyet most a 2.3. ábrán megismétlünk. A reláció neve `Filmek`, és a már többször hivatkozott, állandó példánk `Filmek` egyedhalmazának elemeiről tárolunk benne információt. Minden sor egy filmnek felel meg, az oszlopok pedig egy-egy filmtulajdonságnak. Ebben az alfejezetben bevezetjük a legfontosabb jelöléseket és fogalmakat a relációkhoz, és ezeket a `Filmek` reláción fogjuk szemléltetni.

filmcím	*év*	*hossz*	*műfaj*
Elfújta a szél	1939	231	dráma
Csillagok háborúja	1977	124	sci-fi
Wayne világa	1992	95	vígjáték

2.3. ábra. A `Filmek` relációja

2.2.1. Attribútumok

A reláció oszlopait az *attribútumok* látják el névvel. A 2.3. ábrán az attribútumok a következők: `filmcím`, `év`, `hossz` és `műfaj`. Az attribútumok az oszlopok fejrészében láthatók. Általában az attribútumok megadják az abban az oszlopban szereplő adatok jelentését. Például a `hossz` attribútummal ellátott oszlop minden egyes filmhez megadja annak hosszát percekben.

Relációk és attribútumok szokásos jelölései

Általában követendő példa, hogy a relációk neveit nagybetűvel, az attribútumneveket pedig kisbetűvel kezdjük. A könyv hátralévő részében azonban a relációkról absztrakt értelemben tárgyalunk, ahol nem számítanak az attribútumnevek. Ezen esetekben mind a relációkra, mind az attribútumaikra nagybetűket használunk, mint például az $R(A, B, C)$, ami egy általános 3-attribútumos relációt jelöl.

2.2.2. Sémák

A reláció nevét és a relációattribútumok halmazát együtt nevezzük *relációséma*nak. A relációsémát a reláció nevével és az attribútumainak zárójelek közötti felsorolásával adjuk meg. Tehát a 2.3. ábra `Filmek` relációsémája a következő:

```
Filmek(filmcím, év, hossz, műfaj)
```

A relációséma attribútumai halmazt alkotnak és nem listát. Mégis, amikor általában a relációkról van szó, meg kell határoznunk az attribútumok valamilyen szabványos sorrendjét. Tehát amikor attribútumok listájával vezetjük be a relációsémát, ugyanezt a szabványos sorrendet vesszük akkor is, amikor megjelenítjük a relációt vagy annak a sorait.

A relációs modellben az adatbázis egy vagy több relációsémát tartalmaz. A relációsémákból álló halmazt az adatbázisban *relációs adatbázisséma*nak vagy csak röviden *adatbázisséma*nak nevezzük.

2.2.3. Sorok

A reláció azon sorait, amelyek különböznek az attribútumokból álló fejléc sorától, *sorok*nak (tuple) nevezzük. A reláció minden egyes attribútumához tartozik a sorban egy *komponens*. Például a 2.3. ábra három sora közül az első sor négy komponense: `Elfújta a szél`, `1939`, `231`, `dráma`, amelyek ebben a sorrendben a `filmcím`, `év`, `hossz` és `műfaj` attribútumokhoz tartoznak. Amikor külön írjuk le a sorokat, nem pedig valamely reláció részeként, akkor vesszőkkel választjuk el a komponenseket, és a sort zárójelek közé tesszük. Például

```
(Elfújta a szél, 1939, 231, dráma)
```

a 2.3. ábra relációjának első sora. Megjegyezzük, hogy amikor egy sor magában van, az attribútumokat nem láthatjuk, így meg kell adnunk valamilyen hivatkozást, hogy melyik relációhoz tartozik az adott sor, és mindig ugyanabban a sorrendben írjuk fel a komponenseket, ahogyan az attribútumokat felsoroltuk a relációsémában.

2.2.4. Értéktartományok

A relációs modellben követelmény, hogy minden sor minden komponense atomi, azaz elemi típusú legyen, például egész vagy karaktersorozat. Nem megengedett az értékekhez a rekordszerkezet, halmaz, lista, tömb vagy bármely más olyan típus, amelynek az értékei kisebb komponensekre felbonthatók.

Feltesszük továbbá, hogy a reláció minden attribútumához tartozik egy *értéktartomány*, azaz egy elemi típus. A reláció bármely sorában szereplő komponensek értékének a megfelelő oszlop értéktartományából kell származnia. Például a 2.3. ábrán szereplő `Filmek` reláció sorainak az első komponense karaktersorozat, a második és harmadik komponense egész szám, a negyedik komponens értéke pedig karaktersorozat.

Olyan megadás is lehetséges, amely tartalmazza az értéktartományt (vagy típust) a relációsémában szereplő összes attribútumra. Ezt megtehetjük az attribútumot követő kettőspont és típus hozzáírásával. Azaz a `Filmek` reláció sémáját reprezentálhatjuk például az alábbi módon:

```
Filmek(filmcím:string, év:integer, hossz:integer, műfaj:string)
```

2.2.5. Relációk egyenértékű ábrázolási módjai

Egy reláció sorokból álló halmaz, nem pedig lista. Emiatt lényegtelen, hogy milyen sorrendben jelenítjük meg azokat. Például, ha a 2.3. ábra három sorát a hat lehetséges sorrend bármelyikében is írjuk fel, a reláció „ugyanaz", mint ami a 2.3. ábrán található.

Továbbá a reláció attribútumainak a sorrendjét is felcserélhetjük, a relációt ez nem változtatja meg. Mégis, amikor a relációsémát átrendezzük, ügyelnünk kell az oszlopok fejrészében szereplő attribútumokra. Azaz, amikor megváltoztatjuk az attribútumok sorrendjét, akkor ezzel megváltoztatjuk az oszlopok sorrendjét is. Ha az oszlopokat felcseréljük, akkor vele együtt a sorok komponenseinek a sorrendje is megváltozik. Az eredmény, hogy mindegyik sorban a komponensek ugyanúgy permutálódnak, mint ahogyan az attribútumok vannak permutálva.

Például a 2.4. ábrán látható reláció a 2.3. ábrán szereplő relációból a sorok és oszlopok permutációjával kapható egyik lehetséges reláció. A két relációt „azonosnak" tekintjük. Pontosabban ez a két tábla ugyanannak a relációnak két különböző megjelenítése.

év	*műfaj*	*filmcím*	*hossz*
1977	sci-fi	Csillagok háborúja	124
1992	vígjáték	Wayne világa	95
1939	dráma	Elfújta a szél	231

2.4. ábra. A `Filmek` reláció másik megjelenítése

2.2.6. Relációk előfordulásai

A filmeket tartalmazó reláció nem állandó, sőt a relációk többször is változhatnak az idők során. A változások egy része várhatóan a reláció soraira fog vonatkozni, mint például új sorok beszúrása, azaz az új filmek adatbázisba vétele, a létező sorok megváltoztatása, ha újabb vagy pontosabb információt kapunk a filmekről, és esetleg sorok törlése, ha filmeket valamilyen ok miatt eltávolítunk az adatbázisból.

A relációséma megváltoztatása kevésbé általános. Bár előfordulhatnak olyan helyzetek, amikor attribútumokat szeretnénk felvenni vagy törölni. A forgalomban levő adatbázisrendszerek lehetővé teszik, hogy a sémát megváltoztassuk, ám ez igen költséges, ugyanis előfordulhat, hogy milliónyi sor mindegyikét át kell írni, hogy hozzávegyünk vagy töröljünk komponenseket. Ha új attribútummal bővítettünk, akkor az is nehézséget okoz, vagy egyáltalán nem lehetséges, hogy a sorok új komponenseihez megfelelő értékeket találjunk.

Az adott reláció sorainak halmazát *reláció-előfordulás*nak nevezzük. Például a 2.3. ábrán található három sor a `Filmek` reláció egy előfordulása. Feltehetően a `Filmek` reláció változott már a múltban és változni fog a jövőben. Például 1990-ben a `Filmek` nem tartalmazta a `Wayne világa` sort. A hagyományos adatbázisrendszerek bármely relációnak csak egyetlen változatát kezelik: csak azokat a sorokat, amelyek „most" vannak a relációban. Ezt a reláció-előfordulást *aktuális előfordulás*nak[1] nevezzük.

2.2.7. A reláció kulcsai

A relációs modell jó néhány – a relációra vontkozó – megszorítás megfogalmazását lehetővé teszi számunkra az adatbázissémán belül. A megszorítások tárgyalását a 7. fejezetre hagyjuk. Van viszont egy olyan alapvető megszorítás, amelyet már itt be kell vezetnünk: a *kulcs* megszorítás. Az attribútumok egy halmaza egy kulcsot alkot egy relációra nézve, ha a reláció előfordulásaiban nincs két olyan sor, amelyek a kulcs összes attribútumának értékein megegyeznének.

2.1. példa. Megadhatjuk a `Filmek` relációt egy kulccsal is, amelyet a `filmcím` és `év` attribútumok alkotnak. Azaz nem feltételezzük azt, hogy valaha is lehetne két olyan film, amelyeknek a címe is és a gyártási éve is megegyezne. Figyeljük meg, hogy a „felújított" filmek létezése miatt a `filmcím` önmagában nem alkothat kulcsot. Példaként három *King Kong* című film is van, de mindegyik más `évben` készült. Az is látszik, hogy az év sem lehet kulcs, hiszen több film is készül egy adott évben. □

[1] Azokat az adatbázisokat, amelyek az adat időben korábbi verzióit is nyilvántartják, *temporális adatbázisok*nak nevezzük.

Az attribútum vagy attribútumok aláhúzásával jelöljük, hogy a szóban forgó attribútum kulcsattribútum. Ezért például a `Filmek` reláció sémáját a következőképpen is írhatnánk:

`Filmek(filmcím, év, hossz, műfaj)`

Emlékezzünk, hogy az az állítás, hogy az attribútumok egy halmaza alkot kulcsot a relációra nézve, egyúttal egy, a reláció összes előfordulására vonatkozó állítás is. Például tekintsük a 2.3. ábra apró relációját. Ekkor állíthatnánk azt is, hogy a `műfaj` önmagában kulcsot fog alkotni, hiszen nem látunk két olyan sort, amelyek `műfaj` komponensükön megegyeznének. Viszont az is könnyen elképzelhető, hogy ha a reláció előfordulásai több filmet tartalmaznának, akkor sok dráma, sok vígjáték és sok más egyéb kategóriájú film is lenne. Azaz több olyan sor is lehetne, amelyek a `műfaj` komponensükön megegyeznek. Következésképpen a `Filmek` relációnak a `műfaj` nem lehet kulcsa.

Miközben mi megbizonyosodtunk arról, hogy a `filmcím` és az `év` együtt lehetnek a `Filmek` kulcsa, addig a legtöbb valós adatbázisban mesterségesen előállított kulcsokat használnak, mivel kételkednek annak a biztonságos voltában, hogy nem teljes kontroll alatt lévő attribútumértékekről feltevéseket tegyenek. Például a cégek általában minden egyes alkalmazottukhoz egy olyan egyedi alkalmazottazonosítót rendelnek, amelyeket nagy körültekintéssel egyedi számnak választanak meg. Ezeknek az azonosítóknak célja, hogy a cég adatbázisán belül minden egyes dolgozót meg tudjanak különböztetni a többitől, még akkor is, ha van két azonos nevű alkalmazott. Azaz az alkalmazottazonosító attribútum lehet kulcsa az alkalmazottak relációjának.

Az amerikai cégeknél minden egyes alkalmazottnak van társadalombiztosítási száma. Ha az adatbázisukban van egy társadalombiztosítási számra vonatkozó attribútum, akkor ez is lehet az alkalmazottak relációjának a kulcsa. Megjegyezzük, hogy az nem jelent problémát, ha több kulcsválasztás is létezik, mint ahogyan az alkalmazottakra nézve is mind az alkalmazottazonosító, mind a társadalombiztosítási szám kulcs volt.

A kulcsként használható attribútum létrehozásának az elve széles körben elterjedt. Az alkalmazottazonosítók mellett az egyetemek is hallgatói azonosítóval különböztetik meg a diákjaikat. Vannak még jogosítványszámok és rendszámok a sofőrök, illetve az autók közötti különbségtételhez. Az olvasó minden kétséget kizáróan magától is több olyan attribútumokra vonatkozó példát tud felsorolni, amelyeket azért hoztak létre, hogy kulcsként szolgáljanak.

2.2.8. Példa egy adatbázissémára

Az alfejezetet egy teljes adatbázissémát tartalmazó példa megadásával zárjuk. A szóban forgó példa témája a filmek lesznek. Az egész a korábban már szerepeltetett `Filmek` relációra fog épülni. Az adatbázis sémáját a 2.5. ábrán láthatjuk. A séma alapjainak megértéséhez tekintsük át részleteiben az egyes részeit:

```
Filmek(
    filmcím:string,
    év:integer,
    hossz:integer,
    műfaj:string,
    stúdióNév:string,
    producerAzon:integer
)
FilmSzínész(
    név:string,
    cím:string,
    nem:char,
    születésiDátum:date
)
SzerepelBenne(
    filmCím:string,
    filmÉv:integer,
    színészNév:string
)
GyártásIrányító(
    név:string,
    cím:string,
    azonosító:integer,
    nettóBevétel:integer
)
Stúdió(
    név:string,
    cím:string,
    elnökAzon:integer
)
```

2.5. ábra. Példa egy adatbázissémára a filmek tárolásához

`Filmek`
Ez a reláció a korábban tárgyalt példareláció kiterjesztése. Emlékezzünk rá, hogy a `filmcím` és az `év` együttesen kulcsot alkotnak a relációra nézve. Két új attribútum keletkezett. A `stúdióNév` szolgál a tulajdonos gyártóazonosítójaként. A `producerAzon` pedig egy olyan egész szám lesz, amely a film producerét úgy fogja azonosítani, mint ahogyan a `GyártásIrányító` bemutatásánál majd megbeszéljük.

`FilmSzínész`
Ez a reláció határozza meg a színészekre vonatkozó adatokat. A kulcs a `név` attribútum lesz, amely a színész nevét tartalmazza. Általában viszont a név

nem feltétlenül egyedi, ezért nem is biztos, hogy kulcs lesz. A filmszínészek viszont eltérnek ettől abban az értelemben, hogy nem használnak olyan nevet, amelyet már valamelyik másik színész használt volna. Ezért is használjuk azt a feltevést, hogy a filmszínészek nevei egyediek. Egy általánosabban használható megközelítés egy olyan azonosító sorszám használata, mint a társadalombiztosítási számé, amely minden személyhez egy egyedi sorozatszámot rendel. Így ez használható kulcsattribútumként. Ezt a megközelítést alkalmazzuk a filmgyártókra, mint ahogy azt látni fogjuk. A másik érdekesség a `FilmSzínész` relációval kapcsolatban, hogy két új adattípust is tartalmaz: a nemet, amely lehet egy F vagy egy N karakter, illetve a születési dátumot, amely „dátum" típusú, speciális formátumú karaktersorozatot takar.

`SzerepelBenne`
Ez a reláció szolgál a film és a színészek kapcsolatának leírására azáltal, hogy összekapcsolja a színészt azokkal a filmekkel, melyekben a színész szerepel. Figyeljük meg, hogy a filmeket a `Filmek` tábla kulcsával (`filmcím` és `év`) reprezentáljuk annak ellenére, hogy más attribútumneveket választottunk hozzájuk, nevezetesen filmCím, filmÉv. A színészeket ehhez hasonlóan a `FilmSzínész színészNév` kulcsattribútumával ábrázoljuk. Végül figyeljük meg, hogy itt a három attribútum együttesen alkot kulcsot. Ez pedig pontosan megfelel annak az elképzelésnek, hogy a `SzerepelBenne` relációnak lehet két olyan különböző sora, amelyek valamely két attribútumukon megegyeznek. Például, ha egy adott színész egy adott évben két filmben is szerepelt, akkor lesz két olyan sor, amelyek a `filmÉv`, illetve `színészNév` komponensükben megegyeznek, de a `filmCím` komponenseiken eltérnek.

`GyártásIrányító`
Ez a reláció a filmgyártók adatait tartalmazza: a gyártó nevét, címét és a gyártó nettó bevételét. A filmgyártókat, beleértve a producereket (ahogy a `Filmek` relációban fel van tüntetve) és a stúdió vezetőit is, akik csak a következő `Stúdió` relációban szerepelnek, egy „azonosító számmal" láttuk el, amely kulcsként fog szolgálni. Ezek az azonosítók egész számok, és minden gyártóra különbözőek.

`Stúdió`
Ez a reláció a filmstúdiókról szól. Feltételezzük, hogy két stúdiónak nem lehet azonos a neve, így a `név` kulcs lesz a relációra nézve. A másik két mező: a stúdió címe, illetve a stúdió vezetőjéhez rendelt azonosító szám. Feltételezzük, hogy a stúdió vezetője egy valódi gyártásirányítót takar, és így szerepel a `GyártásIrányító` relációban is.

2.2.9. Feladatok

2.2.1. feladat. Legyen a 2.6. ábrán található két reláció-előfordulás egy banki adatbázis része. Mutassuk meg a következőket:

a) Mindegyik reláció attribútumait.

b) Mindegyik reláció sorait.

c) Mindegyik reláció egy sorának a komponenseit.

d) Mindegyik reláció relációsémáját.

e) Az adatbázissémát.

f) Mindegyik attribútumhoz egy megfelelő értéktartományt.

g) Mindegyik relációhoz egy másik, vele ekvivalens megjelenítését.

számlaSzám	*típus*	*egyenleg*
12345	Betétszámla	12000
23456	Folyószámla	1000
34567	Betétszámla	25

A Számlák reláció

vezetékNév	*keresztNév*	*azonosítóSzám*	*számla*
Balogh	Róbert	901-222	12345
Kovács	Léna	805-333	12345
Kovács	Léna	805-333	23456

A Vevők reláció

2.6. ábra. Egy banki adatbázis két relációja

2.2.2. feladat. Több példát is javasoltunk a 2.2.7. alfejezetben az olyan attribútumok kiválasztására, amelyek a relációk kulcsaként szolgálhatnak. Adjunk néhány további példát!

!! **2.2.3. feladat.** Hány különböző módon reprezentálható egy reláció-előfordulás (az attribútumok és sorok sorrendjét tekintve), ha az előfordulás az alábbiakkal rendelkezik:

a) Három attribútum és három sor, mint a 2.6. ábra `Számlák` relációjában?

b) Négy attribútum és öt sor?

c) n attribútum és m sor?

2.3. Relációsémák definiálása SQL-ben

Az SQL a legfontosabb nyelv, amely lehetővé teszi relációs adatbázisok leírását illetve módosítását is. A legutóbbi SQL-szabvány az SQL-99. A forgalomban lévő adatbázis-kezelő rendszerek túlnyomó része a szabványhoz hasonló – de nem teljesen azonos – dolgokat valósít meg. Az SQL-nek két szempontnak megfelelő része van:

1. az *adatdefiníciós* résznyelv az adatbázissémák megadásához, illetve
2. az *adatmanipulációs* résznyelv az adatbázis *lekérdezés*éhez (vagy az adatbázisnak adható kérdések megfogalmazásához), illetve az adatbázis módosításához.

A legtöbb programozási nyelvben megvan egy ilyen típusú megkülönböztetés. Például a *C*-nek és a *Java*nak is van deklarációs része, illetve futtatható kód része. Ez pedig pontosan megfelel az adatdefiníciós, illetve adatmanipulációs kettősnek.

Ebben a részben az adatdefiníciós rész tárgyalását kezdjük el, de részletesebben a 7. fejezetben térünk vissza rá, főként az adatokon tehető megszorítások témakörénél. Az adatmanipulációs résszel a 6. fejezet részletesen foglalkozik.

2.3.1. SQL-relációk

Az SQL három típusú relációt ismer:

1. Tárolt relációk vagy más néven *táblák*. Általában ilyen relációkkal foglalkozunk. Ezek benne vannak az adatbázisban, soraik változtatásával megváltoztathatók és soraik is lekérdezhetők.
2. A *Nézetek* számításokból kapott relációk. Ezeket nem tároljuk, de részben vagy teljes egészben szerkeszthetjük őket, ha ez szükséges. Ezek lesznek a 8.1. alfejezet fő témái.
3. Ideiglenes táblák, amelyeket az SQL nyelvi feldolgozója készít, amikor valamilyen lekérdezéseket és adatmódosításokat végez el. Ezeket a relációkat eldobja és nem tárolja sehol az SQL feldolgozója.

Ebben a részben a táblák megadásának módjait sajátítjuk el. Itt nem foglalkozunk a nézetek meghatározásával és definiálásával sem. Az ideiglenes táblák pedig sohasem deklaráltak. Az SQL `CREATE TABLE` utasításával határozható meg egy tárolt reláció sémája. Megadja a tábla nevét, az attribútumait, illetve az attribútumok típusát. Emellett kulcs, illetve kulcsok megadását is lehetővé teszi. A `CREATE TABLE` utasításnak az eddig említetteken túl több lehetősége is van. Ilyenek például a különféle megszorítások, amelyeket szintén meghatározhatunk vele. Illetve ilyenek az *indexek* (vagy olyan adatstuktúrák, amelyek a táblán végezhető műveletek felgyorsítására szolgálnak) megadása is, de most halasszuk ezeket egy alkalmasabb időpontra.

2.3.2. Adattípusok

Bevezetésképpen bemutatjuk az SQL-rendszerek által használt fő adattípusokat. Minden attribútumhoz kötelező megadni egy adattípust.

1. Rögzített vagy változó hosszúságú karaktersorok. A `CHAR(n)` típus egy rögzített n hosszúságú karaktersort jelöl. A `VARCHAR(n)` egy legfeljebb n hosszúságú karaktersort jelöl. A különbség közöttük implementációfüggő, ugyanis a `CHAR` általában azt jelenti, hogy a rövidebb karaktersorozatokat kiegészítjük n hosszúvá, míg a `VARCHAR` esetén használnak egy lezáró jelet vagy hosszértéket is. Az SQL ésszerű típuskényszerítést is megenged a két típus értékei között. Normál körülmények között a stringet a végére fűzött üres karakterekkel egészítjük ki, amennyiben egy annál hosszabb, rögzített méretű karaktersorozatnak adjuk értékül. Tekintsük a `'foo'` karaktersorozatot, ha ezt egy `CHAR(5)` típusú attribútumnak adjuk értékül, akkor ez a `'foo '` értéket jelenti (azaz két szóközzel kiegészítve az eredetit).[2]

2. Rögzített vagy változó hosszúságú bitsorok. Ezek hasonlóak a rögzített és a változó hosszúságú karaktersorokhoz, csak nem karakterekből, hanem bitekből állnak. A `BIT(n)` típus egy n hosszúságú bitsort, míg a `BIT VARYING(n)` egy legfeljebb n bitből álló bitsort jelképez.

3. A `BOOLEAN` egy logikai értékű attribútumot jelöl. Az ilyen attribútumok lehetséges értékei: `TRUE`, `FALSE` vagy – ami még George Boole-t is meglepné – `UNKNOWN`.

4. Az `INT` vagy más néven `INTEGER` típusok egész számokat jelképeznek. A `SHORTINT` típus is egész számot jelképez, de kevesebb biten tároljuk, ez a bitszám megvalósítástól függő (ugyanúgy mint az `int` és `short int` a C-ben).

5. A lebegőpontos értékeket többféleképpen is tárolhatjuk. Használhatjuk a `FLOAT` és a `REAL` típusokat (amelyek megegyeznek). Nagyobb pontossággal

[2] Megjegyezzük, hogy sok más egyéb programozási nyelvhez hasonlóan SQL-ben is a karakterláncokat idézőjelek, illetve macskakörmök közé írjuk.

Dátumok és időpontok SQL-ben

A különböző SQL-megvalósításokban eltérő módokon ábrázolhatják a dátumot, illetve az időt, de a most következőkben mi az SQL szerinti reprezentálását vizsgáljuk majd. Egy dátumot a `DATE` kulcsszóval, utána pedig egy idézőjelek közötti speciális karaktersorral írhatunk le. Például a `DATE '1948-05-14'` megfelel a leírásnak. Az első négy karakter az év számjegyeit jelképezi. Utána következik egy kötőjel, majd két számjegy jelképezi a hónapot. Végül következik egy újabb kötőjel és két számjegy, mely a napot jelképezi. Megjegyezzük, hogy mind a hónapok, mind a napok esetében az egy számjegyből álló számokat kiegészítjük egy 0-val.

Egy *idő* értéket hasonlóan, a `TIME` kulcsszó és egy idézőjelek közötti karakterlánc segítségével lehet ábrázolni. A karaktersorban az órának két számjegy felel meg a 24 órás órán. Ezután kettőspont következik, két számjegy a perceknek, újabb kettőspont, majd két számjegy a másodperceknek. Ha a másodperc törtrészeit is szeretnénk használni, következhet egy pont és annyi számjegy, amennyire szükség van. Így a `TIME '15:00:02.5'` azt az időt jelképezi, amikor már az összes diák elhagyta az osztálytermet egy olyan óra után, mely délután 3-kor ért véget, tehát két és fél másodperccel három után.

tárolhatók az értékek a `DOUBLE PRECISION` típusban; a különbségek ismét a C-hez hasonlóak. Az SQL-ben vannak olyan típusok is, amelyek fixpontos valós számok. Például a `DECIMAL(n,d)` olyan értékeket tárol, amelyek n számjegyből állnak, és a tizedesponttól jobbra d számú tizedesjegy áll. Így a 0123,45 érték típusa `DECIMAL(6,2)`. A `NUMERIC` is a `DECIMAL` egyik szinonimája, habár lehetnek különbségek a megvalósítástól függően.

6. Dátumokat és időket a `DATE` és `TIME` típusok jelképeznek. Lásd bővebben a „Dátumok és időpontok SQL-ben" keretes résznél. Ezek az értékek tulajdonképpen speciális alakú karaktersorok. A dátum és idő értékeket átkonvertálhatjuk karaktersorrá és visszafelé is, ha a karaktersor értelmes idő, illetve dátum értéket képvisel.

2.3.3. Egyszerű táblalétrehozások

A legegyszerűbb módja egy tábla létrehozásának a `CREATE TABLE` kulcsszavakból, a relációnévből, az attribútumoknak és típusaiknak zárójelek közé tett listájából áll.

2.2. példa. A 2.5. ábrán látható `Filmek` relációt a 2.7. ábrán látható módon adhatjuk meg. A `filmcím` 100 hosszú karakterláncként deklarált, az `év` és `hossz` attribútumok egész számok, a `műfaj` pedig 10 hosszú karakterlánc.

```
CREATE TABLE Filmek (
    filmcím       CHAR(100),
    év            INT,
    hossz         INT,
    műfaj         CHAR(10),
    stúdióNév     CHAR(30),
    producerAzon  INT
);
```

2.7. ábra. A `Filmek` tábla meghatározása SQL-ben

A cím hosszának választása tetszőleges, most éppen 100 karaktert engedtünk meg azért, mert nem akarjuk túlságosan korlátozni vagy éppen megcsonkítani a címek értékét. Feltettük azt is, hogy 10 karakter megfelel a műfaj tárolására. Ez megint egy véletlenszerű választás, amit meg is bánhatunk, ha lesz egy hosszú nevű műfajunk. A stúdiónevek hasonló okokból 30 karakterből állnak majd. A producerek azonosító száma pedig egy egész szám lesz. □

2.3. példa. A 2.5. ábra `FilmSzínész` reláció sémáját a 2.8. ábrán látható SQL-utasítással hozhatjuk létre. Néhány új lehetőségét is látjuk az adattípusoknak. A tábla neve `FilmSzínész`, és négy attribútuma lesz. Az első két attribútum, a `név` és a `cím` típusa karaktersor. A különbség abban áll, hogy a `név` attribútum egy 30 karakterből álló rögzített hosszúságú karaktersor, amelyet szükség esetén üres karakterekkel töltünk ki, illetve lecsonkítjuk a végét, ha hosszabb. Ezzel szemben a címet egy változó hosszúságú karaktersorba helyeztük, melynek maximális hossza 255 karakter.[3] Nem biztos, hogy ez a két választás a legjobb, csak azért használtuk ezeket, hogy bemutassuk a kétféle karaktersor használatát.

```
CREATE TABLE FilmSzínész (
    név             CHAR(30),
    cím             VARCHAR(255),
    nem             CHAR(1),
    születésiDátum  DATE
);
```

2.8. ábra. A `FilmSzínész` reláció sémájának definiálása

A `nem` attribútum értékei lehetnek `'N'` és `'F'`, tehát egyetlen karakter. Így nyugodtan használhatjuk az egy hosszúságú karaktersort típusként. Végül a `születésiDátum` attribútum értéke természetesen `DATE`. □

[3] A 255 nem valamilyen bűvös szám, hogy minden címnek ilyen hosszúnak kell lennie. Egy bájton 0 és 255 közötti számokat lehet tárolni. Tehát egy változó hosszúságú karaktersort, amelynek maximális hossza 255, úgy lehet tárolni, hogy egy bájton tároljuk a karakterek számát, és magát a karaktersort annyi bájton, amennyi szükséges. A kereskedelmi rendszerek általában hosszabb változó hosszúságú karaktersorokat is megengednek.

2.3.4. Relációsémák módosítása

Most már tudjuk, hogyan lehet megadni egy relációt. De mi van akkor, ha meg akarjuk változtatni egy olyan tábla sémáját, amelyet már régóta használtunk, és amelynek túl sok sora is van emiatt? Eltávolíthatjuk az egész táblát az összes aktuális sorával együtt, vagy módosíthatjuk is a sémát új attribútum hozzáadásával, illetve törlésével.

A következő SQL-utasítás törli a táblát:

```
DROP TABLE R;
```

Végrehajtása után az R reláció nem lesz benne a relációsémában, és így a soraihoz sem férhetünk hozzá a továbbiakban. Egy hosszú életű adatbázis esetében sokkal többször kell egy tábla struktúráját módosítani, mint egy táblát megszüntetni. A megfelelő utasítások az `ALTER TABLE` kulcsszavakkal és a reláció nevével kezdődnek. Ezután több lehetőségünk is van, a legjelentősebb azonban a következő kettő:

1. `ADD`, majd egy attribútumnév és annak adattípusa.
2. `DROP` és egy attribútumnév.

2.4. példa. Módosítsuk a `FilmSzínész` relációt, hozzáadva egy `telefonszám` attribútumot:

```
ALTER TABLE FilmSzínész ADD telefonszám CHAR(16);
```

Az utasítás eredményeképpen a `FilmSzínész` relációnak öt attribútuma lesz, az első négyet a 2.8. ábra bemutatta, az ötödik pedig a `telefonszám` attribútum, amely egy rögzített 16 bájt hosszúságú karaktersor. A megváltozatott relációban minden sorban szerepelni fog a `telefonszám` komponens, de mivel még nincsenek telefonszámok megadva, ezért ezen komponensek értéke egy speciális, `NULL` nevű *nullérték* lesz. A 2.3.5. alfejezetben ismertetni fogjuk, hogyan lehet egy másik alapértelmezett értéket beállítani a `NULL` érték helyett.

Egy másik példaként töröljük ki a `születésiDátum` attribútumot:

```
ALTER TABLE FilmSzínész DROP születésiDátum;
```

Ennek eredményeként ez az attributum nem lesz benne a sémában a továbbiakban, és a `FilmSzínész` összes sorának a `születésiDátum` komponense is törlődni fog. □

2.3.5. Alapértelmezés szerinti értékek

Amikor létrehozunk vagy módosítunk sorokat, néha egyes komponensek értékét nem ismerjük. A 2.4. példában a relációsémához hozzáadtunk egy új oszlopot, és a létező sorokban nincs meg az új oszlop értéke. Ott azt javasoltuk, hogy a komponens értéke `NULL` érték legyen. Vannak esetek, amikor *kezdeti* érték ilyen

választása nem megfelelő, hanem helyette valamilyen rögzített értéket használunk az oszlopra, ha nem ismert az értéke. Általában, amikor egy attribútumot és az adattípusát definiáljuk, hozzáadhatjuk a `DEFAULT` kulcsszót és egy megfelelő értéket. Az érték vagy `NULL` érték, vagy egy konstans. Egyéb olyan értékeket is használhatunk, amelyeket a rendszer rendelkezésünkre bocsát, mint például az aktuális idő.

2.5. példa. Tekintsük a 2.3. példát. Szeretnénk alapértelmezésként a `'?'` karaktert használni a `nem` attribútum esetében és a `'0000-00-00'` dátumot a `születésiDátum` esetében. Így a 2.8. ábra megfelelő sorait a következőkre kell kicserélni:

```
nem CHAR(1) DEFAULT '?',
születésiDátum DATE DEFAULT DATE '0000-00-00'
```

Egy másik példaként az 2.4. példában a `telefonszám` új attribútum alapértelmezett értéke legyen a `'nem ismert'` szöveg. Ekkor a következő `ALTER TABLE` utasítást kell használnunk:

```
ALTER TABLE Filmszínész
     ADD telefonszám CHAR(16) DEFAULT 'nem ismert';
```

□

2.3.6. Kulcsok megadása

Két módszer van, amivel egy attribútumot vagy attribútumoknak egy halmazát kulcsként adhatjuk meg egy `CREATE TABLE` utasítás során (tárolt reláció megadásánál):

1. Egy attribútumot kulcsként adhatjuk meg a relációséma attribútumlistájának megadásánál.

2. A sémában megadott lista elemeit (vagyis az attribútumokat) használhatjuk egy olyan deklarációban, amely azt fejezi ki, hogy a szóban forgó attribútum vagy attribútumhalmaz kulcsot alkot a relációra nézve.

Ha a kulcs egynél több attribútumból áll, akkor a 2. módszert használjuk. Ha pedig egy attribútum alkotja, akkor bármelyik alkalmazható.

A kulcsjelleget kétféleképpen fejezhetjük ki:

a) `PRIMARY KEY`, vagy

b) `UNIQUE`.

Ha az S attribútumhalmazt az R reláció kulcsaként értelmezzük, akkor (akár `PRIMARY KEY`, akár `UNIQUE` kulcsszót használtunk) a hatás a következő lesz:

- Az R-nek nem lehet két olyan sora, ami az S attribútumok értékein megegyezne, még akkor sem, ha az értékek NULL értékek. Bármely olyan beszúrási vagy módosítási próbálkozás, amely sérti ezt a szabályt, azt okozza, hogy az ABKR visszautasítja a szabálysértő művelet végrehajtását.

Ha a PRIMARY KEY kulcsszót használjuk, akkor S attribútumaira nem szabad, hogy a NULL engedélyezve legyen. A rendszer itt is visszautasít minden próbálkozást, amely megsérti a szabályt. Ezzel szemben a UNIQUE esetén S-re megengedettek a NULL értékek. Az ABKR meg tudja különböztetni ezt a két feltételt, ha szükséges.

2.6. példa. Tekintsük újra a FilmSzínész reláció sémát. Mivel egyik filmszínész sem használja a másik nevét, ezért a név önmagában alkothat kulcsot a relációra nézve. Azaz ezt a tényt hozzáadhatjuk a név meghatározásához. A 2.9. ábra szemlélteti a 2.8. ábra változásait. A UNIQUE helyettesíthetné a PRIMARY KEY-t a megadásnál. Ez esetben viszont kettő vagy több sornak is lehetne a név értéke NULL, de ezenkívül más ismétlődő érték nem lesz erre az attribútumra nézve.

```
CREATE TABLE FilmSzínész (
   név CHAR(30) PRIMARY KEY,
   cím VARCHAR(255),
   nem CHAR(1),
   születésiDátum DATE
);
```

2.9. ábra. A név legyen kulcs

A kulcsokat külön is megadhatjuk. Ebben az esetben az eredményséma a 2.10. ábrán látható. Itt is kicserélhető a PRIMARY KEY UNIQUE-ra. □

```
CREATE TABLE FilmSzínész (
   név CHAR(30),
   cím VARCHAR(255),
   nem CHAR(1),
   születésiDátum DATE,
   PRIMARY KEY (név)
);
```

2.10. ábra. A kulcs külön történő megadása

2.7. példa. A 2.6. példában mind a 2.9. ábra, mind a 2.10. ábra formalizmusa megfelelő volt, hiszen a kulcs egy attribútumból állt. Ha több attribútum alkotja a kulcsot, akkor a 2.10. ábrán mutatott megadást kell használnunk. Például a

Filmek relációnál, ahol a filmcím és az év attribútumpár alkotott kulcsot, ez a 2.11. ábrán látható módon adható meg. Ahogy azt már megszokhattuk, a UNIQUE felcserélhető a PRIMARY KEY-vel. □

```
CREATE TABLE Filmek (
    filmcím      CHAR(100),
    év           INT,
    hossz        INT,
    műfaj        CHAR(10),
    stúdióNév    CHAR(30),
    producerAzon INT,
    PRIMARY KEY (filmcím, év)
);
```

2.11. ábra. A filmcím és év a Filmek kulcsa

2.3.7. Feladatok

2.3.1. feladat. Ebben a feladatban bemutatjuk az egyik, a relációs adatbázissémáknál gyakran használt példánkat. Az adatbázisséma négy relációt tartalmaz, amelyeknek a sémája a következő:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

A Termék reláció megadja a gyártót, a modellszámot és a termékek típusát (PC, laptop vagy nyomtató). Az egyszerűség kedvéért a modellszámot egyedinek tekintjük az összes gyártóra és terméktípusra nézve. Ez a feltételezés általában nem helytálló, hiszen egy valós adatbázisban kellene egy kód a gyártókra is, amely a modellszámnak a része. A PC reláció megadja minden modellszámhoz a PC sebességét (a processzorét gigahertzben), a RAM-értéket (megabájtban), a merevlemez méretét (gigabájtban) és az árat. A Laptop reláció hasonló, a különbség csak az, hogy a képernyőméret (inch-ben) is meg van adva. A Nyomtató reláció minden nyomtatóhoz tárolja, hogy a nyomtató színes nyomtató-e (igaz, ha igen), illetve a nyomtató típusát (általában lézer- vagy tintasugaras) és az árát.

Adjuk meg a következő deklarációkat:

a) A Termék reláció megfelelő sémáját.

b) A PC reláció sémáját.

c) A Laptop reláció sémáját.

d) A `Nyomtató` reláció sémáját.

e) A *d)* pontbeli `Nyomtató` séma módosítását úgy, hogy a `színes` attribútumot töröljük.

f) A *c)* pontbeli séms módosítását úgy, hogy tartalmazzon egy `cd` mezőt, melynek alapértelmezett értéke legyen `'nincs'`, abban az esetben, ha a laptop nem tartalmaz CD-olvasót.

2.3.2. feladat. Ebben a feladatban bemutatjuk egy másik gyakran használt példánkat a II. világháború hadihajóiról. A következő relációk lesznek benne:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

A hajók „osztályonként" azonos módon épülnek, és az osztály általában az osztály első hajójáról van elnevezve. A `Hajóosztályok` reláció tárolja az osztály nevét, típusát (`'hh'` a hadihajó, `'hc'` a hadicirkáló), az építtető ország nevét, a fegyvereinek számát, a kaliberüket (az ágyúcső átmérőjét inch-ben) és a vízkiszorításukat (súly tonnában). A `Hajók` reláció tartalmazza a hajó nevét, az osztályának nevét, illetve a vízre bocsátásának dátumát. A `Csaták` reláció a hajók közötti csaták nevét és dátumát rögzíti. A `Kimenetelek` reláció pedig megadja az összes hajóra az összes csatájának az eredményét (elsüllyedt, megsérült vagy ép).

Adjuk meg a következő deklarációkat:

a) Egy megfelelő sémát a `Hajóosztályok` reláció számára.

b) A `Hajók` reláció sémáját.

c) A `Csaták` reláció sémáját.

d) A `Kimenetelek` reláció sémáját.

e) Az *a)* pontban megadott `Hajók` reláció módosítását úgy, hogy töröljük ki a `kaliber` attribútumot.

f) A *b)* pontban megadott `Hajók` reláció sémájának módosítását úgy, hogy tartalmazza a `készült` attribútumot, mely megadja azt a helyet, ahol a hajót készítették.

2.4. Egy algebrai lekérdező nyelv

Ebben a részben bevezetjük a relációs modell adatmmanipulációs részeit. Emlékezzünk, hogy az adatmodell nem csupán egy struktúra, hanem kellenek módszerek az adat lekérdezésére és módosítására is. A relációkon értelmezett műveletek tárgyalása előtt be kell vezetnünk egy speciális algebrát, a *relációs algebrát*, amely egyszerű (de hatékony) lehetőségeket tartalmaz az adott relációkból új relációk létrehozására. Ha a relációk tárolt adatok, akkor az előállított reláció lehet egy, az adatokon történt lekérdezésre a válaszreláció. A relációs algebrát a forgalomban lévő ABKR-ek manapság már nem használják lekérdező nyelvként annak ellenére, hogy a kezdeti próbaverziók viszont direkt módon használták ilyen célokra. Az SQL, a „valódi" lekérdező nyelv, a relációs algebrára épül, és sok SQL-program valójában „szintaktikailag fűszerezett" relációs algebra. Sőt amikor az ABKR egy lekérdezést dolgoz fel, akkor az első dolog, ami az SQL-lekérdezéssel történik, hogy lefordítják vagy relációs algebrába, vagy egy hozzá nagyon hasonló belső formára. Azaz rengeteg oka van, hogy a relációs algebra tanulmányozásával kezdjük a tanulást.

2.4.1. Miért kell egy speciális lekérdező nyelv?

A relációs algebra műveleteinek bevezetése előtt megkérdezhetnénk, hogy miért is van szükségünk egy új típusú programozási nyelvre az adatbázisokhoz. Nem lenne elegendő a legelterjedtebb (mint C, illetve Java) nyelvek használata, hogy kérdéseket fogalmazhassunk meg, illetve hogy kiszámoljuk a kérdéshez tartozó relációt? Mindemellett a reláció egy sora kifejezhető struktúraként (C-ben) vagy objektumként (Javában), és így a relációt megadhatnánk ilyen elemek tömbjeként is. A meglepő válasz, hogy a relációs algebra azért hasznos, mivel *kevésbé* kifejezőbb, mint a C vagy a Java. Vagyis vannak számítások, amelyek megvalósíthatóak ezen nyelvekben, de a relációs algebrában nem. Egy példa erre, ha meg akarjuk határozni, hogy egy relációnak páros vagy páratlan számú sora van-e. Azzal, hogy korlátozzuk, hogy mit tudunk kifejezni, illetve tenni a lekérdező nyelvünkben, két nagy előnyre teszünk szert: kényelmes programozást és a fordító által magas szinten optimalizált kódok előállítását. Ezeket a 2.1.6. alfejezetben tárgyaltuk.

2.4.2. Mit nevezünk algebrának?

Egy algebra általában műveleteket és atomi operandusokat tartalmaz. Például egy számtani algebra esetén az x-hez hasonló változók és a 15-höz hasonló konstansok lesznek az atomi operandusok. A műveletek általában számtani műveletek lesznek: összeadás, kivonás, szorzás és osztás. Az algebra lehetővé teszi *kifejezések* megfogalmazását az atomi operandusokon és/vagy algebrai kifejezéseken végzett műveletek alkalmazásával. Gyakran zárójeleket is kell használnunk a műveletek operandusainak különválasztásához, mint például a következő kifejezéseknél: $(x + y) * z$ vagy $((x + 7)/(y - 3)) + x$.

A relációs algebra is példa egy algebrára. Az atomi operandusok a következők:

1. A relációkhoz tartozó változók.
2. Konstansok, amelyek véges relációt fejeznek ki.

A következőkben megmutatjuk a relációs algebra műveleteit.

2.4.3. A relácós algebra áttekintése

A hagyományos relációs algebrai műveletek négy osztályba sorolhatók:

a) A hagyományos halmazműveletek – egyesítés, metszet és különbség – relációkra alkalmazva.

b) Műveletek, amelyek a reláció egyes részeit eltávolítják: a „kiválasztás" kihagy bizonyos sorokat, a „vetítés" bizonyos oszlopokat hagy ki.

c) Műveletek, amelyek két reláció sorait kombinálják: a „Descartes-szorzat", amely a relációk sorait párosítja az összes lehetséges módon, és a különböző típusú „összekapcsolási" műveletek, amelyek szelektíven párosítják össze a két reláció sorait.

d) Egy művelet, az „átnevezés", amelyik nem befolyásolja a reláció sorait, de megváltoztatja a reláció sémáját, azaz az attribútumok neveit és/vagy a reláció nevét.

A relációs algebrai kifejezésekre általában *lekérdezések*ként hivatkozunk.

2.4.4. Relációkon értelmezett halmazműveletek

Halmazokon a három leggyakoribb művelet az egyesítés, a metszet és a különbség. Feltételezzük, hogy az olvasó ismeri ezeket a műveleteket, amelyeket a következő módon definiálunk tetszőleges R és S halmazok esetén:

- $R \cup S$, R és S *egyesítése* azon elemek halmaza, amelyek vagy az R-ben vagy az S-ben vannak. Egy elem csak egyszer szerepel az egyesítésben, még akkor is, ha jelen van R-ben is és S-ben is.
- $R \cap S$, R és S *metszete* azon elemek halmaza, amelyek az R-ben és az S-ben is benne vannak.
- $R - S$, R és S *különbsége* azon elemek halmaza, amelyek benne vannak R-ben, de nincsenek S-ben. Figyeljük meg, hogy $R - S$ nem ugyanaz, mint $S - R$; az utóbbi azon elemek halmaza, amelyek benne vannak S-ben, de nincsenek R-ben.

Ezen műveletek relációkra történő alkalmazásakor néhány feltételt kell szabnunk:

1. Az R és S relációk sémájának ugyanazt az attribútumhalmazt kell tartalmazniuk, illetve a típusoknak (értéktartományoknak) az összes megfelelő attribútumpárra meg kell egyezniük R-ben és S-ben.

2. Mielőtt kiszámolnánk a halmazelméleti egyesítését, metszetét vagy különbségét a sorhalmazoknak, az R és S oszlopait rendezni kell úgy, hogy az attribútumok sorrendje egyforma legyen mindkét reláció esetén.

Néha szeretnénk kiszámolni két olyan reláció egyesítését, metszetét vagy különbségét, amelyek attribútumainak a száma megegyezik, viszont az attribútumok neve különbözik. Ebben az esetben az egyik vagy mindkét reláció sémájának megváltoztatásához használhatjuk a 2.4.11. alfejezetben bemutatott átnevezés operátort, és így ugyanazt az attribútumhalmazt tudjuk adni a relációknak.

név	*cím*	*nem*	*születésiDátum*
Carrie Fisher	123 Maple St., Hollywood	N	9/9/99
Mark Hamill	456 Oak Rd., Brentwood	F	8/8/88

Az R reláció

név	*cím*	*nem*	*születésiDátum*
Carrie Fisher	123 Maple St., Hollywood	N	9/9/99
Harrison Ford	789 Palm Dr., Beverly Hills	F	7/7/77

Az S reláció

2.12. ábra. Két reláció

2.8. példa. Tegyük fel, hogy van két relációnk, R és S, amelyek a 2.2.8. alfejezetben található `FilmSzínész` reláció előfordulásai. R és S aktuális előfordulásait a 2.12. ábrán láthatjuk. R és S egyesítése, $R \cup S$ a következő reláció:

név	*cím*	*nem*	*születésiDátum*
Carrie Fisher	123 Maple St., Hollywood	N	9/9/99
Mark Hamill	456 Oak Rd., Brentwood	F	8/8/88
Harrison Ford	789 Palm Dr., Beverly Hills	F	7/7/77

Figyeljük meg, hogy a Carrie Fisher adatait tartalmazó sor csak egyszer jelenik meg az eredményben annak ellenére, hogy mindkét relációban szerepelt.

Az $R \cap S$ metszet:

név	*cím*	*nem*	*születésiDátum*
Carrie Fisher	123 Maple St., Hollywood	N	9/9/99

Most csak a Carrie Fisher adatait tartalmazó sor jelenik meg, mert csak ez a sor szerepel mindkét relációban.

Az $R - S$ különbség:

név	*cím*	*nem*	*születésiDátum*
Mark Hamill	456 Oak Rd., Brentwood	F	8/8/88

R-ben a Fisher és Hamill adatait tartalmazó sorok szerepelnek, ezért ők mindketten szerepelhetnének az $R - S$ különbségben. Viszont a Fisher adatait tartalmazó sor szerepel S-ben is, ezért nem szerepelhet az $R - S$ különbségben. □

2.4.5. Vetítés

A *vetítés* operátorral a régi R relációból olyan új reláció hozható létre, amelyik csak R bizonyos oszlopait tartalmazza. A $\pi_{A_1,A_2,\ldots,A_n}(R)$ kifejezés értéke az a reláció, amelyik az R relációnak csak az $A_1, A_2, \ldots, A_n$ attribútumokhoz tartozó oszlopait tartalmazza. Az eredmény sémája az $\{A_1, A_2, \ldots, A_n\}$ attribútumhalmaz, amelyet mi megegyezés szerint egy rendezett listával jelölünk.

filmcím	*év*	*hossz*	*műfaj*	*stúdióNév*	*producerA.*
Csillagok háborúja	1977	124	sci-fi	Fox	12345
Galaktitkos küldetés	1999	104	vígjáték	DreamWorks	67890
Wayne világa	1992	95	vígjáték	Paramount	99999

2.13. ábra. A Filmek reláció

2.9. példa. Tekintsük a 2.2.8. alfejezetben megadott sémájú Filmek relációt. A reláció egy előfordulását a 2.13. ábrán láthatjuk. Ezt a relációt a következő kifejezés segítségével vetíthetjük az első három attribútumára:

$$\pi_{\text{filmcím,év,hossz}}(\text{Filmek})$$

Az eredményül kapott reláció a következő:

filmcím	*év*	*hossz*
Csillagok háborúja	1977	124
Galaktitkos küldetés	1999	104
Wayne világa	1992	95

Megjegyzés az adatok minőségéről :-)

Miközben felettébb odafigyeltünk arra, hogy a példáinkban minél pontosabb adatokat adjunk meg, ennek ellenére fiktív értékeket adtunk a lakcímekre és személyes adatokra nézve azért, hogy az előadóművészek személyes jogait ne sértsük meg. Hiszen sokan közülük érzékenyek lehetnek a magánszférájukra.

Másik példaként levetíthetjük a `Filmek` relációt a műfaj attribútumra a $\pi_{\texttt{műfaj}}(\texttt{Filmek})$ kifejezéssel. Ekkor az eredményül kapott reláció csak egyetlen oszlopot tartalmaz:

műfaj
`sci-fi`
`vígjáték`

Figyeljük meg, hogy az eredményben csak két sor található, mivel a 2.13. ábrán látható utolsó két sor `műfaj` attribútumának értéke ugyanaz. □

2.4.6. Kiválasztás

Az R relációra alkalmazott *kiválasztás* operátor olyan új relációt hoz létre, amely R sorainak egy részhalmazát tartalmazza. Az eredménybe azok a sorok kerülnek, amelyek teljesítenek egy adott, R attribútumaira megfogalmazott C feltételt. Ezt a műveletet a $\sigma_C(R)$ kifejezéssel jelöljük. Az eredményreláció sémája megegyezik R sémájával, és megegyezés szerint az attribútumokat ugyanabban a sorrendben tüntetjük fel, mint ahogyan az R relációban használtuk.

A C egy olyan feltételkifejezés, amilyet megszoktunk a hagyományos programozási nyelveknél. Például az `if` kulcsszót feltételkifejezés követi mind a C, mind pedig a Java programozási nyelvekben. Az egyetlen különbség az, hogy a C feltételben levő operandusok vagy konstansok, vagy az R attribútumai. Alkalmazzuk a C feltételt R minden egyes t sorára oly módon, hogy a C-ben előforduló minden egyes A attribútumra behelyettesítjük a t sornak az A attribútumhoz tartozó komponensét. Ha a C feltétel összes attribútumát behelyettesítve C értéke igaz, akkor a t sor egyike azoknak a soroknak, amelyek megjelennek a $\sigma_C(R)$ eredményében; egyébként t nincs az eredményben.

2.10. példa. Tekintsük a 2.13. ábrán látható `Filmek` relációt. Ebben az esetben a $\sigma_{\texttt{hossz}\geq 100}(\texttt{Filmek})$ kifejezés értéke a következő:

filmcím	*év*	*hossz*	*műfaj*	*stúdióNév*	*producerA.*
Csillagok háborúja	1977	124	sci-fi	Fox	12345
Galaktitkos küldetés	1999	104	vígjáték	DreamWorks	67890

Az első sor kielégíti a `hossz` ≥ 100 feltételt, hiszen ha behelyettesítjük a `hossz` helyébe az első sor megfelelő komponensét, a 124-et, akkor a feltétel így néz ki: $124 \geq 100$. Ez utóbbi feltétel igaz, ezért az első sort elfogadjuk. Ugyanilyen indokok magyarázzák, hogy a 2.13. ábra második sora miért kerül be az eredménybe.

A harmadik sor `hossz` komponense 95. Ezért, amikor behelyettesítünk a `hossz` attribútumba, a $95 \geq 100$ hamis feltételt kapjuk. Ennélfogva a 2.13. ábra utolsó sora nincs az eredményben. □

2.11. példa. Tegyük fel, hogy a `Filmek` relációból azon sorok halmazát szeretnénk megkapni, amelyek a Fox stúdió legalább 100 perces filmjeit tartalmazzák. Ezeket a sorokat egy olyan bonyolultabb feltétel segítségével kaphatjuk meg, amelyet két részfeltétel `AND` összekapcsolásával nyerünk. A keresett kifejezés:

$$\sigma_{\texttt{hossz}\geq 100\ \texttt{AND}\ \texttt{stúdióNév}=\texttt{'Fox'}}(\texttt{Filmek})$$

Az eredmény:

filmcím	*év*	*hossz*	*műfaj*	*stúdióNév*	*producerAzon*
Csillagok háborúja	1977	124	sci-fi	Fox	12345

Az eredményrelációnak egyetlenegy sora van. □

2.4.7. Descartes-szorzat

Két halmaz, R és S *Descartes-szorzata* (vagy *direkt szozata* vagy egyszerűen csak *szorzata*) azon párok halmaza, amelyeknek első eleme R tetszőleges eleme, a második pedig S egy eleme. A szorzat jelölése $R \times S$. Amikor R és S relációk, a szorzat lényegéből adódóan szintén reláció. Mivel azonban R és S elemei sorok, mégpedig általában egynél több komponensből álló sorok, ezért R egy sorának párosítása S egy sorával olyan hosszabb sort eredményez, amelyben az alkotó sorok mindegyik komponense megjelenik. R (a bal oldali operandus) attribútumai megelőzik sorrendben S attribútumait.

Az eredményreláció sémája R és S sémájának egyesítése. Azonban előfordulhat, hogy az R és S relációknak vannak közös attribútumai. Ekkor minden azonos nevű attribútumot tartalmazó pár esetén legalább az egyiknek új nevet kell adni. Egy olyan A attribútum egyértelművé tételéhez, amelyik mind az R, mind az S sémájában szerepel, a nevek megkülönböztetésére az $R.A$, illetve $S.A$ jelöléseket használjuk attól függően, hogy az R reláció A attribútumáról vagy az S reláció A attribútumáról van szó.

A	B
1	2
3	4

(a) Az R reláció

B	C	D
2	5	6
4	7	8
9	10	11

(b) Az S reláció

A	$R.B$	$S.B$	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

(c) Az $R \times S$ eredménye

2.14. ábra. Két reláció és a Descartes-szorzatuk

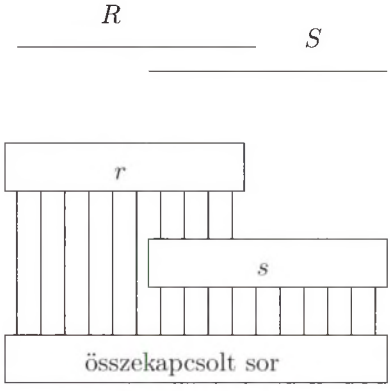
2.12. példa. A tömörség kedvéért használjunk egy absztrakt példát a szorzásművelet szemléltetésére. Tekintsük a 2.14 (a), illetve a 2.14 (b) ábrán látható R és S relációkat, az ábrán megadott sémákkal és sorokkal. Amint az a 2.14 (c) ábrán is látható, az $R \times S$ szorzat hat sorból áll. Figyeljük meg, hogyan párosítottuk R két sorának mindegyikét S három sorának mindegyikével. Mivel a B mindkét sémának attribútuma, ezért az $R.B$ és $S.B$ jelöléseket használtuk az $R \times S$ sémájában. A többi attribútum egyértelmű, ezeknek a nevei változatlanok az eredmény sémájában. □

2.4.8. Természetes összekapcsolás

Két reláció szorzásánál jóval gyakrabban van szükségünk arra, hogy *összekapcsoljunk* relációkat oly módon, hogy csak azokat a sorokat párosítjuk, amelyek valamilyen módon összeillenek. Az összeillesztés legegyszerűbb módja két reláció *természetes összekapcsolás*a, amelynek jelölése $R \bowtie S$, és amelyben R-nek és S-nek csak azokat a sorait párosítjuk össze, amelyek értékei megegyeznek R és S sémájának összes közös attribútumán. Még pontosabban: legyenek

$A_1, A_2, \ldots, A_n$ azok az attribútumok, amelyek megtalálhatók mind az R, mind az S sémájában. R egy r sorának és S egy s sorának párosítása akkor és csak akkor sikeres, ha r és s megfelelő értékei megegyeznek az összes $A_1, A_2, \ldots, A_n$ attribútumon.

Ha az r és s sorok párosítása az $R \bowtie S$ összekapcsolásban sikeres, akkor a párosítás eredménye egy olyan, *összekapcsolt sor*nak nevezett sor, amelyben az R és S sémájának egyesítésében szereplő összes attribútumhoz egyetlen komponens tartozik. Az összekapcsolt sor megegyezik az r sorral az R összes attribútumán, és megegyezik az s sorral az S összes attribútumán. Mivel r és s összekapcsolása sikeres volt, így tudjuk, hogy r és s megegyezik azokon az attribútumokon, amelyek mind az R, mind pedig az S sémájában szerepelnek. Ezért az összekapcsolt sor is megegyezik mind az r, mind az s sorral a mindkét sémában szereplő attribútumokon. Az összekapcsolt sorok szerkezetét szemlélteti a 2.15. ábra. Az attribútumok sorrendjének viszont nem feltétlenül kell megegyezni R és S attribútumainak a sorrendjével.



2.15. ábra. Sorok összekapcsolása

2.13. példa. A 2.14. ábra (a) és (b) részén látható R és S relációk természetes összekapcsolásának eredménye:

A	B	C	D
1	2	5	6
3	4	7	8

R és S egyetlen közös attribútuma a B. Ennek következtében a sorok sikeres párosításához elegendő, hogy a sorok B attribútumán levő értékek megegyezzenek. Ekkor az eredménysornak lesz egy komponense az A attribútumhoz (R-ből), a B attribútumhoz (R-ből vagy S-ből), a C attribútumhoz (S-ből) és a D attribútumhoz (S-ből).

Ebben a példában R első sora csak az S első sorával párosítható sikeresen; mindkét sor B attribútumának értéke 2. Ennek a párosításnak az eredménye az első sor: $(1, 2, 5, 6)$. R második sora csak az S második sorával párosítható sikeresen, és ez a párosítás a $(3, 4, 7, 8)$ sort eredményezi. Megfigyelhetjük, hogy S harmadik sora nem párosítható R egyetlen sorával sem, ezért nincs is semmi hatása az $R \bowtie S$ eredményére nézve. Az olyan sort, amelyet nem lehet sikeresen párosítani az összekapcsolásban szereplő másik reláció egyetlen sorával sem, *lógó sor*nak is nevezzük. □

2.14. példa. Az előző példa nem szemlélteti a természetes összekapcsolással felmerülő összes lehetőséget. Így például nem volt olyan sor, amelyik egynél több sorral is párosítható lett volna, és a két reláció sémájának csak egyetlen egy közös attribútuma volt. A 2.16. ábrán látható U és V relációk sémájának két közös attribútuma van, a B és a C. Ráadásul az ábrán szereplő egyik előfordulásnak van egy olyan sora, amelyik több sorral is összekapcsolható.

A sikeresen párosítható sorok B és C komponenseinek is egyeznie kell. Ekképpen U első sora sikeresen párosítható a V első két sorával, viszont U második és harmadik sora csak a V harmadik sorával párosítható sikeresen. A négy párosítás eredménye a 2.16 (c) ábrán látható. □

2.4.9. Théta-összekapcsolás

A természetes összekapcsolás előírja, hogy egyetlen speciális feltétel szerint párosítsuk a sorokat. Bár a relációk összekapcsolásának leggyakoribb kiindulási pontja a közös attribútumokban levő értékek egyenlővé tétele, néha szükség lehet két reláció sorainak más szempontból történő párosítására. Ezért bevezetjük a *théta-összekapcsolás* műveletet: a „théta" egy tetszőleges feltételre utal, amit mi θ helyett inkább C-vel jelölünk.

R és S relációknak C feltételre vonatkozó théta-összekapcsolásának jelölése $R \bowtie_C S$. Ennek a műveletnek az eredményét a következő módon kapjuk:

1. Kiszámoljuk R és S szorzatát.
2. Kiválasztjuk a szorzatból azokat a sorokat, amelyek eleget tesznek a C feltételnek.

Éppúgy, mint a szorzásműveletnél, az eredmény sémája itt is az R és S sémáinak egyesítése. Ha szükséges megjelölni, hogy az attribútumok melyik sémából származnak, akkor használjuk az attribútumokhoz az „R.", illetve „S." előtagokat.

2.15. példa. Tekintsük az $U \bowtie_{A<D} V$ műveletet, ahol U és V a 2.16. ábra (a) és (b) részében látható két reláció. Figyelembe kell vennünk a relációk sorainak mind a kilenc lehetséges párosítását, és meg kell néznünk, hogy vajon U sorának A komponense kisebb-e, mint V sorának D komponense. U első sorában az A komponens értéke 1, és ez a sor sikeresen párosítható V összes sorával.

A	B	C
1	2	3
6	7	8
9	7	8

(a) Az U reláció

B	C	D
2	3	4
2	3	5
7	8	10

(b) A V reláció

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

(c) Az $U \bowtie V$ eredménye

2.16. ábra. Relációk természetes összekapcsolása

U második és harmadik sorában az A komponens értéke 6, illetve 9, ezért ezek csak a V utolsó sorával párosíthatók sikeresen. Az öt sikeres párosításból eredően az eredménynek öt sora lesz. Az eredményreláció a 2.17. ábrán látható. □

A	$U.B$	$U.C$	$V.B$	$V.C$	D
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

2.17. ábra. Az $U \bowtie_{A<D} V$ eredménye

Figyeljük meg, hogy a 2.17. ábrán látható eredmény sémájában mind a hat attribútum szerepel. A B és C attribútumok a megfelelő előtagokkal szerepelnek, megkülönböztetendő, hogy az U vagy a V attribútumai. Tehát a théta-összekapcsolás különbözik a természetes összekapcsolástól, hiszen ez utóbbiban a közös attribútumok csak egyszer szerepelnek. Természetesen ennek csak a természetes összekapcsolásnál van értelme, hiszen ott csak azokat a sorokat párosítjuk, amelyek megegyeznek a közös attribútumokon. A théta-összekapcsolás esetében az összehasonlító operátor más is lehet, mint az $=$, éppen ezért nem biztos, hogy az összehasonlított attribútumok megegyeznek az eredményben.

2.16. példa. Vegyük a korábbi U, V relációk théta-összekapcsolását egy összetettebb feltétellel:

$$U \bowtie_{A<D \text{ AND } U.B \neq V.B} V$$

A sikeres párosításhoz már nem elég, hogy U sorának A komponense kisebb legyen, mint V sorának D komponense, hanem annak is teljesülnie kell, hogy a két sor értéke legyen különböző a B attribútumokon. Az egyetlen sor, ami teljesíti ezt a feltételt, a következő:

A	$U.B$	$U.C$	$V.B$	$V.C$	D
1	2	3	7	8	10

Tehát a théta-összekapcsolás eredménye a fenti egyetlenegy sort tartalmazó reláció. □

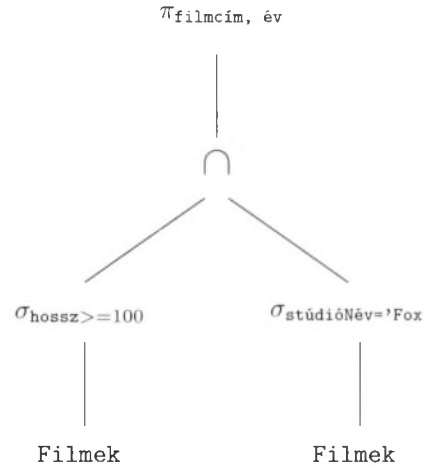
2.4.10. Lekérdezések megfogalmazása műveletek segítségével

Ha lekérdezéseket csak a műveletek egy vagy két relációra történő alkalmazásával fogalmazhatnánk meg, akkor a relációs algebra nem lenne annyira hasznos, mint amilyen hasznos valójában. Azonban a relációs algebra is, mint minden algebra, lehetőséget ad arra, hogy tetszőleges bonyolultságú kifejezéseket képezzünk. Ekkor az operátorokat vagy az adott relációkra alkalmazzuk, vagy olyan relációkra, amelyek más operátorok relációkra történő alkalmazásának eredményei.

Relációs algebrai kifejezések megadásakor, amikor a műveleteket részkifejezésekre alkalmazzuk, használhatunk zárójeleket az operandusok csoportosításának egyértelművé tételére. Lehetséges a kifejezések kifejezésfával történő megadása is: ez utóbbit nekünk könnyebb olvasni, de géppel kevésbé olvasható.

2.17. példa. Tekintsük az általunk megadott `Filmek` relációt. Tegyük fel, hogy a következő kérdésre szeretnénk megkapni a választ: „Melyek a Fox stúdióban készült, legalább 100 perc hosszúságú filmek, és ezek mikor készültek?" A kérdés megválaszolására az egyik lehetőség:

1. Kiválasztjuk a `Filmek` relációból azokat a sorokat, amelyekre `hossz` $\geq$ 100.
2. Kiválasztjuk a `Filmek` relációból azokat a sorokat, amelyekre a `stúdióNév` = `'Fox'`.
3. Kiszámoljuk az 1. és 2. metszetét.
4. A 3. lépésben megkapott relációt levetítjük a `filmcím` és `év` attribútumokra.



2.18. ábra. Egy relációs algebrai kifejezés kifejezésfája

A 2.18. ábrán a fenti lépéseknek megfelelő kifejezésfát láthatjuk. A kifejezésfa kiértékelése alulról felfelé történik. A belső csúcsokban az argumentumokra (a gyerekekből kapott eredményekre) alkalmazzuk a megfelelő műveleteket. Az alulról felfelé kiértékelésnél az argumentumok a megfelelő részeknél mindig rendelkezésre fognak állni. A két kiválasztást tartalmazó csúcs az 1. és 2. lépéseknek felel meg. A metszetet tartalmazó csúcs a 3. lépésnek felel meg, a vetítést tartalmazó csúcs pedig a 4. lépés.

Ugyanezt a kifejezést felírhatjuk zárójelek használatával a hagyományos lineáris jelöléssel is. A következő formula ugyanazt a kifejezést jelenti.

$$\pi_{\texttt{filmcím,év}}\big(\sigma_{\texttt{hossz}\geq 100}(\texttt{Filmek}) \cap \sigma_{\texttt{stúdióNév='Fox'}}(\texttt{Filmek})\big).$$

Egyébként gyakori, hogy több relációs algebrai kifejezésnek is ugyanaz az eredménye. Például a fenti lekérdezés felírható egyetlen kiválasztás használatával, ha a metszetet az `AND` operátorral helyettesítjük. A következő kifejezés

$$\pi_{\texttt{filmcím,év}}\big(\sigma_{\texttt{hossz}\geq 100\ \texttt{AND}\ \texttt{stúdióNév='Fox'}}(\texttt{Filmek})\big)$$

ekvivalens az előző kifejezéssel. □

Ekvivalens kifejezések és lekérdezések optimalizálása

Minden adatbázisrendszernek van egy lekérdezés-válaszoló rendszere, ahol a lekérdezés legtöbbször olyan nyelvre épül, amely kifejező erejét tekintve hasonló a relációs algebrához. Éppen ezért a felhasználó által megfogalmazott lekérdezéshez létezhet több *ekvivalens kifejezés* (az ekvivalens kifejezések olyan kifejezések, amelyeket ha ugyanazokon a relációkon értékelünk ki, ugyanazt az eredményt adják). Ezek között vannak olyanok, amelyek gyorsabban kiértékelhetők. Az 1.2.5. alfejezetben vázlatosan tárgyalt lekérdezés-válaszolónak egyik fontos feladata éppen az, hogy egy relációs algebrai kifejezést olyan ekvivalens kifejezéssel helyettesítsen, amely hatékonyabban értékelhető ki.

2.4.11. Elnevezés és átnevezés

Ahhoz, hogy a relációk attribútumainak nevét szabályozni tudjuk, gyakran szükséges egy olyan operátor használata, amelyik kifejezetten átnevezi a relációkat. Egy R reláció átnevezéséhez a $\rho_{S(A_1,A_2,\ldots,A_n)}(R)$ operátort használjuk. Az eredményreláció neve S, sorai azonban megegyeznek R soraival, és attribútumainak neve balról jobbra: $A_1, A_2, \ldots, A_n$. Ha az attribútumok neveit meg akarjuk őrizni és csak a reláció nevét szeretnénk megváltoztatni, akkor egyszerűen csak annyit írunk: $\rho_S(R)$.

2.18. példa. A 2.12. példában kiszámoltuk a 2.14. ábra (a) és (b) részében látható két reláció, R és S szorzatát. Megállapodtunk abban is, hogy ha egy attribútum mindkét operandusban szerepel, akkor ezeket az attribútumokat átnevezzük oly módon, hogy az új névben az attribútumokhoz tartozó relációk neve is jelenjen meg előtagként. Tegyük fel azonban, hogy a B attribútum két előfordulását nem $R.B$, illetve $S.B$ névvel szeretnénk illetni, hanem az R reláció B attribútumának a nevét szeretnénk megőrizni, az S reláció B attribútumát pedig X névvel szeretnénk illetni. E célból átnevezhetjük S attribútumait úgy, hogy az elsőnek a neve legyen X. A $\rho_{S(X,C,D)}(S)$ kifejezés eredménye pontosan egy olyan S nevű reláció, amelyik ugyanolyan, mint a 2.14. ábrán látható S reláció, csak az első attribútumának a neve nem B, hanem X.

Amikor megszorozzuk az R relációt ezzel az új relációval, már nem lesz gond az egyforma attribútumnevekkel, további átnevezésekre így nincs szükség. Vagyis az $R \times \rho_{S(X,C,D)}(S)$ kifejezés eredménye a 2.14 (c) ábrán látható $R \times S$ reláció, kivéve, hogy az öt attribútum neve balról jobbra: A, B, X, C és D. A 2.19. ábrán ez a reláció látható.

Egy másik lehetőség az, hogy amint a 2.12. példában is tettük, kiszámoljuk a szorzatot átnevezés nélkül, ezt követően pedig átnevezzük az eredményt. A

$$\rho_{RS(A,B,X,C,D)}(R \times S)$$

A	B	X	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

2.19. ábra. Az $R \times \rho_{S(X,C,D)}(S)$ eredménye

kifejezés eredményének ugyanazok az attribútumai, mint a 2.19. ábrán látható relációnak, viszont a neve RS, míg a 2.19. ábrán látható relációnak nincs neve. □

2.4.12. Műveletek közötti kapcsolatok

A 2.4. alfejezetben bemutatott műveletek között vannak olyanok, amelyek kifejezhetők más relációs algebrai műveletek segítségével. Például a metszet kifejezhető halmazok különbségével:

$$R \cap S = R - (R - S)$$

Vagyis, ha R és S két, egyforma sémával rendelkező reláció, akkor R és S metszete kiszámolható úgy, hogy először kiszámoljuk azt a T relációt, amelyben mindazok a sorok benne vannak, amelyek R-ben benne vannak, de S-nek nem sorai. Azaz kivonjuk R-ből S-et. Ezután kivonjuk R-ből T-t, ezáltal R-nek azokat a sorait kapjuk, amelyek S-ben is benne vannak.

A két összekapcsolás szintén kifejezhető más műveletek segítségével. A théta-összekapcsolás szorzás és kiválasztás segítségével fejezhető ki:

$$R \bowtie_C S = \sigma_C(R \times S)$$

R és S természetes összekapcsolásának kifejezéséhez először vegyük az $R \times S$ szorzatot. Ezután alkalmazzuk a kiválasztás operátort a következő alakú C feltétellel:

$$R.A_1 = S.A_1 \texttt{ AND } R.A_2 = S.A_2 \texttt{ AND} \cdots \texttt{AND } R.A_n = S.A_n$$

ahol $A_1, A_2, \ldots, A_n$ olyan attribútumok, amelyek mind az R, mind az S sémájában szerepelnek. Utolsó lépésként pedig mindegyik közös attribútumból csak egy példányt őrizzünk meg. Legyen L egy olyan attribútumlista, amelyben szerepel R összes attribútuma és emellett S-ből mindazok az attribútumok, amelyek nincsenek benne R sémájában. Ekkor:

$$R \bowtie S = \pi_L\big(\sigma_C(R \times S)\big)$$

2.19. példa. A 2.16. ábrán látható U és V relációk természetes összekapcsolása felírható szorzás, kiválasztás és vetítés segítségével a következőképpen:

$$\pi_{A,U.B,U.C,D}(\sigma_{U.B=V.B \text{ AND } U.C=V.C}(U \times V))$$

Vagyis vesszük az $U \times V$ szorzatot. Ezután kiválasztjuk azokat a sorokat, amelyekben az egyforma nevű – jelen esetben a B és a C – attribútumokhoz tartozó értékek egyenlők. Végül pedig egy vetítést végzünk az összes attribútumra, kivéve egy B és egy C attribútumot: választásunk azon V-attribútumok elhagyására esett, amelyek benne vannak az U sémájában.

Egy másik példa a 2.16. példában kiszámolt théta-összekapcsolás átírása:

$$\sigma_{A<D \text{ AND } U.B \neq V.B}(U \times V)$$

Vagyis kiszámoljuk az U és V szorzatát, és utána alkalmazunk egy kiválasztást a théta-összekapcsolásban szereplő feltétel szerint. □

Az ebben az alfejezetben említett redundanciák csak a bevezetett műveletek közötti „redundanciák". A fennmaradó hat operátor – egyesítés, különbség, kiválasztás, vetítés, szorzat és átnevezés – független halmazt alkot, egyik sem fejezhető ki a másik öt segítségével.

2.4.13. Egy lineáris jelölési mód az algebrai kifejezésekhez

A 2.4.10. alfejezetben a kifejezésfákat használtuk az összetett relációs algebrai kifejezések leírására. Egy másik megközelítés az, hogy a belső csúcsoknak megfelelő ideiglenes relációknak nevet adunk, és egy értékadási sorozatot adunk meg az értékeik meghatározására. Az értékadások sorrendje rugalmas abban az értelemben, hogy ha az N csúcs összes gyerekének értéke már adott, akkor már kiszámíthatjuk N értékét is.

Az értékadó utasításokat a következőképpen jelölhetjük:

1. Először szerepel a reláció neve és a reláció attribútumainak zárójelezett listája. A `Válasz` nevet az utolsó lépés eredményének jelölésére használjuk, azaz a kifejezésfa gyökeréhez tartozó reláció neveként.

2. Ezt követi a `:=` értékadási szimbólum.

3. A jobb oldalon pedig egy algebrai kifejezés áll. Minden értékadásnál használhatunk akár egyetlen műveletet is, ekkor tulajdonképpen a kifejezésfa minden csúcsának pontosan egy értékadás felel meg. Viszont kombinálhatjuk is a jobb oldalon a relációs algebrai műveleteket, ami szintén egy elfogadott módszer.

2.20. példa. Vegyük a 2.18. ábra kifejezésfáját. Az értékadások egy lehetséges sorozata a kiértékeléshez a következő lehetne:

$$\begin{aligned}
&\texttt{R(fc,é,h,m,s,p)} := \sigma_{\texttt{hossz}\geq 100}(\texttt{Filmek})\\
&\texttt{S(fc,é,h,m,s,p)} := \sigma_{\texttt{stúdióNév='Fox'}}(\texttt{Filmek})\\
&\texttt{T(fc,é,h,m,s,p)} := \texttt{R} \cap \texttt{S}\\
&\texttt{Válasz(filmcím,év)} := \pi_{\texttt{fc,é}}(\texttt{T})
\end{aligned}$$

Az első lépésben a 2.18. ábra $\sigma_{\texttt{hossz}\geq 100}$ címkéjű belső csúcsát számítjuk ki. A második lépésben pedig a $\sigma_{\texttt{stúdióNév='Fox'}}$ értékét számítjuk. Megjegyezzük, hogy „ingyen" átnevezést kaphatunk, hiszen az értékadás bal oldalán tetszőleges relációnevet és attribútumnevet is használhatnánk. A fennmaradó két lépésben természetesen a metszetet és a vetítést végezzük el.

A lépések összevonása is megengedett. Például az utolsó két lépést össze is vonhatnánk:

$$\begin{aligned}
&\texttt{R(fc,é,h,m,s,p)} := \sigma_{\texttt{hossz}\geq 100}(\texttt{Filmek})\\
&\texttt{S(fc,é,h,m,s,p)} := \sigma_{\texttt{stúdióNév='Fox'}}(\texttt{Filmek})\\
&\texttt{Válasz(filmcím,év)} := \pi_{\texttt{fc,é}}(\texttt{R} \cap \texttt{S})
\end{aligned}$$

Akár R-et és S-et is beírhatnánk az utolsó sorba, és így egy egysoros kifejezést kaphatnánk. □

2.4.14. Feladatok

2.4.1. feladat. Ez a feladat a 2.3.1. feladatban szereplő termékek sémáján alapszik. Itt megismételjük azt az adatbázissémát, amely négy táblán alapult. Ezek sémája az alábbi volt:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

A `Termék` reláció néhány mintaadata a 2.20. ábrán látható. A másik három reláció mintaadatai a 2.21. ábrán láthatók. A gyártók, illetve a modellszámok fiktív adatok, de a többi adat az 2007. év elejének piacát tükrözi.

gyártó	*modell*	*típus*
A	1001	pc
A	1002	pc
A	1003	pc
A	2004	laptop
A	2005	laptop
A	2006	laptop
B	1004	pc
B	1005	pc
B	1006	pc
B	2007	laptop
C	1007	pc
D	1008	pc
D	1009	pc
D	1010	pc
D	3004	nyomtató
D	3005	nyomtató
E	1011	pc
E	1012	pc
E	1013	pc
E	2001	laptop
E	2002	laptop
E	2003	laptop
E	3001	nyomtató
E	3002	nyomtató
E	3003	nyomtató
F	2008	laptop
F	2009	laptop
G	2010	laptop
H	3006	nyomtató
H	3007	nyomtató

2.20. ábra. A Termék reláció mintaadatai

modell	*sebesség*	*memória*	*merevlemez*	*ár*
1001	2.66	1024	250	2114
1002	2.10	512	250	995
1003	1.42	512	80	478
1004	2.80	1024	250	649
1005	3.20	512	250	630
1006	3.20	1024	320	1049
1007	2.20	1024	200	510
1008	2.20	2048	250	770
1009	2.00	1024	250	650
1010	2.80	2048	300	770
1011	1.86	2048	160	959
1012	2.80	1024	160	649
1013	3.06	512	80	529

(a) A PC reláció mintaadatai

modell	*sebesség*	*memória*	*merevlemez*	*képernyő*	*ár*
2001	2.00	2048	240	20.1	3673
2002	1.73	1024	80	17.0	949
2003	1.80	512	60	15.4	549
2004	2.00	512	60	13.3	1150
2005	2.16	1024	120	17.0	2500
2006	2.00	2048	80	15.4	1700
2007	1.83	1024	120	13.3	1429
2008	1.60	1024	100	15.4	900
2009	1.60	512	80	14.1	680
2010	2.00	2048	160	15.4	2300

(b) A Laptop reláció mintaadatai

modell	*színes*	*típus*	*ár*
3001	igen	tintasugaras	99
3002	nem	lézer	239
3003	igen	lézer	899
3004	igen	tintasugaras	120
3005	nem	lézer	120
3006	igen	tintasugaras	100
3007	igen	lézer	200

(c) A Nyomtató reláció mintaadatai

2.21. ábra. A 2.4.1. feladat relációinak mintaadatai

Írjuk fel a következő lekérdezésekhez tartozó relációs algebrai kifejezéseket. A 2.4.13. alfejezetben bemutatott lineáris jelölésmód használható. A 2.20. és a 2.21. ábrán látható adatok alapján számoljuk ki a lekérdezések eredményeit. A válaszul felírt kifejezéseknek más adatokra is a helyes választ kell megadniuk, nem csak az ábrán látható adatokra.

a) Melyek azok a PC-modellek, amelyek sebessége legalább 3.00?

b) Mely gyártók készítenek legalább száz gigabájt méretű merevlemezzel rendelkező laptopot?

c) Adjuk meg a B gyártó által gyártott összes termék modellszámát és árát, típustól függetlenül.

d) Adjuk meg valamennyi színes lézernyomtató modellszámát.

e) Melyek azok a gyártók, amelyek laptopot árulnak, PC-t viszont nem?

! *f)* Melyek azok a merevlemezméretek, amelyek legalább két PC-ben megtalálhatók?

! *g)* Adjuk meg azokat a PC-modell párokat, amelyek ugyanolyan gyorsak és a memóriájuk is ugyanakkora. Egy pár csak egyszer jelenjen meg, azaz ha (i, j) már szerepel, akkor (j, i) ne jelenjen meg.

!! *h)* Melyek azok a gyártók, amelyek gyártanak legalább két, egymástól különböző, legalább 2.80 gigahertzen működő számítógépet (PC-t vagy laptopot)?

!! *i)* Melyik gyártó gyártja a leggyorsabb számítógépet (PC-t vagy laptopot)?

!! *j)* Melyik gyártó gyárt legalább három, különböző sebességű PC-t?

!! *k)* Melyek azok a gyártók, amelyek pontosan három típusú PC-t forgalmaznak?

2.4.2. feladat. A 2.4.1. feladatban megfogalmazott lekérdezéseknek rajzoljuk fel a kifejezésfáit is.

2.4.3. feladat. Ez a példa, a 2.3.2. feladatban megfogalmazott témával, a II. világháború csatahajóival foglalkozik, és a következő sémájú relációkat tartalmazza:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

osztály	*típus*	*ország*	*ágyúkSzáma*	*kaliber*	*vízkiszorítás*
Bismarck	bb	Németország	8	15	42000
Iowa	bb	USA	9	16	46000
Kongo	bc	Japán	8	14	32000
North Carolina	bb	USA	9	16	37000
Renown	bc	Nagy-Britannia	6	15	32000
Revenge	bb	Nagy-Britannia	8	15	29000
Tennessee	bb	USA	12	14	32000
Yamato	bb	Japán	9	18	65000

(a) A Hajóosztályok reláció mintaadatai

név	*dátum*
Denmark Strait	5/24-27/41
Guadalcanal	11/15/42
North Cape	12/26/43
Surigao Strait	10/25/44

(b) A Csaták reláció mintaadatai

hajó	*csata*	*eredmény*
Arizona	Pearl Harbor	elsüllyedt
Bismarck	Denmark Strait	elsüllyedt
California	Surigao Strait	ép
Duke of York	North Cape	ép
Fuso	Surigao Strait	elsüllyedt
Hood	Denmark Strait	elsüllyedt
King George V	Denmark Strait	ép
Kirishima	Guadalcanal	elsüllyedt
Prince of Wales	Denmark Strait	megsérült
Rodney	Denmark Strait	ép
Scharnhorst	North Cape	elsüllyedt
South Dakota	Guadalcanal	megsérült
Tennessee	Surigao Strait	ép
Washington	Guadalcanal	ép
West Virginia	Surigao Strait	ép
Yamashiro	Surigao Strait	elsüllyedt

(c) A Kimenetelek reláció mintaadatai

2.22. ábra. A 2.4.3. feladat adatai

név	*osztály*	*felavatva*
California	Tennessee	1921
Haruna	Kongo	1915
Hiei	Kongo	1914
Iowa	Iowa	1943
Kirishima	Kongo	1915
Kongo	Kongo	1913
Missouri	Iowa	1944
Musashi	Yamato	1942
New Jersey	Iowa	1943
North Carolina	North Carolina	1941
Ramillies	Revenge	1917
Renown	Renown	1916
Repulse	Renown	1916
Resolution	Revenge	1916
Revenge	Revenge	1916
Royal Oak	Revenge	1916
Royal Sovereign	Revenge	1916
Tennessee	Tennessee	1920
Washington	North Carolina	1941
Wisconsin	Iowa	1944
Yamato	Yamato	1941

2.23. ábra. A `Hajók` reláció mintaadatai

A 2.22. és a 2.23. ábra a fenti négy reláció mintaadatait tartalmazza.[4] Figyeljük meg, hogy a 2.4.1. feladat adataival ellentétben ezek az adatok tartalmaznak „lógó sorokat". Ilyenek például azok a hajók, amelyeket a `Kimenetelek` reláció tartalmaz, de a `Hajók` reláció nem.

Írjuk fel a következő kérdésekhez tartozó relációs algebrai kifejezéseket. A 2.4.13. alfejezetben bemutatott lineáris jelölésmód használható. A 2.22. és a 2.23. ábrán látható adatok alapján számoljuk ki a lekérdezések eredményeit. A válaszul felírt kifejezéseknek más adatokra is a helyes választ kell adniuk, nem csak az ábrán látható adatokra.

a) Adjuk meg azokat a hajóosztályokat a gyártó országok nevével együtt, amelyeknek az ágyúi legalább 16-os kaliberűek.

b) Melyek azok a hajók, amelyeket 1921 előtt avattak fel?

c) Adjuk meg a Denmark Strait-i csatában elsüllyedt hajók nevét.

[4] Forrás: J. N. Westwood, *Fighting Ships of World War II*, Follett Publishing, Chicago, 1975 és R. C. Stern, *US Battleships in Action*, Squadron/Signal Publications, Carrollton, TX, 1980.

d) Az 1921-es Washingtoni Egyezmény betiltotta a 35 000 tonnánál súlyosabb hajókat. Adjuk meg azokat a hajókat, amelyek megszegték az egyezményt.

e) Adjuk meg a guadalcanali csatában részt vett hajók nevét, vízkiszorítását és ágyúinak számát.

f) Adjuk meg az adatbázisban szereplő összes hadihajó nevét. (Ne feledjük, hogy a `Hajók` relációban nem feltétlenül szerepel az összes hajó!)

! *g)* Adjuk meg azokat az osztályokat, amelyekbe csak egyetlenegy hajó tartozik.

! *h)* Melyek azok az országok, amelyeknek csatahajóik is és cirkálóhajóik is voltak?

! *i)* Adjuk meg azokat a hajókat, amelyek „újjáéledtek", azaz egyszer már megsérültek egy csatában, de egy későbbi csatában újra harcoltak.

2.4.4. feladat. Az előző feladatban kapott kifejezések mindegyikéhez rajzoljuk fel a megfelelő kifejezésfát.

2.4.5. feladat. Adott az $R \bowtie S$ természetes összekapcsolás és az $R \bowtie_C S$ théta-összekapcsolás. A C feltétel az összes olyan A attribútumra, amely az R-ben és S-ben is szerepel, tartalmazza az $R.A = S.A$ egyenlőséget. Mi a különbség a két összekapcsolás között?

! **2.4.6. feladat.** Egy relációkon értelmezett operátor akkor *monoton*, ha bármelyik argumentumrelációhoz egy újabb sort hozzávéve az eredmény tartalmazza az összes olyan sort, amelyet addig tartalmazott és esetleg újabb sorokat is. Az ebben a fejezetben felsorolt operátorok közül melyek monotonok? Mindegyik operátorra indokoljuk, hogy miért monoton, illetve ha nem monoton, adjunk ellenpéldát.

! **2.4.7. feladat.** Tegyük fel, hogy az R relációnak n, az S relációnak pedig m sora van. Adjuk meg a következő kifejezések eredményeiben keletkezhető sorok maximális és minimális számát.

a) $R \cup S$.

b) $R \bowtie S$.

c) $\sigma_C(R) \times S$, tetszőleges C feltétel esetén.

d) $\pi_L(R) - S$, tetszőleges L attribútumlista esetén.

! **2.4.8. feladat.** Az R és S relációk *félig-összekapcsolása*, $R \ltimes S$ az R azon sorainak halmaza, amely sorok megegyeznek S legalább egy sorával R és S összes közös attribútumán. Adjunk meg olyan három különböző relációs algebrai kifejezést, amelyek ekvivalensek az $R \ltimes S$ kifejezéssel.

! **2.4.9. feladat.** Az R és S relációk *nem félig-összekapcsolása*, $R \overline{\ltimes} S$ az R azon t sorainak halmaza, amely sorok R és S összes közös attribútumain nem egyeznek meg S egyetlen sorával sem. Adjunk meg olyan három különböző relációs algebrai kifejezést, amelyek ekvivalensek az $R \overline{\ltimes} S$ kifejezéssel.

!! **2.4.10. feladat.** Az R reláció sémája

$$(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$$

és az S reláció sémája $(B_1, B_2, \ldots, B_m)$; azaz S összes attribútuma benne van R attribútumainak halmazában. Az R és S *hányadosa*, $R \div S$ megadja azon $A_1, A_2, \ldots, A_n$ attribútumú t sorok halmazát, amelyekre igaz, hogy az S reláció minden s sorára a ts sor benne van az R relációban. Írjuk fel az eddig bemutatott operátorok segítségével azt a relációs algebrai kifejezést, amely ekvivalens az $R \div S$ kifejezéssel.

2.5. Relációkra vonatkozó megszorítások

Most térjünk rá az adatmodell harmadik legfontosabb részére: az adatbázisban tárolható adatok megszorításainak a képességére. Ezeket is az adatbázisban tároljuk le. Eddig csak egyetlen típusú megszorítást láthattunk, mégpedig azt, mikor egy vagy több attribútum kulcsot alkothat (lásd 2.3.6. alfejezet). Ez utóbbi, illetve sok egyéb megszorítás is kifejezhető a relációs algebrában. Ennek az alfejezetnek a során látni fogjuk, hogyan fejezhetjük ki mind a kulcsmegszorítást, mind a „hivatkozási épség" megszorítást. Ez utóbbi elvárás az, hogy ha egy érték egy reláció egy oszlopában előfordul, akkor ez szerepelni fog a reláció vagy egy másik reláció valamelyik oszlopában is. A 7. fejezetben látni fogjuk, hogyan kényszeríti ki ezt a relációs algebrában kifejezhető megszorítást az SQL adatbázisrendszere.

2.5.1. Megszorítások megadása relációs algebra segítségével

Megszorításokat két módon fejezhetünk ki relációs algebrai kifejezésekkel:

1. Ha R egy relációs algebrai kifejezés, akkor $R = \emptyset$ egy olyan megszorítás, amelynek jelentése: „R-nek üresnek kell lennie" vagy másképpen fogalmazva „R eredményében egyetlen sor sincs".

2. Ha R és S relációs algebrai kifejezések, akkor $R \subseteq S$ egy olyan megszorítás, melynek jelentése: „R eredményének minden sora benne kell legyen S eredményében". Természetesen S eredménye tartalmazhat R sorain kívül más sorokat is.

A megszorítások megadásának ez a két módja a kifejezőerő szempontjából ekvivalens egymással, viszont bizonyos esetekben az egyik forma érthetőbb vagy

tömörebb. Az $R \subseteq S$ megszorítás $R - S = \emptyset$ alakban is felírható, hiszen ha R minden sora benne van S-ben, akkor biztos, hogy $R - S$ üres. Fordítva, ha $R - S$ egyetlen sort sem tartalmaz, akkor R minden sorának benne kell lennie S-ben, máskülönben $R - S$ eredményében lenne.

Másrészről az $R = \emptyset$ formájú megszorítások is felírhatók úgy, mint $R \subseteq \emptyset$. Technikailag a $\emptyset$ nem egy relációs algebrai kifejezés, de mivel az $R - R$ kifejezés eredménye $\emptyset$, nyugodtan használhatjuk relációs algebrai kifejezésként.

A következő alfejezetekben láthatjuk majd, hogy miként fejezhetők ki fontos megszorítások a két stílus egyikével. Amint azt a 7. fejezetben látni fogjuk, SQL-ben a megszorításokat általában az első módon, üres halmazok segítségével fejezzük ki. A megszorítást azonban nyugodtan megfogalmazhatjuk halmazok tartalmazásaként is, majd pedig az előző gondolatmenetet követve átalakíthatjuk az üres halmazt használó formára.

2.5.2. Hivatkozási épség

A megszorítások egy gyakori formája a *hivatkozási épség*, mely szerint ha egy érték megjelenik valahol egy környezetben, akkor ugyanez az érték egy másik, az előzővel összefüggő környezetben is megjelenik. Például a filmeket nyilvántartó adatbázisunkban elvárhatjuk, hogy a `SzerepelBenne` egy sorában lévő p `színészNév` komponens esetén egy olyan színésznek a nevéről beszélünk, aki benne van a `FilmSzínész` relációban is. Ellenkező esetben ugyanis feltehetnénk a kérdést, hogy az illető „színész" valóban színész-e.

Általánosságban, ha az R reláció egy sorának A attribútumában szerepel egy v érték, akkor tervezési szándékaink miatt elvárhatjuk, hogy ez a v érték egy másik S reláció valamely sorának egy bizonyos komponensében (például B-ben) is megjelenjen. A hivatkozási épséget relációs algebrában kifejezhetjük a $\pi_A(R) \subseteq \pi_B(S)$ vagy az (ezzel ekvivalens) $\pi_A(R) - \pi_B(S) = \emptyset$ kifejezések valamelyikével.

2.21. példa. Vegyük a filmekről szóló szokásos példánkból a következő két reláció sémáját:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Ésszerűnek tűnik a feltételezés, miszerint mindegyik film producere szerepel a `GyártásIrányító` relációban. Ha ez nem így van, akkor ez hiba, és szeretnénk, ha a relációs adatbázisrendszerünk legalább tájékoztatna arról, hogy van olyan film az adatbázisban, amelynek produceréről nincsenek adataink.

Hogy pontosabbak legyünk, a `Filmek` reláció mindegyik sorának `producerAzon` komponense meg kell jelenjen a `GyártásIrányító` reláció valamely sorának `azonosító` komponensében. Mivel minden egyes gyártásirányító egyedi azonosítóval rendelkezik, ezért azt kell biztosítanunk, hogy egy film producere szerepeljen a film gyártásirányítói között. Ezt a megszorítást a következő tartalmazással fejezhetjük ki:

$$\pi_{\texttt{producerAzon}}(\texttt{Filmek}) \subseteq \pi_{\texttt{azonosító}}(\texttt{GyártásIrányító})$$

A bal oldali kifejezés megadja a `Filmek` reláció sorainak `producerAzon` komponenseiben szereplő összes azonosító halmazát. Hasonló módon a jobb oldali kifejezés megadja a `GyártásIrányító` reláció sorainak azonosító komponenseiben szereplő összes azonosító halmazát. A megszorítás tehát azt mondja ki, hogy ez utóbbi halmaz tartalmazza az előző halmazt. □

2.22. példa. Egynél több attribútum értékére vonatkozó hivatkozási épséget hasonló módon fejezhetünk ki. Például biztosítani akarjuk, hogy a

```
SzerepelBenne(filmCím, filmÉv, színészNév)
```

relációban szereplő mindegyik film szerepeljen a

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

relációban. A filmeket mindkét relációban a cím és a készítési év segítségével reprezentáljuk, hiszen megegyeztünk, hogy a filmek azonosítására egyik attribútum sem lenne egyedül elegendő. A következő megszorítás

$$\pi_{\texttt{filmCím,filmÉv}}(\texttt{SzerepelBenne}) \subseteq \pi_{\texttt{filmcím,év}}(\texttt{Filmek})$$

a relációk megfelelő komponenseinek levetítésével nyert filmcím-év párok összehasonlításával fejezi ki a hivatkozási épséget. □

2.5.3. Kulcsmegszorítás

Ugyanezen megszorítási jelöléssel többet is kifejezhetünk, mint a hivatkozási épség. A most következőkben látni fogjuk, hogyan fejezhető ki algebrai úton az a megszorítás, hogy egy konkrét attribútum vagy egy attribútumhalmaz kulcsot kell alkosson a relációra nézve.

2.23. példa. Emlékezzünk, hogy a `név` a

```
FilmSzínész(név, cím, nem, születésiDátum)
```

reláció kulcsa volt. Ezért egyetlen sorpárnak sem egyezik meg a `név` komponense a másikéval. Ezt a megszorítást algebrailag egy implikációval fejezhetjük ki, azaz: ha két sor megegyezik a `név` komponensében, akkor a `cím` értékükben is meg kell egyezniük. Megjegyezve, hogy ezen „két" sorra vonatkozó állítás (azaz megegyeznek a `név` komponensben), csak akkor lehetséges, ha a két sor megegyezik, azaz minden attribútumukban azonosak.

Az elv az, hogy ha a `FilmSzínész` összes (t_1, t_2) sorpárját képezzük, akkor nem lehet olyan pár, amely a `név` komponensen megegyezik, de a `cím` komponensen eltér. Az összes pár előállítására képezzük a direkt szorzatot, és a feltételt megsértő párok meghatározására kiválasztást használunk. A megszorítás megadható úgy, hogy a kiértékelés ereménye egyezzen meg $\emptyset$-val.

Mivel a relációt önmagával szorozzuk meg, ezért legalább az egyik másolatának az attribútumneveit át kell neveznünk. A tömörség kedvéért használjuk az FSZ1 és FSZ2 új neveket a FilmSzínész relációra való hivatkozásnál. Ekkor az elvárás kifejezhető az alábbi algebrai megszorítással:

$$\sigma_{\text{FSZ1.név=FSZ2.név AND FSZ1.cím}\neq\text{FSZ2.cím}}(\text{FSZ1} \times \text{FSZ2}) = \emptyset$$

A fenti kifejezésben az FSZ1 × FSZ2 szorzatban szereplő FSZ1 a következő átnevezésnek a rövidítése:

$$\rho_{\text{FSZ1(név,cím,nem,születésiDátum)}}(\text{FilmSzínész})$$

Az FSZ2 szintén a FilmSzínész ehhez hasonló átnevezése lesz. □

2.5.4. További példák megszorításokra

Többféle megszorítás van, amelyet kifejezhetünk relációs algebrában, és amelyek hasznosak lesznek az adatbázis korlátozására nézve is. A megszorítások egy nagyobb családját bizonyos értékek tiltása alkotja. Például, hogy minden attribútumnak van egy típusmegszorítása az attribútum értékeire nézve. Gyakran elég egyértelműek ezek a megszorítások, mint például: az attribútum típusa legyen egész szám vagy 30 hosszúságú karakterlánc. Más esetekben azt szeretnénk, ha az attribútumban lévő értékek egy kisebb felsorolható halmaz értékeire korlátozódnának. Megint más esetekben viszont elvárhatjuk, hogy összetett korlátozásokat adhassunk meg az előforduló értékekre. Két egyszerű példát fogunk adni: az első egy attribútum egyszerű *tartománymegszorítás*a, a másik pedig egy bonyolultabb elvárás lesz.

2.24. példa. Tegyük fel, hogy a FilmSzínész reláció nem attribútumának megengedett értékei 'N' és 'F'. Ezt a megszorítást algebrailag a következőképpen fejezhetjük ki:

$$\sigma_{\text{nem}\neq\text{'N' AND nem}\neq\text{'F'}}(\text{FilmSzínész}) = \emptyset$$

Ez azt jeleni, hogy a FilmSzínész reláció azon sorainak halmaza, amelyekben a nem komponens értéke sem nem 'N', sem nem 'F', üres. □

2.25. példa. Tegyük fel, hogy valaki csak akkor lehet egy filmstúdió elnöke, ha a nettó bevétele legalább 10 000 000 $. Ezt a következő módon fejezhetjük ki algebrailag. Először is szükségünk van a következő két reláció théta-összekapcsolására:

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

A théta-összekapcsolás feltétele az, hogy a Stúdió reláció elnökAzon komponense egyenlő legyen a GyártásIrányító reláció azonosító komponensével. Az összekapcsolással olyan sorpárokat kapunk, amelyben a gyártásirányító egyúttal a stúdió elnöke. Ha ebből a relációból kiválasztjuk azokat a sorokat, ahol a nettó bevétel kisebb, mint tízmillió, olyan halmazt kapunk, amely a megszorításunk értelmében üres. Tehát a megszorítást a következőképpen fejezhetjük ki:

$$\sigma_{\text{nettóBevétel}<10000000}(\text{Stúdió} \bowtie_{\text{elnökAzon}=\text{azonosító}} \text{GyártásIrányító}) = \emptyset$$

Ennek a megszorításnak a kifejezésére egy másik lehetőség az, ha a stúdiók elnökeinek azonosítóiból kapott halmazt hasonlítjuk össze azon gyártásirányítók azonosítóinak halmazával, akiknek a nettó bevétele legalább 10 000 000 $. Ez előző részhalmaza kell legyen az utóbbinak. A következő tartalmazás éppen ezt fejezi ki:

$$\pi_{\text{elnökAzon}}(\text{Stúdió}) \subseteq \pi_{\text{azonosító}}(\sigma_{\text{nettóBevétel}\geq 10000000}(\text{GyártásIrányító}))$$

□

2.5.5. Feladatok

2.5.1. feladat. Írjuk fel a következő megszorításokat a 2.3.1. feladat relációira:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

A megszorításokat tartalmazással és üres halmaz segítségével is kifejezhetjük. A 2.4.1. feladat adatait használva, valamennyi megszorítás esetén mutassuk meg azokat a sorokat, amelyek megsértik az adott megszorítást.

a) Az olyan PC-ket, amelyek processzorának sebessége kisebb, mint 2.00, nem árulhatják 500 $-nál drágábban.

b) A 15,4 hüvelyknél kisebb képernyőjű laptopoknak a merevlemeze legalább 100 gigabájtos kell legyen, ellenkező esetben csak 1000 $-nál olcsóbban árulhatják.

! *c)* A PC-gyártók nem gyárthatnak laptopokat.

!! *d)* Ha egy gyártó készít PC-t, akkor készítenie kell olyan laptopot is, amelynek a sebessége legalább akkora, mint a PC sebessége.

! *e)* Ha egy laptopnak nagyobb memóriája van, mint egy PC-nek, akkor a laptop drágább kell legyen, mint a PC.

2.5.2. feladat. Fejezzük ki relációs algebrában a következő megszorításokat. A megszorítások a 2.3.2. feladat relációira vonatkoznak.

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

A megszorításokat tartalmazással és üres halmaz segítségével is kifejezhetjük. A 2.4.3. feladat adatait használva valamennyi megszorítás esetén mutassuk meg azokat a sorokat, amelyek megsértik az adott megszorítást.

a) Egyik hajóosztály ágyúinak kalibere sem lehet nagyobb, mint 16 hüvelyk.

b) Ha egy hajóosztály ágyúinak száma több mint 9, akkor a kaliberük nem lehet nagyobb 14 hüvelyknél.

! *c)* A hajóosztályok nem tartalmazhatnak kettőnél több hajót.

! *d)* Egy országnak sem lehet csatahajója és cirkálóhajója is egyszerre.

!! *e)* A több mint 9 ágyúval rendelkező hajó nem csatázhat olyan 9-nél kevesebb ágyúval rendelkező hajóval, amelyik elsüllyedt.

! **2.5.3. feladat.** Tegyük fel, hogy van két relációnk, R és S. Legyen C egy olyan hivatkozási épség, amely azt állítja, hogy ha az R relációban szerepel egy olyan sor, amelynek az $A_1, A_2, \ldots, A_n$ attribútumokon vett értéke rendre $v_1, v_2, \ldots, v_n$, akkor az S relációban szerepelnie kell egy olyan sornak, amelynek a $B_1, B_2, \ldots, B_n$ attribútumokon vett értéke rendre $v_1, v_2, \ldots, v_n$. Írjuk fel ezt a C megszorítást relációs algebrában.

! **2.5.4. feladat.** Másik algebrai módszer egy megszorítás kifejezésére az $E_1 = E_2$ alak, ahol E_1 és E_2 relációs algebrai kifejezések. Tud-e többet kifejezni a megszorításnak ez a formája, mint az ebben a fejezetben megbeszélt két társa?

2.6. Összefoglalás

- *Adatmodellek:* Az adatmodell egy jelölésmód egy adatbázis adatszerkezetének a leírására, beleértve az adatra vonatkozó megszorításokat is. Az adatmodell általában az adatokon végezhető műveletek leírására alkalmas jelölést is magába foglalja. A műveletek lehetnek lekérdezések, illetve adatmódosítások.

- *Relációs modell:* A relációk információt reprezentáló táblák. Az oszlopok fejlécében attribútumok vannak, minden attribútumhoz tartozik egy tartomány vagy adattípus. A táblasorokat soroknak hívjuk, és a reláció minden attribútumához tartozik a sornak egy komponense.

- *Sémák:* A relációnév a reláció attribútumaival együtt adja a relációsémát. A relációsémák összessége az adatbázisséma. Egy relációhoz vagy relációk összességéhez tartozó adatokat az adott relációsémához vagy adatbázissémához tartozó előfordulásnak nevezzük.

- *Kulcsok:* Egy fontos megszorítási típus a táblákra nézve, hogy ki tudunk jelölni egy attribútumot vagy attribútumhalmazt, amely egy kulcsot alkot a relációra nézve. Egy reláció semelyik két sora nem egyezhet meg a kulcsattribútumokon, de egy részhalmazukon igen.

- *Félig-strukturált adatmodell:* Ebben a modellben az adat fa- vagy gráfszerkezetbe van szervezve. Az XML a félig-strukturált adatmodellek egyik fontos példája.

- *SQL:* A relációs adatbázisrendszerek alapvető lekérdező nyelve az SQL. A jelenlegi szabványt SQL-99-nek nevezik. A forgalomban lévő rendszerek általában eltérnek a szabványtól, de nagyjából megőrzik azt.

- *Adatdefiníció:* Az SQL rendelkezik az adatbázisséma elemeit meghatározó utasításokkal. A `CREATE TABLE` utasítás lehetővé teszi egy tárolt reláció (nevezetesen tábla) sémájának megadását: az attribútumok, a hozzájuk tartozó típusok, az alapértelmezett értékek és a kulcsok meghatározását.

- *Sémamódosítás:* Az adatbázisséma részeit megváltoztathatjuk az `ALTER` utasítás segítségével. Ez a változtatás lehet a relációsémák egy attribútumának a hozzáadása vagy törlése, és lehetőséget ad egy attribútum alapértelmezett értékének megváltoztatására is. A `DROP` utasítás használható a reláció teljes sémájának vagy egyéb sémaelemnek a törlésére.

- *Relációs algebra:* Ez az algebra egy fontos lekérdező nyelv a relációs modellben. Alapvető műveletei: egyesítés, metszet, különbség, kiválasztás, vetítés, Descartes-szorzat, természetes összekapcsolás, théta-összekapcsolás és átnevezés.

- *Vetítés és kiválasztás:* A kiválasztás művelet egy olyan eredményt ad vissza, amely az argumentumában szereplő relációnak az összes olyan sorát tartalmazza, amely megfelel a kiválasztási feltételnek. A vetítés az argumentumában szereplő reláció nem kívánt oszlopait eltávolítja az eredményrelációból.

- *Összekapcsolások:* Két relációt kapcsol össze egy relációba a relációkban szereplő sorok összehasonlításával. A természetes összekapcsolásnál azon sorokat rakjuk össze, amelyek az összes közös attribútumon megegyeznek a két relációban. A théta-összekapcsolásnál azon sorokat fűzzük egybe, amelyek a théta-összekapcsolás kiválasztási feltételének eleget tesznek.

- *Megszorítások a relációs algebrában:* Többféle megszorítás is kifejezhető két relációs algebrai kifejezés közötti tartalmazásként vagy egy relációs algebrai kifejezés és az üres halmaz közötti egyenlőség segítségével.

2.7. Irodalomjegyzék

Codd klasszikusnak nevezhető tanulmánya a relációs modellről [1], amelyben bevezeti a relációs algebrát is. A megszorítások leírását a relációs algebra segítségével [2] tárgyalja. Az SQL irodalmi hivatkozásai a 6. fejezet irodalomjegyzékében találhatók.

[3] a félig-strukturált adatmodellt mutatja be. Az XML-szabványt a World-Wide-Web Consortium fejleszti. A hivatalos XML-ről szóló honlap címe [4].

[1] E. F. Codd, „A relational model for large shared data banks," *Comm. ACM* **13**:6, pp. 377–387, 1970.

[2] J.-M. Nicolas, „Logic for improving integrity checking in relational databases," *Acta Informatica* **18**:3, pp. 227–253, 1982.

[3] Y. Papakonstantinou, H. Garcia-Molina, J. Widom, „Object exchange across heterogeneous information sources," *IEEE Intl. Conf. on Data Engineering*, pp. 251–260, March 1995.

[4] World-Wide-Web Consortium, `http://www.w3.org/XML/`

3. fejezet

Relációs adatbázisok tervezésének elmélete

Egy alkalmazáshoz kapcsolódó relációs adatbázisséma megtervezésekor nagyon sokféle módon járhatunk el. A 4. fejezetben számos magas szintű adatszerkezetet leíró módszerrel ismerkedhetünk meg, illetve azokkal a módszerekkel, amelyekkel ezek a magas szintű leírások sémákká alakíthatók. Továbbá tanulmányozzuk az adatbázisokra vonatkozó követelményeket és megvizsgáljuk, hogyan definiálhatunk relációkat a magas szintű köztes lépés kihagyásával. Bármilyen megközelítést is használunk, leggyakrabban egy kiinduló sémát készítünk, amelyet javítani kell, például a redundancia kiküszöbölésével. Gyakran az adatbázissémákkal kapcsolatos problémákat az okozza, hogy túl sok mindent szeretnénk egyetlen sémába összekombinálni.

Szerencsére az adatbázisok tervezésének elméleti háttere jól kidolgozott: a „függőségek" és azok következményei segítségével jó sémákat állíthatunk elő, illetve segítenek megoldani azokat a helyzeteket, amikor egy séma hibáit kell kijavítani. Ebben a fejezetben először azonosítjuk azokat a helyzeteket, amelyekben bizonyos függőségek jelenléte a sémában problémát okoz. Ezeket a problémákat „anomáliáknak" nevezzük.

A részletes tárgyalást a „funkcionális függőségek" vizsgálatával kezdjük, amely a relációk kulcsának általánosítása. Ezután a funkcionális függőség fogalmát felhasználva normálformákat állítunk elő az adatbázisokhoz. Ennek az elméletnek hatása a „normalizáció", melynek során egyes relációkat két vagy több részre vágunk, így megszüntetve az anomáliákat. Ezután bemutatjuk a „többértékű függőségeket", amelyek azt fogalmazzák meg, hogy egy reláció egy vagy több attribútuma független egy vagy több másik attribútumtól. Ezek a függőségek szintén normálformákhoz, illetve relációk felbontásához vezetnek a redundancia kiküszöbölése érdekében.

3.1. Funkcionális függőségek

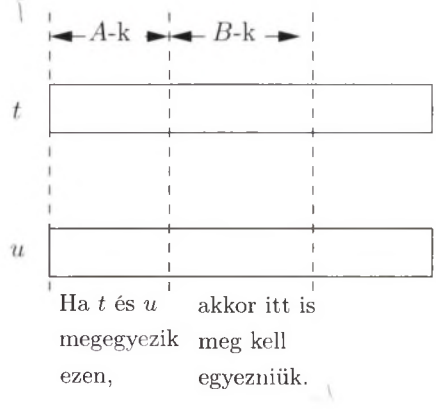
A funkcionális függőségek elmélete egy olyan tervezési elvet szolgáltat, amellyel alaposan tanulmányozhatjuk a relációkat, és néhány fontos alapelv segítségével javíthatjuk őket. Első lépésként megállapítjuk azokat a megszorításokat, amelyeket alkalmazni kell a relációra. A leggyakoribb megszorítás a „funkcionális függőség", amely a 2.5.3. alfejezetben tárgyalt kulcsfogalom általánosításával fogalmaz meg állításokat. A fejezet későbbi részében látni fogjuk, hogy ez az elmélet egy egyszerű eszközt ad a relációk „dekompozíciójának" tökéletesítésére: egy relációt helyettesítünk több másikkal, melyek összes attribútumának halmaza tartalmazza az eredeti reláció összes attribútumát.

3.1.1. A funkcionális függőség definíciója

A *funkcionális függőség* (FD[1]) egy R reláción a következő formájú állítás: „Ha R két sora megegyezik az $A_1, A_2, \ldots, A_n$ attribútumokon (azaz ezen attribútumok mindegyikéhez megfeleltetett komponensnek ugyanaz az értéke a két sorban), akkor meg kell egyezniük más attribútumok egy $B_1, B_2, \ldots, B_m$ sorozatán." Ezt a függőséget formálisan $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$-mel jelöljük, és azt mondjuk, hogy

„$A_1, A_2, \ldots, A_n$ funkcionálisan meghatározza $B_1, B_2, \ldots, B_m$-et".

A 3.1. ábrán láthatjuk, mit jelent a funkcionális függőség az R reláció bármely két, t és u sorára. Ugyanakkor az A-k és a B-k bárhol elhelyezkedhetnek, nem szükséges hogy az A-k és a B-k egymást kövessék, vagy hogy az A-k megelőzzék a B jelűeket.



3.1. ábra. A funkcionális függőség két soron vett hatása

[1] FD – Functional Dependency, a funkcionális függőség angol nyelvű megfelelőjének rövidítése. *(A lektor megjegyzése.)*

Ha biztosak lehetünk abban, hogy az R reláció minden előfordulása olyan, amelyen az adott FD igaz, akkor azt mondhatjuk, hogy R *kielégíti* FD-t. Fontos megjegyezni, hogy amikor azt állítjuk, hogy R kielégíti az f FD-t, akkor ez egy megszorítást jelent R-re, nem csupán egy állítást R valamely egyedi előfordulásáról.

Gyakori, hogy a függőség jobb oldala egyetlen attribútum. Valójában az $A_1A_2\cdots A_n \rightarrow B_1B_2\cdots B_m$ függőségre úgy fogunk tekinteni, mint FD-k egy halmazára:

$$\begin{aligned} &A_1A_2\cdots A_n \rightarrow B_1\\ &A_1A_2\cdots A_n \rightarrow B_2\\ &\cdots\\ &A_1A_2\cdots A_n \rightarrow B_m \end{aligned}$$

filmcím	*év*	*hossz*	*műfaj*	*stúdióNév*	*színészNév*
Csillagok háborúja	1977	124	sci-fi	Fox	Carrie Fisher
Csillagok háborúja	1977	124	sci-fi	Fox	Mark Hamill
Csillagok háborúja	1977	124	sci-fi	Fox	Harrison Ford
Elfújta a szél	1939	231	dráma	MGM	Vivien Leigh
Wayne világa	1992	95	vígjáték	Paramount	Dana Carvey
Wayne világa	1992	95	vígjáték	Paramount	Mike Meyers

3.2. ábra. A `Filmek1(filmcím, év, hossz, műfaj, stúdióNév, színészNév)` reláció egy előfordulása

3.1. példa. Tekintsük a

`Filmek1(filmcím, év, hossz, műfaj, stúdióNév, színészNév)`

relációt és ennek a 3.2. ábrán található előfordulását. Kapcsolódunk a filmekről szóló példánkhoz, de ez a reláció tartalmaz további attribútumokat, így „`Filmek`" helyett „`Filmek1`"-nek nevezzük. Megjegyezzük, hogy ez a reláció túl sok mindent akar egyszerre magába foglalni. Megpróbálja egyesíteni mindazt, amit az eddigi példákban három külön relációban tároltunk: `Filmek`, `Stúdió` és `SzerepelBenne`. Ahogy látni fogjuk, a `Filmek1` reláció sémája nincs jól megtervezve. Ahhoz, hogy lássuk, miért rossz ez a terv, meg kell határoznunk a relációra érvényes funkcionális függőségi megszorításokat. Azt állítjuk, hogy a

`filmcím év` → `hossz műfaj stúdióNév`

FD fennáll.

Informálisan ez azt jelenti, hogy ha két sorban a `filmcím` mezőben ugyanaz az érték szerepel, és szintén megegyeznek az `év` attribútumon, akkor ezekben a sorokban azonos értéknek kell szerepelnie a `hossz`, `műfaj` és `stúdióNév` komponensekben is. Ezt az állítást úgy kell érteni, hogy nincs két olyan film, amely

ugyanabban az évben azonos címmel jelent meg (ugyanakkor lehetséges, hogy vannak azonos című filmek, amelyek különböző években jelentek meg). Erről a 2.1. példában volt szó. Tehát azt várjuk, hogy a filmcím és az év megadása egy egyedi filmet határoz meg. Azaz a filmnek egyedi hossza, műfaja és stúdiója van.

Másrészt vegyük észre, hogy a

$$\texttt{filmcím év} \rightarrow \texttt{színészNév}$$

állítás hamis, ez a függőség nem érvényes a filmekre. Egy filmnek ugyanis több szereplője is lehet, így egy adott filmhez több színész is szerepelhet az adatbázisban. Jegyezzük meg, hogy ha lusták lettünk volna felsorolni több szereplőt a *Csillagok háborújá*hoz és a *Wayne világá*hoz (ahogyan azt tettük az *Elfújta a szél* esetében), ez a függőség akkor sem lenne azonnal igaz a `Filmek1` relációra. Ennek az az oka, hogy az FD a lehetséges előfordulásokról fogalmaz meg állítást, nem pedig az aktuális előfordulásról. Az a tény, hogy *lehetne* olyan előfordulásunk, amelyben több színész tartozhatna egy mozifilmhez, kizárja azt, hogy a `filmcím` és az `év` meghatározza a `színészNév` értékét. □

3.1.2. Relációk kulcsai

Azt mondjuk, hogy az egy vagy több attribútumból álló $\{A_1, A_2, \ldots, A_n\}$ halmaz az R *kulcs*a, ha:

1. Ezek az attribútumok funkcionálisan meghatározzák a reláció minden más attribútumát, azaz nem lehet R-ben két olyan különböző sor, amely mindegyik $A_1, A_2, \ldots, A_n$-nen megegyezne.

2. Nincs olyan valódi részhalmaza $\{A_1, A_2, \ldots, A_n\}$-nek, amely funkcionálisan meghatározná R összes többi attribútumát, azaz a kulcsnak *minimális*nak kell lennie.

Amikor csak egyetlenegy A attribútumból áll a kulcs, akkor gyakran azt mondjuk, hogy A (ahelyett hogy $\{A\}$) kulcs.

3.2. példa. A {`filmcím`, `év`, `színészNév`} attribútumok a 3.2. ábrán szereplő `Filmek1` reláció egy kulcsát alkotják. Először azt kell megmutatnunk, hogy ezek az attribútumok funkcionálisan meghatározzák az összes többi attribútumot. Azaz vegyünk két sort, amelyek megegyeznek a három attribútumon: `filmcím`, `év` és `színészNév`. Mivel ezek megegyeznek a `filmcím`-en és `év`-en, ezért meg kell egyezniük a `hossz`, `műfaj` és a `stúdióNév` attribútumokon, mint korábban, a 3.1. példában már láttuk. Tehát nincs két olyan különböző sor, amely mindhárom – `filmcím`, `év` és `színészNév` – attribútumon megegyezne, hiszen ez tulajdonképpen ugyanazt a sort jelentené.

Most azt kell megmutatnunk, hogy {`filmcím`, `év`, `színészNév`} attribútumhalmaznak nincs olyan valódi részhalmaza, amely funkcionálisan meghatározná az összes többi attribútumot. Ahhoz, hogy ezt belássuk, először figyeljük meg,

Mitől „funkcionális" a funkcionális függőség?

$A_1A_2\cdots A_n \rightarrow B$-t „funkcionális" függőségnek nevezzük, mivel elméletben van egy olyan függvény (funkció), ami olyan értéklistán van értelmezve, amely az $A_1, A_2, \ldots, A_n$ minden egyes attribútumához hozzárendel egy-egy értéket, és előállítja a B egyértelmű értékét (vagy egyáltalán semmilyen értéket). Például a `Filmek1` relációhoz elképzelhetünk egy olyan függvényt, amely a `"Csillagok háborúja"` sztringhez és az 1977 egészhez rendeli hozzá a `hossz` egyértelmű értékét, nevezetesen a 124-et, ami a `Filmek1` relációban megtalálható. Habár ez a függvény, mint leképezés, általános értelemben vett függvény, mégsem olyan, mint amivel a matematikában gyakran találkozunk, mivel nem tudjuk hogyan kiszámolni az értékét az argumentumértékekből. Azaz nincs olyan művelet, amelyet ha a `"Csillagok háborúja"` sztringen és az 1977 egészen végrehajtanánk, eredményül a helyes filmhosszt adná vissza. A függvény kiszámolása inkább csupán a relációból való visszakeresést jelenti. Megkeressük az adott `filmcím`-hez és `év`-hez tartozó sort és megnézzük, hogy milyen érték található a `hossz`-ra abban a sorban.

hogy a `filmcím` és `év` nem határozza meg a `színészNév` attribútumot, mivel több filmben is szerepel egynél több színész. Tehát a {`filmcím`, `év`} nem kulcs.

Az {`év`, `színészNév`} szintén nem kulcs, mivel egy színész két filmben is játszhat ugyanabban az évben, és így

$$\texttt{év színészNév} \rightarrow \texttt{filmcím}$$

nem teljesülő funkcionális függőség. Végül azt állítjuk, hogy {`filmcím`, `színészNév`} sem kulcs, hiszen két olyan filmnek, amelyeknek ugyanaz a címük, de más évben készültek, lehetséges, hogy van közös szereplője.[2] □

Előfordulhat, hogy a relációnak egynél több kulcsa van. Ha így van, akkor általában kijelöljük az egyik kulcsot mint *elsődleges kulcs*ot. A kereskedelmi adatbázisrendszereknél az elsődleges kulcs megválasztása az implementációt is befolyásolhatja, például azt, hogyan tároljuk a relációt a lemezen. Ugyanakkor a funkcionális függőségek elmélete nem kezeli speciálisan az „elsődleges kulcsokat".

[2] Egy korábbi könyvben erre nem tudtunk valós példát hozni, de azóta többen is megmutatták, hogy van ilyen. Érdekes kihívás olyan színészeket keresni, akik ugyanannak a filmnek kétféle változatában is szerepeltek.

Kulcsok eltérő terminológiája

Egyes cikkekben és könyvekben más szóhasználattal találkozhatunk a kulcsokra vonatkozóan. A „kulcs" (key) fogalmat ugyanis tágabb értelemben, a mi fogalmaink szerinti „szuperkulcsra" használják, azaz olyan attribútumhalmazt jelölnek vele, melytől funkcionálisan függ a séma összes attribútuma, és ez nem szükségszerűen minimális. Ebben az esetben a „minimális kulcs" (vagy „kulcsjelölt", azaz „candidate key") felel meg az általunk definiált „kulcsnak".

3.1.3. Szuperkulcsok

Azokat az attribútumhalmazokat, amelyek tartalmaznak kulcsot, *szuperkulcsok*nak nevezzük, ez a rövidítése a „kulcsnál bővebb halmazoknak" (kulcsok szuperhalmazának). Tehát minden kulcs egyben szuperkulcs is. Vannak azonban olyan szuperkulcsok, amelyek nem (minimális) kulcsok. Megjegyezzük, hogy minden szuperkulcs eleget tesz a kulcsdefiníció első feltételének: funkcionálisan meghatározzák a reláció összes többi attribútumát. A szuperkulcs azonban nem feltétlenül tesz eleget a második feltételnek, a minimalitásnak.

3.3. példa. A 3.2. példa relációjában több szuperkulcs van. Nemcsak a

$$\{\texttt{filmcím}, \texttt{év}, \texttt{színészNév}\}$$

kulcs szuperkulcs, hanem ennek az attribútumhalmaznak bármely hatványhalmaza is az, mint például

$$\{\texttt{filmcím}, \texttt{év}, \texttt{színészNév}, \texttt{hossz}, \texttt{stúdióNév}\}$$

is szuperkulcs. □

3.1.4. Feladatok

3.1.1. feladat. Tekintsünk egy relációt az Egyesült Államokban élő emberekről, amely tartalmazza a nevüket, a társadalombiztosítási számukat, a lakcím utcáját, városát, államát, irányítószámát, területi kódját és egy (7 számjegyű) telefonszámot. Milyen funkcionális függőségekről várhatjuk, hogy érvényesek? Melyek a reláció kulcsai? Ahhoz, hogy megválaszoljuk ezeket a kérdéseket, tudnunk kell, hogyan történik ezeknek a számoknak a megadása. Például egy területi kód kiterjedhet-e két államra? Az irányítószám kiterjedhet-e két területi kódra? Lehet-e két embernek ugyanaz a társadalombiztosítási száma? Lehet-e ugyanaz a lakcímük vagy telefonszámuk?

3.1.2. feladat. Tekintsünk egy zárt konténerben található molekulák jelenlegi helyzetét leíró relációt. Az attribútumok a molekulaazonosító, a molekulák x, y és z koordinátái és a sebességek az x, y és z irányokban. Milyen funkcionális függőségekre várhatjuk, hogy érvényesek? Melyek a kulcsok?

!! **3.1.3. feladat.** Legyen R reláció, és $A_1, A_2, \ldots, A_n$ az attribútumai. Adjuk meg n függvényeként, hány szuperkulcsa van R-nek, ha:

a) csak A_1 kulcs;

b) csak A_1 és A_2 kulcsok;

c) csak $\{A_1, A_2\}$ és $\{A_3, A_4\}$ kulcsok;

d) csak $\{A_1, A_2\}$ és $\{A_1, A_3\}$ kulcsok.

3.2. Funkcionális függőségekre vonatkozó szabályok

Ebben a fejezetben azt tanuljuk meg, milyen *levezetési szabályok* érvényesek a funkcionális függőségekre. Tegyük fel, tudjuk, hogy a reláció milyen függőségi halmazt elégít ki. Ebből gyakran vezethetünk le további függőségeket, melyeket a relációnak szintén ki kell elégítenie. Az a képesség, hogy fel tudjuk tárni a további függőségeket, a jó relációsémák tervezésénél válik majd jelentőssé a 3.3. alfejezetben.

3.2.1. Funkcionális függőségek levezetése

Kezdjük egy példával, amelyben megmutatjuk, hogyan lehet szabályokat levezetni más adott FD-k felhasználásával.

3.4. példa. Ha azt tudjuk, hogy az $R(A, B, C)$ reláció eleget tesz az $A \to B$ és a $B \to C$ funkcionális függőségeknek, akkor ebből levezethető, hogy R megfelel az $A \to C$ függőségnek is. Nézzük az érvelést! Ahhoz, hogy bebizonyítsuk $A \to C$ érvényességét, meg kell vizsgálnunk R bármely két olyan sorát, amelyek megegyeznek A-n, és be kell bizonyítanunk, hogy ezek megegyeznek C-n is.

Legyen az A attribútumon megegyező két sor (a, b_1, c_1) és (a, b_2, c_2). Feltesszük, hogy a sorokban az attribútumok sorrendje A, B, C. Mivel R eleget tesz $A \to B$-nek, és ezek a sorok megegyeznek A-n, így meg kell egyezniük B-n is. Azaz $b_1 = b_2$, és a sorok valójában (a, b, c_1) és (a, b, c_2), ahol b b_1-et és b_2-t is jelenti egyben. Hasonlóan: mivel R eleget tesz $B \to C$-nek, és a sorok megegyeznek B-n, így megegyeznek C-n is. Azaz $c_1 = c_2$, tehát a sorok tényleg megegyeznek C-n, és így teljesül az $A \to C$ funkcionális függőség.[3] □

[3] A bizonyításból igazolódott az is, hogy az A, B és C attribútumokból álló R relációban A kulcs, tehát nem is lehet a két függőséget kielégítő relációban két olyan sor, amelyik A-ban megegyezik. A bizonyítás valójában indirekt volt. *(A fordítói megjegyzése.)*

A funkcionális függőségeket gyakran többféleképpen is megadhatjuk anélkül, hogy változna a reláció érvényes előfordulásainak a halmaza. Ilyenkor azt mondjuk, hogy:

- Két funkcionális függőségi halmaz, S és T *ekvivalensek*, ha S-et pontosan ugyanazok a reláció-előfordulások elégítik ki, mint amelyek T-t.
- Általánosabban azt mondjuk, hogy az S funkcionális függőségi halmaz *következik* a T funkcionális függőségi halmazból, ha minden olyan reláció-előfordulás, amely eleget tesz az összes T-beli függőségnek, eleget tesz minden S-beli függőségnek is.

A fentiek értelmében az S és T függőségi halmazok ekvivalensek, ha S következik T-ből, és T következik S-ből.

Ebben a fejezetben a funkcionális függőségekre vonatkozó több hasznos szabályt láthatunk. Általában ezekkel a szabályokkal valamely függőségi halmazt tudunk helyettesíteni egy, vele ekvivalens halmazzal, vagy kibővíteni az eredeti halmazból következő függőségekből álló halmazzal. Erre egy példa a *tranzitív szabály*, amellyel funkcionális függőségek láncolatait tudjuk követni, mint azt korábban a 3.4. példában láttuk. Megadunk egy algoritmust is, amelynek segítségével válaszolhatunk arra az általános kérdésre, hogy egy funkcionális függőség következik-e egy vagy több függőségből.

3.2.2. A szétvághatósági és összevonhatósági szabály

Idézzük fel, hogy az alábbi, a 3.1.1. alfejezetben már látott funkcionális függőség:

$$A_1A_2\cdots A_n \to B_1B_2\cdots B_m$$

ekvivalens az következő függőséghalmazzal:

$$A_1A_2\cdots A_n \to B_1,\quad A_1A_2\cdots A_n \to B_2,\ldots,A_1A_2\cdots A_n \to B_m$$

Azaz szétvághatjuk a jobb oldalon szereplő attribútumokat úgy, hogy csak egy attribútum legyen mindegyik funkcionális függőség jobb oldalán. Hasonlóan tudjuk helyettesíteni azonos bal oldalú függőségek készletét egyetlen függőséggel, amelynek a bal oldala ugyanaz, és a jobb oldala pedig az összes jobb oldali attribútum egy attribútumhalmazzá való összevonása. Mindkét esetben az új függőségi halmaz ekvivalens a régivel. A fenti ekvivalencia kétféleképpen alkalmazható.

- Az $A_1A_2\cdots A_n \to B_1B_2\cdots B_m$ függőséget helyettesíthetjük az $A_1A_2\cdots A_n \to B_i$ $(i = 1,2,\ldots,m)$ funkcionális függőségekből álló halmazzal. Ezt az átalakítást *szétvághatósági szabály*nak nevezzük.
- Az $A_1A_2\cdots A_n \to B_i$ $(i = 1,2,\ldots,m)$ funkcionális függőségekből álló halmazt helyettesíthetjük az $A_1A_2\cdots A_n \to B_1B_2\cdots B_m$ egyetlen függőséggel. Ezt az átalakítást *összevonhatósági szabály*nak nevezzük.

3.5. példa. A 3.1. példában már látott függőséghalmaz:

```
filmcím év → hossz
filmcím év → műfaj
filmcím év → stúdióNév
```

ekvivalens az alábbi egyetlen függőséggel:

```
filmcím év → hossz műfaj stúdióNév
```

ahogyan azt már akkor is állítottuk. □

A szétvághatósági és összevonhatósági szabályok bizonyítása nyilvánvalónak tűnik. Tegyük fel, hogy van két sorunk, melyek az $A_1, A_2, \ldots, A_n$ attribútumokon megegyeznek. Egyetlen szabályban azt mondhatnánk, hogy „ezeknek a soroknak meg kell egyezniük a $B_1, B_2, \ldots, B_m$ attribútumokon". Ehelyett különálló függőségekként azt állítjuk, hogy „ezek a sorok megegyeznek B_1-en, és megegyeznek B_2-n, ..., és megegyeznek B_m-en." Ez a két megfogalmazás pontosan ugyanazt jelenti.

Azt gondolhatnánk, hogy a szétvághatóság a funkcionális függőségek bal oldalára ugyanúgy alkalmazható, mint a jobb oldalakra, azonban ez téves, ugyanis nincs bal oldalakra vonatkozó szétvághatósági szabály, ahogyan azt a következő példában is láthatjuk:

3.6. példa. Tekintsük az alábbi függőséget, amely a 3.1. példa `Filmek1` relációjára érvényes:

```
filmcím év → hossz
```

Ha megpróbálnánk a bal oldalt szétvágni:

```
filmcím → hossz
év → hossz
```

akkor hibás függőségeket kapnánk. Ugyanis a `filmcím` nem határozza meg funkcionálisan a `hossz`-t, mivel lehet két azonos című film (például *King Kong*), amelyeknek a hossza különbözik. Hasonlóan az `év` sem határozza meg funkcionálisan a `hossz`-t, hiszen biztosan több különböző hosszúságú film készült ugyanabban az évben. □

3.2.3. Triviális funkcionális függőségek

Egy megszorításra azt mondjuk, hogy *triviális*, ha fennáll a reláció minden előfordulására attól függetlenül, hogy milyen más megszorításokat fogalmazunk meg. Ha a megszorítások funkcionális függőségek, könnyen meghatározhatjuk, hogy egy FD triviális. Ezek azok az $A_1A_2\cdots A_n \to B_1B_2\cdots B_m$ függőségek, melyekre

$$\{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\}$$

Azaz triviális függőségről beszélhetünk, ha a jobb oldal a bal oldal részhalmaza. Például a

$$\texttt{filmcím év} \rightarrow \texttt{filmcím}$$

triviális függőség csakúgy, mint a

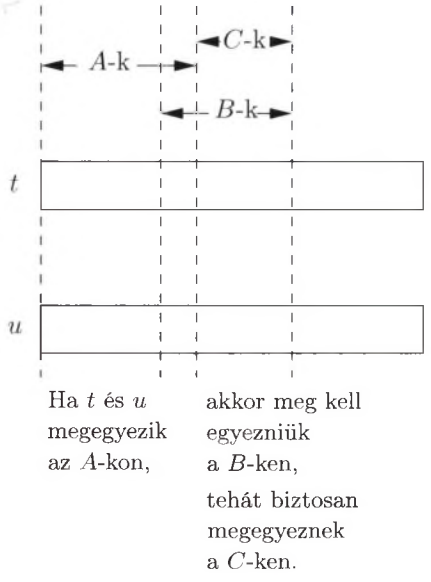
$$\texttt{filmcím} \rightarrow \texttt{filmcím}$$

Minden olyan triviális függőség minden adatbázisban igaz, amely azt mondja, hogy „ha két sor megegyezik az $A_1, A_2, \ldots, A_n$ attribútumok mindegyikén, akkor ezek egy részhalmazán is megegyeznek".

Abban az esetben, amikor néhány jobb oldali attribútum szerepel a függőség bal oldalán, de nem az összes, akkor ez a függőség nem triviális. De egyszerűsíthető úgy, hogy a jobb oldalról eltávolítjuk a bal oldalon előforduló attribútumokat. Azaz:

- $A_1A_2\cdots A_n \rightarrow B_1B_2\cdots B_m$ ekvivalens az $A_1A_2\cdots A_n \rightarrow C_1C_2\cdots C_k$ függőséggel, ahol C-kkel jelöltük azokat a B-ket, amelyek nem fordulnak elő az A-k között.

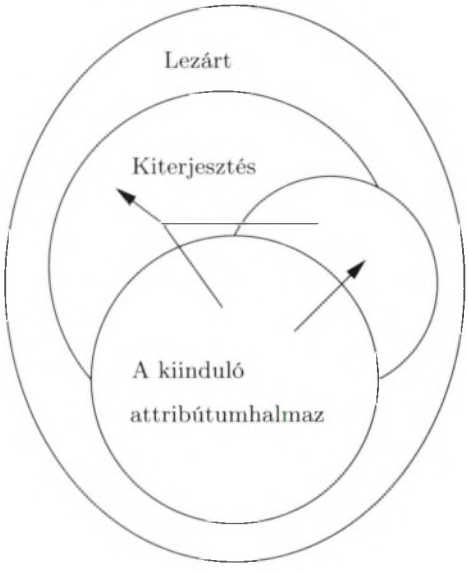
Ezt a szabályt, amelyet a 3.3. ábrán is láthatunk, *triviális függőségi szabály*nak nevezzük.



3.3. ábra. A triviális függőségi szabály

3.2.4. Attribútumhalmazok lezártjának kiszámítása

Mielőtt a többi szabállyal folytatnánk, megadunk egy általános elvet, amelyből az összes érvényes szabály következik. Legyen $\{A_1, A_2, \ldots, A_n\}$ egy attribútumhalmaz, S pedig funkcionális függőségeknek egy halmaza. Az $\{A_1, A_2, \ldots, A_n\}$ halmaz S-beli függőségek szerint vett *lezártja* azoknak a B attribútumoknak a halmaza, amelyekre minden olyan reláció, amely eleget tesz az összes S-beli függőségnek, eleget tesz $A_1A_2 \cdots A_n \rightarrow B$-nek is. Azaz $A_1A_2 \cdots A_n \rightarrow B$ az S-beli függőségekből következik. Jelöljük az $A_1A_2 \cdots A_n$ attribútumhalmaz lezártját $\{A_1, A_2, \ldots, A_n\}^+$-szal. Ahhoz, hogy egyszerűsítsük a lezártak kiszámolásának a tárgyalását, megengedjük a triviális függőségeket, így $A_1, A_2, \ldots, A_n$ mindig benne van $\{A_1, A_2, \ldots, A_n\}^+$-ban.



3.4. ábra. Egy attribútumhalmaz lezárásának kiszámítása

A 3.4. ábra szemléletesen mutatja a lezárás folyamatát. Kiindulunk az adott attribútumhalmazból, és többször ismételten növeljük ezt a halmazt azoknak a funkcionális függőségeknek a jobb oldali attribútumaival, amely függőségeknek a bal oldalát már tartalmazza az attribútumhalmaz. Nyilvánvalóan eljutunk egy pontig, amikor a halmaz már nem bővíthető tovább. Ez az eredményhalmaz lesz a lezárt.

3.7. algoritmus. Attribútumhalmaz lezártja

BEMENET: Egy $\{A_1, A_2, \ldots, A_n\}$ attribútumhalmaz és funkcionális függőségek egy halmaza.

KIMENET: Az $\{A_1, A_2, \ldots, A_n\}^+$ lezárt.

1. Ha szükséges, vágjuk szét a függőségeket úgy, hogy a jobb oldalukon egyetlen attribútum maradjon.

2. Legyen X az az attribútumhalmaz, amely végül a lezárt lesz. Inicializáláskor legyen $\{A_1, A_2, \ldots, A_n\}$.

3. Ismételten keresünk olyan $B_1B_2 \cdots B_m \to C$ funkcionális függőséget S-ből, amelyre minden $B_1, B_2, \ldots, B_m$ benne van az X attribútumhalmazban, de a C nincs. Ekkor C-t hozzávesszük az X halmazhoz. Mivel X minden lépésben csak nőhet, és minden reláció attribútumainak száma véges, tehát lesz olyan lépés, hogy X-hez már nem lehet elemet hozzávenni. Ekkor az eljárás befejeződik.

4. Az az X halmaz lesz $\{A_1, A_2, \ldots, A_n\}^+$-nak a helyes értéke, amelyet már nem tudunk tovább bővíteni.

□

3.8. példa. Tekintsünk egy relációt, amelynek attribútumai A, B, C, D, E és F. Legyenek ehhez a relációhoz tartozó funkcionális függőségek $AB \to C$, $BC \to AD$, $D \to E$ és $CF \to B$. Mi az $\{A, B\}$ lezártja, azaz $\{A, B\}^+$?

Először vágjuk szét a $BC \to AD$ függőséget $BC \to A$ és $BC \to D$ függőségekre. Induljunk ki $X = \{A, B\}$-ből. Figyeljük meg, hogy az $AB \to C$ funkcionális függőség bal oldalán szereplő mindkét attribútum benne van az X-ben, így hozzávehetjük a függőség jobb oldali attribútumát, azaz C-t X-hez. Tehát a 3. lépés első ciklusában X egyenlő lesz $\{A, B, C\}$-vel.

Ezután látjuk, hogy $BC \to A$ és $BC \to D$ bal oldalai benne vannak az X-ben, így hozzávehetjük az A és D attribútumokat X-hez. A már benne van, D nincs, ezzel bővítjük X-et, így az egyenlő lesz $\{A, B, C, D\}$-vel. Most a $D \to E$ függőséget használva E-vel bővítjük X-et, így az $\{A, B, C, D, E\}$-re bővül. Ezután X-en már nem tudunk tovább változtatni. Ebben a példában a $CF \to B$ funkcionális függőséget nem vettük figyelembe, mivel X soha nem tartalmazta a függőség bal oldalát. Tehát $\{A, B\}^+ = \{A, B, C, D, E\}$. □

Attribútumhalmazok lezártjának előállításával el tudjuk dönteni, hogy egy $A_1A_2 \cdots A_n \to B$ függőség következik-e egy funkcionális függőségeket tartalmazó S halmazból. Először számoljuk ki az $\{A_1, A_2, \ldots, A_n\}$ halmaz lezártját az S-beli szabályok felhasználásával. Ha B benne van $\{A_1, A_2, \ldots, A_n\}^+$-ban, akkor $A_1A_2 \cdots A_n \to B$ következik S-ből, ha B nincs benne $\{A_1, A_2, \ldots, A_n\}^+$-ban, akkor ez az FD nem következik S-ből. Általánosabban: $A_1A_2 \cdots A_n \to B_1B_2 \cdots B_m$ következik egy S szabályhalmazból, akkor és csak akkor, ha $B_1, B_2, \ldots, B_m$ benne van az $\{A_1, A_2, \ldots, A_n\}^+$ halmazban.

3.9. példa. Vegyük a 3.8. példában szereplő relációt és funkcionális függőségeket. Tegyük fel, hogy ellenőrizni szeretnénk, vajon az $AB \to D$ következik-e ezekből a függőségekből. Számoljuk ki $\{A, B\}^+$-t, amely $\{A, B, C, D, E\}$, ahogyan az előző példában láttuk. Mivel D eleme a lezártnak, ezért azt kapjuk,

hogy $AB \rightarrow D$ következik az adott függőségekből.

Másrészt vegyük a $D \rightarrow A$ funkcionális függőséget. Ahhoz, hogy ellenőrizzük, vajon ez a függőség következik-e az adott függőségekből, először számoljuk ki $\{D\}^+$-t. Ehhez kezdjük $X = \{D\}$-vel. Felhasználhatjuk a $D \rightarrow E$ függőséget, hogy az X halmazt bővítsük E-vel. Ezután leragadtunk. Nem tudunk találni más függőséget, amelynek a bal oldalát X tartalmazná, így $\{D\}^+ = \{D, E\}$. Mivel A nem eleme $\{D, E\}$-nek, azt kapjuk, hogy $D \rightarrow A$ nem következik. □

3.2.5. Miért működik a lezárási algoritmus?

Ebben az alfejezetben megmutatjuk, hogy a 3.7. algoritmus miért tudja helyesen eldönteni, hogy egy $A_1A_2 \cdots A_n \rightarrow B$ funkcionális függőség következik-e avagy sem az adott S funkcionális függőségi halmazból. A bizonyítás két részből áll:

1. Bizonyítanunk kell, hogy a 3.7. algoritmus nem követel túl sokat, azaz meg kell mutatnunk, hogy ha $A_1A_2 \cdots A_n \rightarrow B$ a lezárási próba alapján bekerült (vagyis B benne van $\{A_1, A_2, \ldots, A_n\}^+$-ban), akkor $A_1A_2 \cdots A_n \rightarrow B$ fennáll bármilyen olyan reláció esetén, amely S összes funkcionális függőségét kielégíti.

2. Be kell látnunk, hogy a lezárási algoritmus nem fog kihagyni egyetlen funkcionális függőséget sem, ami tényleg következne az S funkcionális függőségi halmazból.

Miért csak az igaz funkcionális függőségeket fogadja el a lezárási algoritmus?

Indukcióval bizonyíthatjuk be – aszerint, hogy az X-ben szereplő egyes D attribútumokra a harmadik lépés bővítési műveletét hányszor kellett alkalmazzuk –, hogy az $A_1A_2 \cdots A_n \rightarrow D$ funkcionális függőség teljesül. Eszerint minden olyan R reláció, amely eleget tesz az összes S-beli függőségnek, eleget tesz az $A_1A_2 \cdots A_n \rightarrow D$-nek is.

KIINDULÁS: Az indukció 0. lépése. Ekkor D csak az $A_1, A_2, \ldots, A_n$ közül lehet az egyik, és az $A_1A_2 \cdots A_n \rightarrow D$ funkcionális függőség biztosan fennáll tetszőleges relációban, mivel ez triviális függőség.

INDUKCIÓ: Az indukcióhoz tegyük fel, hogy D-vel akkor bővítünk, amikor felhasználjuk a $B_1B_2 \cdots B_m \rightarrow D$ függőséget az S halmazból. Az indukciós feltétel miatt R eleget tesz az $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$ függőségnek. Tekintsünk két R-beli sort, melyek megegyeznek az összes $A_1, A_2, \ldots, A_n$-en. Mivel R megfelel az $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$ függőségnek, a két sornak meg kell egyeznie az összes $B_1, B_2, \ldots, B_m$-en is. Mivel R eleget tesz $B_1B_2 \cdots B_m \rightarrow D$-nek, azt is tudjuk, hogy ez a két sor megegyezik D-n. Tehát R eleget tesz $A_1A_2 \cdots A_n \rightarrow D$-nek is.

Miért talál meg minden igaz funkcionális függőséget a lezárási algoritmus?

Tegyük fel, hogy $A_1A_2\cdots A_n \to B$ volt az a függőség, amely nem következik S halmazból a 3.7. lezárási algoritmus szerint. Azaz S funkcionális függőségi halmazát felhasználva $\{A_1, A_2, \ldots, A_n\}$ halmaz lezárása nem tartalmazza B-t. Be kell látnunk, hogy $A_1A_2\cdots A_n \to B$ valójában nem következik S-ből. Tehát mutatnunk kell legalább egy olyan relációt, amelyre S minden funkcionális függősége teljesül, és még így sem teljesíti $A_1A_2\cdots A_n \to B$-t.

Egy ilyen I előfordulást viszonylag könnyű készíteni. A 3.5. ábrán is egy ilyet láthatunk. Az I-nek csak két sora van: t és s. A két sor $\{A_1, A_2, \ldots, A_n\}^+$ minden attribútumában megegyezik, de a többiben eltérnek egymástól. Először meg kell mutatnunk, hogy I kielégíti S összes függőségét, és azt, hogy $A_1A_2\cdots A_n \to B$-t pedig nem elégíti ki.

	$\{A_1, A_2, \ldots, A_n\}^+$	Más attribútumok
t:	1 1 1 ··· 1 1	0 0 0 ··· 0 0
s:	1 1 1 ··· 1 1	1 1 1 ··· 1 1

3.5. ábra. Egy I előfordulás, amely kielégíti S-et, de $A_1A_2\cdots A_n \to B$-t nem

Tegyük fel, hogy létezik néhány $C_1C_2\cdots C_k \to D$ függőség S-ben (a szabályok szétvágása után), amelyeket I nem elégít ki. Mivel I-nek összesen két sora van, s és t, így ezek azok a sorok, amelyek megsértik a $C_1C_2\cdots C_k \to D$ szabályt. Ez azt jelenti, hogy s és t megegyeznek a $\{C_1, C_2, \ldots, C_k\}$ attribútumokon, de eltérnek D-n. A 3.5. ábra alapján látható, hogy $C_1, C_2, \ldots, C_k$ az $\{A_1, A_2, \ldots, A_n\}^+$ halmazban vannak, mert csak olyan attribútumok, amelyeken t és s megegyezik. Hasonlóan D a másik attribútumhalmazba tartozik, mert t és s azokon az attribútumokon különbözik.

Ekkor viszont nem jól számítottuk ki a lezárást. $C_1C_2\cdots C_k \to D$-t fel kellett volna használnunk, hogy D-t X-hez adjuk, amikor $X = \{A_1, A_2, \ldots, A_n\}$ volt. Így arra következtethetünk, hogy $C_1C_2\cdots C_k \to D$ nem létezhet, vagyis az I előfordulása kielégíti S-et.

Másrészt be kell látnunk, hogy I nem elégíti ki $A_1A_2\cdots A_n \to B$-t. Ez a rész viszont egyszerű. $A_1, A_2, \ldots, A_n$ valójában azok az attribútumok, amelyeken t és s megegyeznek. Azt is tudjuk még, hogy B nincs benne $\{A_1, A_2, \ldots, A_n\}^+$-ban, hiszen B olyan attribútumokat jelent, ahol t és s eltérnek, vagyis I nem elégíti ki $A_1A_2\cdots A_n \to B$-t. Így levonhatjuk azt a következtetést, hogy a 3.7. lezárási algoritmus nem állítja sem több, sem kevesebb funkcionális függőségnek a teljesülését, csak pontosan azokat a függőségeket állítja igaznak, amelyek S-ből következnek.

Lezárások és kulcsok

Megjegyezzük, hogy $\{A_1, A_2, \ldots, A_n\}^+$ akkor és csak akkor az összes attribútumból álló halmaz, ha $A_1, A_2, \ldots, A_n$ a szóban forgó reláció szuperkulcsa. Csak ekkor határozza meg funkcionálisan $A_1, A_2, \ldots, A_n$ az összes attribútumot. Ezzel ellenőrizni tudjuk, hogy $A_1, A_2, \ldots, A_n$ a reláció kulcsa-e: először ellenőrizzük, hogy $\{A_1, A_2, \ldots, A_n\}^+$ tartalmazza-e az összes attribútumot, majd ellenőrizzük, hogy nincs olyan X halmaz, amelyet $\{A_1, A_2, \ldots, A_n\}$-ból kapnánk az egyik attribútum törlésével, és amelyre X^+ az összes attribútumot tartalmazná.

3.2.6. A tranzitivitási szabály

A tranzitivitási szabállyal két funkcionális szabályt kapcsolhatunk össze, általánosítva a 3.4. példa megállapítását:

- Ha $A_1A_2\cdots A_n \to B_1B_2\cdots B_m$ és $B_1B_2\cdots B_m \to C_1C_2\cdots C_k$ teljesül az R relációban, akkor $A_1A_2\cdots A_n \to C_1C_2\cdots C_k$ szintén teljesül R-ben.

Ha a C-k között vannak olyanok, amely az A-k között is megtalálhatók, azokat a triviális függőségi szabállyal kiküszöbölhetjük a jobb oldalról.

Ahhoz, hogy belássuk, miért teljesül a tranzitivitási szabály, felhasználjuk a 3.2.4. alfejezet ellenőrzési algoritmusát. Ahhoz, hogy ellenőrizzük, $A_1A_2\cdots A_n \to C_1C_2\cdots C_k$ teljesül-e, ki kell számolnunk $\{A_1, A_2, \ldots, A_n\}^+$-t a megadott függőségekre.

Az $A_1A_2\cdots A_n \to B_1B_2\cdots B_m$ funkcionális függőség miatt az összes $B_1, B_2, \ldots, B_m$ benne van $\{A_1, A_2, \ldots, A_n\}^+$-ban. Ezután alkalmazhatjuk a $B_1B_2\cdots B_m \to C_1C_2\cdots C_k$ függőséget, hogy $C_1, C_2, \ldots, C_k$-t hozzávegyük $\{A_1, A_2, \ldots, A_n\}^+$-hoz. Mivel az összes C benne van $\{A_1, A_2, \ldots, A_n\}^+$-ban, megkapjuk, hogy $A_1A_2\cdots A_n \to C_1C_2\cdots C_k$ fennáll minden olyan relációban, amely eleget tesz mind az $A_1A_2\cdots A_n \to B_1B_2\cdots B_m$, mind a $B_1B_2\cdots B_m \to C_1C_2\cdots C_k$ függőségeknek.

3.10. példa. Az alábbi példában a `Filmek` reláció egy újabb verzióját láthatjuk, amely tartalmazza a filmstúdiót és információkat is róla.

filmcím	*év*	*hossz*	*műfaj*	*stúdióNév*	*stúdióCím*
Csillagok háborúja	1977	124	sci-fi	Fox	Hollywood
Kutyahideg	2005	120	dráma	Disney	Buena Vista
Wayne világa	1992	95	vígjáték	Paramount	Hollywood

Feltehetjük, hogy fennáll az alábbi két függőség:

filmcím év → stúdióNév
stúdióNév → stúdióCím

Az első azért indokolt, mert csak egy film lehet adott címmel egy évben, és csak egy stúdióhoz tartozhat egy adott film. A másodikat az igazolja, hogy a stúdióknak egyértelmű címük van.

A tranzitív szabállyal a fenti két függőség kombinálásával kapunk egy új függőséget:

$$\texttt{filmcím év} \rightarrow \texttt{stúdióCím}$$

Ez a függőség azt jelenti, hogy a filmcím és az év (azaz a film) meghatározza a címet – a filmet gyártó stúdió címét. □

3.2.7. Funkcionális függőségi halmazok lezárása

Előfordulhat, hogy választhatunk, milyen funkcionális függőségeket használunk, amelyek reprezentálják egy reláció teljes FD-halmazát. Ha megadunk egy S függőséghalmazt (olyan függőségekből, amelyek fennállnak a relációban), akkor bármely S-sel ekvivalens függőséghalmazt S *bázis*ának nevezzük. Hogy elkerüljük a lehetséges halmazok számának megugrását, olyan függőségek vizsgálatára szorítkozunk, amelyeknek a jobb oldalán egyetlen attribútum szerepel. Ha van egy bázisunk, akkor a szétvágási szabályt alkalmazva a jobb oldalak ilyen alakúra hozhatók. A *minimális bázis* egy relációhoz az a B bázis, amely az alábbi három feltételt elégíti ki:

1. B összes függőségének jobb oldalán egy attribútum van.
2. Ha bármelyik B-beli függőséget elhagyjuk, a fennmaradó halmaz már nem bázis.
3. Ha bármelyik B-beli funkcionális függőség bal oldaláról elhagyunk egy vagy több attribútumot, akkor az eredmény már nem marad bázis.

Jegyezzük meg, hogy egyetlen triviális funkcionális függőség sem lehet a minimális bázisban, mert azt a 2. szabály szerint eltávolíthatjuk.

3.11. példa. Tekintsük az $R(A, B, C)$ relációt, amelynek minden attribútumától funkcionálisan függ a másik két attribútuma. A származtatott függőségek teljes halmaza hat függőséget tartalmaz, egyetlen attribútummal a jobb oldalon: $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow A$, $B \rightarrow C$, $C \rightarrow A$ és $C \rightarrow B$. Ezenkívül van három nem triviális funkcionális függőség, két attribútummal a bal oldalán: $AB \rightarrow C$, $AC \rightarrow B$ és $BC \rightarrow A$. Továbbá vannak olyan függőségek, amelyeknek több attribútum van a jobb oldalán, például $A \rightarrow BC$, vagy triviális függőségek, mint például $A \rightarrow A$.

Az R relációnak számos minimális bázisa van. Ezek közül az egyik:

$$\{A \rightarrow B,\ B \rightarrow A,\ B \rightarrow C,\ C \rightarrow B\}$$

Egy másik az $\{A \rightarrow B,\ B \rightarrow C,\ C \rightarrow A\}$. Ezeken kívül még több minimális bázisa van R-nek, amelyek megkeresését az olvasóra bízzuk. □

A levezetési szabályok egy teljes halmaza

Ha meg akarjuk tudni, hogy egy funkcionális függőség következik-e néhány adott függőségből, a 3.2.4. alfejezetben szereplő lezárás kiszámolását mindig használhatjuk. Nem árt tudni, hogy létezik a szabályoknak egy olyan halmaza, az úgynevezett *Armstrong-axiómák*, amelyek segítségével egy adott halmazból következő bármely funkcionális függőség levezethető. Ezek az axiómák:

1. *Reflexivitás.* Ha $\{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\}$, akkor $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$. Ezeket neveztük triviális függőségeknek.

2. *Bővítés.* Ha $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$, akkor

$$A_1A_2 \cdots A_nC_1C_2 \cdots C_k \rightarrow B_1B_2 \cdots B_mC_1C_2 \cdots C_k$$

bármely $\{C_1, C_2, \ldots, C_k\}$ attribútumhalmazra. Mivel tudjuk, hogy a C-k lehetnek A-k és B-k is, a bal oldalról ki kell szűrnünk a duplikált attribútumokat, majd ugyanezt kell tennünk a jobb oldalakkal is.

3. *Tranzitivitás.* Ha

$$A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m \quad \text{és} \quad B_1B_2 \cdots B_m \rightarrow C_1C_2 \cdots C_k,$$

akkor $A_1A_2 \cdots A_n \rightarrow C_1C_2 \cdots C_k$.

3.2.8. Funkcionális függőségek vetítése

Egy relációséma tervének vizsgálatánál a következő, funkcionális függőségekkel kapcsolatos kérdéseket is meg kell válaszolnunk. Tegyük fel, hogy adott az R reláció egy S funkcionális függőséghalmazzal, ahol elkészítjük R egy vetítését: $R_1 = \pi_L(R)$ R néhány attribútumára. Mely függőségek állnak fenn R_1-ben?

A választ elvileg az S-beli funkcionális függőségek vetületének kiszámításával nyerjük, mely függőségek:

a) S-ből levezethetők és

b) csak R_1 attribútumait tartalmazzák.

Mivel nagyon nagy számú ilyen funkcionális függőség lehet, és ezek közül sok redundáns is (azaz más ugyanilyen típusú függőségekből következik), a függőséghalmazt szabadon egyszerűsíthetjük. Általában viszont az R_1 funkcionális függőségeinek kiszámítási ideje a legrosszabb esetben az R_1-beli attribútumok

számának exponenciális függvénye. Az alábbiakban egy egyszerű algoritmus összefoglalása látható.

3.12. algoritmus. Funkcionális függőséghalmaz vetületének kiszámítása

BEMENET: Egy R reláció, és egy másik R_1 reláció, amelyet az $R_1 = \pi_L(R)$ vetítéssel számítottunk ki. Továbbá adott egy S függőséghalmaz, mely fennáll R-ben.

KIMENET: Az R_1-ben fennálló funkcionális függőségek halmaza.

ELJÁRÁS:

1. Legyen T a végül előálló funkcionális függőségek halmaza. Kezdetben T üres.
2. Minden olyan X attribútumhalmazra, amely R_1-nek része, számítsuk ki X^+-t. Ezt a kiszámítást az S-beli funkcionális függőségeket figyelembe véve végezzük, és előfordulhatnak olyan attribútumok is, amelyek R-ben szerepelnek, de R_1-ben nem. Adjuk hozzá T-hez az összes nem triviális függőséget, amelyek $X \to A$ formátumúak, ahol A eleme az X^+ és az R_1 attribútumhalmaznak is.
3. Ezután a kapott T bázisa az R_1-beli funkcionális függőségeknek, de nem feltétlen minimális bázis. A minimális bázist előállíthatjuk T-ből az alábbi módosítások végrehajtásával:
   a) Ha szerepel egy F funkcionális függőség T-ben, amely más T-beli függőségekből következik, akkor töröljük F-et a T halmazból.
   b) Legyen $Y \to B$ egy T-beli funkcionális függőség, ahol Y legalább két attribútumot tartalmaz, és legyen Z az a halmaz, amelyet úgy kapunk, hogy Y-ból egy attribútumot elhagyunk. Ha $Z \to B$ függőség következik a T-beli funkcionális függőségekből (beleértve $Y \to B$-t is), akkor cseréljük ki $Y \to B$-t $Z \to B$-re.
   c) Ismételjük az előző lépéseket addig, amíg már nem lehet semmilyen módosítást végrehajtani T-n.

□

3.13. példa. Vegyük az $A \to B$, $B \to C$ és $C \to D$ funkcionális függőségekkel rendelkező $R(A, B, C, D)$ relációt. Tegyük fel azt is, hogy a B attribútumot akarjuk elhagyni, hogy vetítéssel egy $R_1(A, C, D)$ relációt kapjunk. Elviekben az R_1 funkcionális függőségeinek megtalálásához vennünk kell az $\{A, C, D\}$ mind a nyolc részhalmazának a lezártját az összes funkcionális függőség figyelembevételével (beleértve a B-t érintőket is). Van viszont néhány nyilvánvaló egyszerűsítési lehetőség, amelyeket elvégezhetünk.

- Az üres, illetve a minden attribútumot tartalmazó halmaz lezárása nem vezet nem triviális funkcionális függőséghez.

- Ha tudjuk, hogy egy X halmaz lezárása tartalmazza az összes attribútumot, akkor nem tudunk újabb funkcionális függőséghez jutni az X szuperhalmazainak lezárásával.

Azaz kezdhetünk az egyértékű halmazok lezártjával, majd utána folytathatjuk a kétértékű halmazokkal, ha szükséges. Minden X halmaz lezártjára hozzátesszük az $X \to E$ funkcionális függőséget az R_1 séma minden olyan E attribútumára, amely X^+-ban benne van, de nincs benne X-ben.

Először $\{A\}^+ = \{A, B, C, D\}$, azaz $A \to C$ és $A \to D$ fennáll R_1-ben. Figyeljük meg, hogy $A \to B$ fennáll R-ben, viszont R_1-ben nem értelmezhető, hiszen B nem attribútuma R_1-nek.

Majd tekintsük $\{C\}^+ = \{C, D\}$-t, amelyből a $C \to D$ függőséget kapjuk. Mivel $\{D\}^+ = \{D\}$, nem tudunk további FD-ket felvenni, és az egyelemű halmazokkal végeztünk.

Mivel $\{A\}^+$ tartalmazza R_1 minden attribútumát, ezért nem érdemes $\{A\}$ szuperhalmazával sem foglalkozni. Ennek oka a megtalálható funkcionális függőségek típusa. Például az $AC \to D$ függőség következik abból a függőségből, amelynek bal oldalán csak A szerepel, azaz most $A \to D$-ből. Az egyetlen kételemű halmaz, amelyet meg kell vizsgálni, a $\{C, D\}^+ = \{C, D\}$ halmaz. Ez a vizsgálat sem nyújt semmi újat. A lezártak előállításával végeztünk, a megtalált funkcionális függőségek: $A \to C$, $A \to D$ és $C \to D$.

Ha szeretnénk, akkor azt is megfigyelhetjük, hogy $A \to D$ a másik két szabályból a tranzitivitással már következik. Ezért egy egyszerűbb, ekvivalens funkcionális függőségi halmazt kapunk R_1-re, amelyben $A \to C$ és $C \to D$. Ez a halmaz valójában az R_1-beli funkcionális függőségek minimális bázisa. □

3.2.9. Feladatok

3.2.1. feladat. Tekintsünk egy relációt $R(A, B, C, D)$ sémával és $AB \to C$, $C \to D$ és $D \to A$ funkcionális függőségekkel.

a) Melyek az összes nem triviális függőségek, amelyek az adott függőségekből következnek? Csak olyan függőségekkel foglalkozzunk, amelyek jobb oldala egy attribútumból áll.

b) Melyek az R összes kulcsai?

c) Melyek az R összes olyan szuperkulcsai, amelyek nem kulcsok?

3.2.2. feladat. Ismételjük meg a 3.2.1. feladatot az alábbi sémákra és függőségekre:

i) $S(A, B, C, D)$, az $A \to B$, $B \to C$ és $B \to D$ funkcionális függőségekkel.

ii) $T(A, B, C, D)$, az $AB \to C$, $BC \to D$, $CD \to A$ és $AD \to B$ funkcionális függőségekkel.

iii) $U(A, B, C, D)$, az $A \to B$, $B \to C$, $C \to D$ és $D \to A$ funkcionális függőségekkel.

3.2.3. feladat. Mutassuk meg, hogy az alábbi szabályok érvényesek, használva a 3.2.4. alfejezet lezárási algoritmusát.

a) Bővítés a bal oldalon. Ha $A_1A_2 \cdots A_n \to B$ funkcionális függőség és C egy másik attribútum, akkor $A_1A_2 \cdots A_nC \to B$ fennáll.

b) Teljes bővítés. Ha $A_1A_2 \cdots A_n \to B$ funkcionális függőség és C egy másik attribútum, akkor $A_1A_2 \cdots A_nC \to BC$ fennáll. Megjegyezzük, hogy ebből a szabályból könnyen belátható a „bővítési" szabály, amit a 3.2.7. alfejezet keretes írásában („A levezetési szabályok egy teljes halmaza") állítottunk.

c) Pszeudotranzitivitás. Tegyük fel, hogy $A_1A_2 \cdots A_n \to B_1B_2 \cdots B_m$ és $C_1C_2 \cdots C_k \to D$ funkcionális függőségek fennállnak, és B-k mindegyike a C-k között előfordul. Ekkor

$$A_1A_2 \cdots A_nE_1E_2 \cdots E_j \to D$$

fennáll, ahol E-k éppen azon C-k közül valók, amelyek nincsenek B-k között.

d) Additivitás. Ha

$$A_1A_2 \cdots A_n \to B_1B_2 \cdots B_m \quad \text{és} \quad C_1C_2 \cdots C_k \to D_1D_2 \cdots D_j$$

funkcionális függőségek fennállnak, akkor

$$A_1A_2 \cdots A_nC_1C_2 \cdots C_k \to B_1B_2 \cdots B_mD_1D_2 \cdots D_j$$

is fennáll. Ebben a függőségben ki kellene törölnünk azoknak az attribútumoknak az egyik másolatát, amelyek mind A-k és C-k, valamint B-k és D-k között előfordulnak.

! **3.2.4. feladat.** Mutassuk meg, hogy az alábbiak nem érvényes szabályok a funkcionális függőségekre, úgy, hogy adjunk példát olyan relációra, amely eleget tesz a „Ha..." részben adott függőségeknek, de nem tesz eleget az állítás szerinti következménynek.

a) Ha $A \to B$, akkor $B \to A$.

b) Ha $AB \to C$ és $A \to C$, akkor $B \to C$.

c) Ha $AB \to C$, akkor $A \to C$ vagy $B \to C$.

! **3.2.5. feladat.** Mutassuk meg, hogy ha egy relációnak nincs olyan attribútuma, amelyet funkcionálisan meghatározna az összes többi attribútum, akkor a relációnak egyáltalán nincs nem triviális függősége.

! **3.2.6. feladat.** Legyenek X és Y attribútumokból álló halmazok. Mutassuk meg, hogy ha $X \subseteq Y$, akkor $X^+ \subseteq Y^+$, ahol a lezárásokat ugyanarra a funkcionális függőségekből álló halmazra számoljuk.

! **3.2.7. feladat.** Bizonyítsuk be, hogy $(X^+)^+ = X^+$.

!! **3.2.8. feladat.** Azt mondjuk, hogy X attribútumhalmaz *zárt* (a funkcionális függőségek egy adott halmazára nézve), ha $X^+ = X$. Tekintsünk egy $R(A, B, C, D)$ sémájú relációt és funkcionális függőségek egy ismeretlen halmazát. Ha tudjuk, mely attribútumhalmazok zártak, fel tudjuk tárni a funkcionális függőségeket. Melyek a funkcionális függőségek, ha:

a) a négy attribútum összes részhalmaza zárt;

b) a zárt halmazok csak az $\emptyset$ és az $\{A, B, C, D\}$;

c) a zárt halmazok az $\emptyset$, az {A,B} és az $\{A, B, C, D\}$.

! **3.2.9. feladat.** Keressük meg a 3.11. példában szereplő relációhoz és függőségekhez az összes minimális bázist.

! **3.2.10. feladat.** Tegyük fel, hogy adott egy $R(A, B, C, D, E)$ relációnk funkcionális függőségekkel, melyeket vetíteni szeretnénk az $S(A, B, C)$ relációra. Adjuk meg az S-ben érvényes függőségeket, ha R-ben a következők érvényesek:

a) $AB \to DE$, $C \to E$, $D \to C$ és $E \to A$.

b) $A \to D$, $BD \to E$, $AC \to E$ és $DE \to B$.

c) $AB \to D$, $AC \to E$, $BC \to D$, $D \to A$ és $E \to B$.

d) $A \to B$, $B \to C$, $C \to D$, $D \to E$ és $E \to A$.

Mind a négy esetben adjuk meg az S-ben érvényes függőségek egy minimális bázisát.

!! **3.2.11. feladat.** Mutassuk meg, hogy ha F funkcionális függőség következik adott funkcionális függőségekből, akkor F-et be tudjuk bizonyítani az adott függőségekből az Armstrong-axiómák (lásd a 3.2.7. alfejezet „A levezetési szabályok egy teljes halmaza" című keretes írását) segítségével. *Útmutatás:* Vizsgáljuk meg az attribútumhalmaz lezárását előállító 3.7. algoritmust, és lássuk be, hogy az algoritmus minden lépését helyettesíthetjük az Armstrong-axiómákból következő funkcionális függőségekkel.

3.3. Relációs adatbázissémák tervezése

Egy relációs adatbázisséma nem kellően gondos megtervezése redundanciához és anomáliákhoz vezet. Vizsgáljuk meg például a 3.2. ábrán látható relációt, amelyet most a 3.6. ábraként idemásolunk. Megjegyezzük, hogy a *Csillagok háborúja* és a *Wayne világa* filmek hossz és műfaj attribútumai minden egyes színész sorában megismétlődnek. Az információ ilyen megismétlése redundanciához vezet. Ahogy látni fogjuk, ez további hibalehetőségekhez vezet.

Ebben a fejezetben a jó relációsémák tervezésének a problémáit a következő lépésekben tárgyaljuk:

1. Először részletesebben feltárjuk, milyen problémák merülhetnek fel hibás séma esetén.

2. Ezután bevezetjük a „felbontás" (dekompozíció) ötletét, mellyel a relációsémát (attribútumok halmazát) széttördeljük kisebb sémákra.

3. A következő lépésben bevezetjük a „Boyce–Codd normálformát" vagy „BCNF"-et, amely a relációsémára jelent olyan feltételt, amellyel kiküszöböljük ezeket a problémákat.

4. Ezután összekapcsoljuk a két témakört, és megnézzük, hogyan biztosíthatjuk a BCNF feltételét a relációsémák felbontásával.

filmcím	*év*	*hossz*	*műfaj*	*stúdióNév*	*színészNév*
Csillagok háborúja	1977	124	sci-fi	Fox	Carrie Fisher
Csillagok háborúja	1977	124	sci-fi	Fox	Mark Hamill
Csillagok háborúja	1977	124	sci-fi	Fox	Harrison Ford
Elfújta a szél	1939	231	dráma	MGM	Vivien Leigh
Wayne világa	1992	95	vígjáték	Paramount	Dana Carvey
Wayne világa	1992	95	vígjáték	Paramount	Mike Meyers

3.6. ábra. A `Filmek1` reláció az anomáliák bemutatására

3.3.1. Anomáliák

Azokat a problémákat, amelyek a redundanciához hasonlóan akkor fordulnak elő, amikor túl sok információt próbálunk egyetlenegy relációba belegyömöszölni, *anomáliák*nak nevezzük. A számításba jövő anomáliák alapvető fajtái a következők:

1. *Redundancia.* Az információk feleslegesen ismétlődhetnek több sorban. A 3.6. ábrán a filmek hossza és a műfaj erre példa.

2. *Módosítási anomáliák.* Lehet, hogy megváltoztatjuk az egyik sorban tárolt információt, miközben ugyanaz az információ változatlanul marad egy másik sorban. Például, ha kiderül, hogy a *Csillagok háborúja* valójában 125 perces, – ha nem vagyunk elég gondosak – a 3.6. ábra első sorában változtatjuk meg a hosszt és a második vagy harmadik sorokban pedig nem. Persze mondatjuk, hogy nem szabadna ilyen figyelmetlennek lenni, de majd látni fogjuk, hogy a `Filmek1` relációt át lehet úgy tervezni, hogy az ilyen hibák veszélye egyáltalán ne merülhessen fel.

3. *Törlési anomáliák.* Ha az értékek halmaza üres halmazzá válik, akkor ennek mellékhatásaként más információt is elveszíthetünk. Például ha törölnünk kell az *Elfújta a szél*-ben szereplő színészek közül Vivien Leigh-t, akkor az adatbázisban ehhez a filmhez nem maradna több színész, így a `Filmek1` relációban az *Elfújta a szél* című filmre vonatkozó sor eltűnne azzal az információval együtt, hogy a film egy 231 perces dráma.

3.3.2. Relációk felbontása

Az anomáliák megszüntetésének az elfogadott útja a *relációk felbontása* (*dekompozíciója*). R felbontása azt jelenti, hogy R attribútumait szétosztjuk úgy, hogy két új reláció sémáját alakítjuk ki belőlük. A felbontási eljárás megadása után megmutatjuk, hogyan válasszunk ki egy olyan felbontást, amellyel megszüntetjük az anomáliákat.

Legyen adott az R reláció az $\{A_1, A_2, \ldots, A_n\}$ sémával. R-t felbonthatjuk $S(B_1, B_2, \ldots, B_m)$ és $T(C_1, C_2, \ldots, C_k)$ relációkra úgy, hogy

1. $\{A_1, A_2, \ldots, A_n\} = \{B_1, B_2, \ldots, B_m\} \cup \{C_1, C_2, \ldots, C_k\}$.
2. $S = \pi_{B_1, B_2, \ldots, B_m}(R)$.
3. $T = \pi_{C_1, C_2, \ldots, C_k}(R)$.

3.14. példa. Bontsuk fel a 3.6. ábrán látható `Filmek1` relációt. A választott felbontás, amelynek az előnyeit a 3.3.3. alfejezetben fogjuk látni, az alábbi:

1. A `Filmek2` séma tartalmazza az összes attribútumot, kivéve a **`színészNév`** nevűt.
2. A `Filmek3` séma pedig tartalmazza a **`filmcím`**, **`év`** és **`színészNév`** attribútumokat.

A `Filmek1` reláció vetítése erre a két sémára a 3.7. ábrán látható. □

Figyeljük meg, hogyan szünteti meg ez a felbontás a 3.3.1. alfejezetben említett anomáliákat. A redundanciát kizártuk, például minden film hossza csak egyszer fordul elő a `Filmek2` relációban. A módosítási anomália kockázata megszűnt. Például, mivel csak a `Filmek2` egy sorában kell megváltoztatnunk a *Csillagok háborúja* film hosszát, ezért nem fordulhat elő, hogy ennek a filmnek két

filmcím	*év*	*hossz*	*műfaj*	*stúdióNév*
Csillagok háborúja	1977	124	sci-fi	Fox
Elfújta a szél	1939	231	dráma	MGM
Wayne világa	1992	95	vígjáték	Paramount

(a) A Filmek2 reláció

filmcím	*év*	*színészNév*
Csillagok háborúja	1977	Carrie Fisher
Csillagok háborúja	1977	Mark Hamill
Csillagok háborúja	1977	Harrison Ford
Elfújta a szél	1939	Vivien Leigh
Wayne világa	1992	Dana Carvey
Wayne világa	1992	Mike Meyers

(b) A Filmek3 reláció

3.7. ábra. A Filmek1 reláció vetületei

különböző hosszát tároljuk véletlenül. Végül a törlési anomália kockázata is megszűnt. Ha töröljük az *Elfújta a szél* című filmben szereplő összes színészt, akkor ez a film kiesik a Filmek3 relációból, de az összes többi információ erről a filmről még megtalálható lesz a Filmek2-ben.

Még mindig előfordulhat redundancia a Filmek3-ban, mivel egy film címe és éve többször is szerepelhet. Jóllehet ez a két attribútum a filmeknek egy kulcsát alkotja, mégsem tudjuk másképpen szabatosan reprezentálni a filmet. Továbbá a Filmek3-ban mégsem merül fel a módosítási anomália lehetősége. Ha megváltoztatjuk a *Csillagok háborúja* évét 2008-ra Carrie Fisher sorában, de a másik két sorban nem, akkor lehetne módosítási anomália. Jóllehet egyik feltételezett funkcionális függőség sem akadályozza meg, hogy ne készüljön *Csillagok háborúja* című másik film 2008-ban, és Carrie Fisher szerepelhet abban is. Tehát nem kívánjuk megakadályozni, hogy a *Csillagok háborúja* egyik sorában az évet megváltoztassuk, sőt még csak azt sem akarjuk állítani, hogy egy ilyen változás biztosan hibás eredményre vezetne.

3.3.3. Boyce–Codd normálforma

A felbontás célja, hogy egy relációt többel helyettesítsünk úgy, hogy ezzel megszüntessük az anomáliákat. Kiderült, hogy van egy egyszerű feltétel, ami biztosítja, hogy a korábban tárgyalt anomáliák ne fordulhassanak elő. Ezt a feltételt *Boyce–Codd normálformá*nak vagy *BCNF*-nek nevezzük.

- Az R reláció BCNF-ben van akkor és csak akkor, ha minden olyan esetben, ha R-ben érvényes egy $A_1A_2\cdots A_n \to B_1B_2\cdots B_m$ nem triviális függőség, akkor az $\{A_1, A_2, \ldots, A_n\}$ halmaz R szuperkulcsa.

Azaz minden nem triviális funkcionális függőség bal oldalának szuperkulcsnak kell lennie. Emlékeztetünk arra, hogy a szuperkulcsnak nem kell minimálisnak lennie. Emiatt a BCNF feltétel egy ekvivalens megfogalmazása az, hogy minden nem triviális funkcionális függőség bal oldalának tartalmaznia kell egy kulcsot.

3.15. példa. A 3.6. ábrán látható `Filmek1` reláció nincs BCNF-ben. Ahhoz, hogy belássuk, miért nem, először meg kell határoznunk, mely attribútumhalmazok alkotják a kulcsokat. A 3.2. példában láttuk, hogy a {`filmcím`, `év`, `színészNév`} miért alkot kulcsot. Tehát bármely olyan attribútumhalmaz, amely tartalmazza ezt a hármat, szuperkulcs. A 3.2. példa megfontolásaihoz hasonlóan beláthatjuk azt is, hogy nem lehet szuperkulcs az olyan attribútumhalmaz, amely nem tartalmazza mindhármat a `filmcím`, `év` és `színészNév` közül. Tehát azt kaptuk, hogy a {`filmcím`, `év`, `színészNév`} az egyetlen kulcs a `Filmek1` relációhoz.

Tekintsük a következő funkcionális függőséget:

`filmcím év` → `hossz műfaj stúdióNév`

amelyről a 3.2. példából tudjuk, hogy fennáll a `Filmek1` relációban.

Sajnos a fenti függőség bal oldala nem szuperkulcs, hiszen tudjuk, hogy a `filmcím` és `év` funkcionálisan nem határozza meg a hatodik attribútumot, a `színészNév` attribútumot. Azaz ennek a függőségnek a létezése megsérti a BCNF feltételt, és emiatt a `Filmek1` nincs BCNF-ben. □

3.16. példa. Másrészt a 3.7. ábrán látható `Filmek2` reláció BCNF-ben van. Mivel

`filmcím év` → `hossz műfaj stúdióNév`

fennáll ebben a relációban, és korábban láttuk, hogy sem a `filmcím` sem az `év` önmagában nem határozza meg funkcionálisan a többi attribútum egyikét sem, ezért a `Filmek2` reláció egyetlen kulcsa a {`filmcím`, `év`}. Továbbá a nem triviális funkcionális függőségek legalább a `filmcím` és `év` attribútumokat tartalmazzák a bal oldalon. Emiatt a bal oldalak szuperkulcsok, tehát a `Filmek2` BCNF-ben van. □

3.17. példa. Azt állítjuk, hogy bármely két attribútumból álló reláció BCNF-ben van. Azokat a lehetséges nem triviális függőségeket kell megvizsgálnunk, amelyeknek jobb oldala egyetlen attribútumból áll. Nincs túl sok eset, amit figyelembe kell vennünk, ezért nézzük meg ezeket sorban. Tegyük fel, hogy az attribútumok A és B.

1. Nincs nem triviális függőség. Ekkor a BCNF feltétel mindenképpen érvényes, hiszen csak a nem triviális függőségek sérthetik meg a feltételt. Megjegyezzük egyébként, hogy $\{A, B\}$ az egyetlen kulcs ebben az esetben.

2. $A \rightarrow B$ fennáll, de $B \rightarrow A$ nem áll fenn. Ebben az esetben A az egyedüli kulcs, és minden nem triviális függőség tartalmazza A-t a bal oldalon (valójában a bal oldal csak A lehet). Tehát nem sérül a BCNF feltétel.

3. $B \rightarrow A$ fennáll, de $A \rightarrow B$ nem áll fenn. Ez az eset szimmetrikus az előző, 2. esettel.

4. Mindkét $A \rightarrow B$ és $B \rightarrow A$ fennáll. Ekkor mindkét attribútum, A és B is kulcs. Bármelyik függőséget nézzük, a két attribútum közül az egyik a bal oldalon áll, emiatt nem sértheti meg a BCNF feltételt.

Érdemes megjegyeznünk a 4. eset kapcsán, hogy egy relációnak több kulcsa lehet. Továbbá a BCNF feltétel csak azt követeli meg, hogy bármely nem triviális függőség bal oldala tartalmazzon valamilyen kulcsot, de nem kell minden kulcsot tartalmaznia a bal oldalnak. Azt is megjegyezzük, hogy a kétattribútumú relációnál az az eset, amikor bármelyik attribútum funkcionálisan meghatározza a másikat, nem teljesen valószínűtlen. Például egy vállalat a dolgozóinak egyértelmű alkalmazottazonosítót adhat meg, és a személyi számukat is bejegyzik. Abban a relációban, amelyiknek a `alkalmazottAzonosító` és a `személyiSzám` az attribútumai, bármelyik attribútum funkcionálisan meghatározza a másikat. Másképpen kifejezve, mindkét attribútum külön-külön kulcs, mivel várhatóan nem találunk két olyan sort, ahol az egyes attribútumértékek megegyeznének. □

3.3.4. Boyce–Codd normálformájú felbontás

Alkalmas felbontások ismétlődő választásával bármely relációsémát fel tudunk bontani az attribútumaiból álló részhalmazok összességére, amelyre az alábbi fontos tulajdonságok teljesülnek:

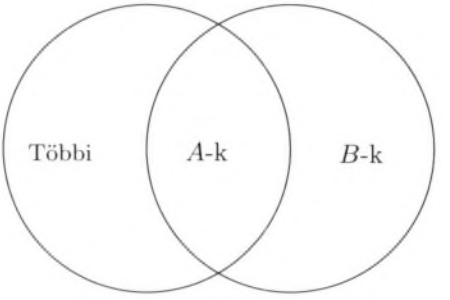
1. Ezek a részhalmazok BCNF-ben levő relációsémák.

2. Az eredeti relációban tárolt adatokat pontosan reprezentálják a felbontás eredményeként nyert relációk adatai. A 3.4.1. alfejezetben pontosítjuk, hogyan értjük ezt. Durván megfogalmazva arra van szükségünk, hogy a felbontással kapott relációk előfordulásaiból képesek legyünk pontosan visszaállítani az eredeti reláció előfordulását.

A 3.17. példa azt sugallja, hogy talán csak annyit kell tennünk, hogy a relációsémát kétattribútumos részhalmazokra bontjuk fel, és ennek az eredménye feltétlenül BCNF-ben van. Azonban ezek a felbontások nem mindig teljesítik a 2. feltételt, ahogyan később a 3.4.1. alfejezetben látni fogjuk. Valójában óvatosabban kell haladnunk, és a BCNF-et megszegő funkcionális függőségeket használjuk majd a felbontás vezérfonalaként.

Azt a felbontási stratégiát követjük, hogy kiválasztunk egy $A_1A_2\cdots A_n \rightarrow B_1B_2\cdots B_m$ nem triviális funkcionális függőséget, amely megsérti a BCNF-et, azaz $\{A_1, A_2, \ldots, A_n\}$ nem szuperkulcs. A jobb oldalhoz hozzáadjuk az összes

$\{A_1, A_2, \ldots, A_n\}$ által funkcionálisan meghatározott attribútumot. Ez a lépés nem kötelező, de többnyire csökkenti a végrehajtás összköltségét, ezért be fogjuk építeni az algoritmusunkba. A 3.8. ábrán látható, hogyan bonthatók az attribútumok két, egymást részben fedő relációsémára. Az egyik az összes olyan attribútum, amely a feltételt megsértő függőségben előfordul, a másik pedig a függőség bal oldala és azok az attribútumok, amelyek *nem* szerepelnek a függőségben, például az összes attribútum, kivéve azokat a B-ket, amelyek nem A-k.



3.8. ábra. BCNF megsértésén alapuló relációséma felbontása

3.18. példa. Vizsgáljuk meg a futó példánkat, a `Filmek1`-et a 3.6. ábrán. A 3.15. példában láttuk, hogy a

`filmcím év` → `hossz műfaj stúdióNév`

függőség megsérti a BCNF feltételt. Ebben az esetben a jobb oldal már tartalmazza azokat az attribútumokat, amelyeket a `filmcím` és az `év` funkcionálisan meghatároz, tehát a BCNF megsértése miatt a `Filmek1` relációt kétfelé bontjuk:

1. A {`filmcím`, `év`, `hossz`, `műfaj`, `stúdióNév`} reláció tartalmazza az összes attribútumot a függőség mindkét oldaláról.

2. A {`filmcím`, `év`, `színészNév`} séma pedig tartalmazza a függőségek bal oldalát és a `Filmek1` azon attribútumait, amelyek nem fordulnak elő a függőségben (ebben az esetben csak a `színészNév`).

Megjegyezzük, hogy ezek a sémák a 3.14. példában szereplő `Filmek2` és `Filmek3` sémák. A 3.16. példában láttuk, hogy a `Filmek2` BCNF-ben van. A `Filmek3` szintén BCNF-ben van, nincs egyetlen nem triviális függősége sem. □

A 3.18. példában elég volt a dekompozíciós szabályt józan ésszel egyszer alkalmazni ahhoz, hogy BCNF-ben lévő relációk egy csoportját kapjuk. Általában nem ez a helyzet, ahogy a következő példa is mutatja.

3.19. példa. Vizsgáljuk meg a

$$\{\texttt{filmcím, év, stúdióNév, elnök, elnökCím}\}$$

sémát. Azaz minden sorban tároljunk egy filmet, a stúdiót, ahol készült, a stúdió vezetőjét és az ő lakcímét. Feltételezzük az alábbi függőségek teljesülését:

$$\begin{aligned}&\texttt{filmcím év} \to \texttt{stúdióNév}\\ &\texttt{stúdióNév} \to \texttt{elnök}\\ &\texttt{elnök} \to \texttt{elnökCím}\end{aligned}$$

Az öt attribútumból álló halmaz lezárásával azt tapasztalhatjuk, hogy ennek a relációnak az egyetlen kulcsa a $\{\texttt{filmcím, év}\}$. Így a fent említett függőségek közül az utolsó kettő megsérti a BCNF feltételt. Tegyük fel, hogy a szétvágást a

$$\texttt{stúdióNév} \to \texttt{elnök}$$

függőséggel kezdjük. Először hozzáadjuk ennek a függőségnek a jobb oldalához az összes attribútumot, ami a `stúdióNév` lezártjában szerepel. A lezárt tartalmazza az `elnökCím` attribútumot, tehát a szétvágáshoz választott függőség végül:

$$\texttt{stúdióNév} \to \texttt{elnök elnökCím}$$

A dekompozíció, amelyet e szerint a függőség szerint hajthatunk végre, az alábbi két relációsémához vezet:

$$\begin{aligned}&\{\texttt{filmcím, év, stúdióNév}\}\\ &\{\texttt{stúdióNév, elnök, elnökCím}\}\end{aligned}$$

Ha a 3.12. algoritmust használjuk a függőség vetületének elkészítéséhez, akkor megállapíthatjuk, hogy az első reláción a függőségek bázisa

$$\texttt{filmcím év} \to \texttt{stúdióNév}$$

míg a másodikon

$$\begin{aligned}&\texttt{stúdióNév} \to \texttt{elnök}\\ &\texttt{elnök} \to \texttt{elnökCím}\end{aligned}$$

Az egyetlen kulcs az első relációban a $\{\texttt{filmcím, év}\}$, és ennek következtében BCNF-ben van. Ugyanakkor a másodiknak a $\{\texttt{stúdióNév}\}$ az egyetlen kulcsa, de érvényes benne az

$$\texttt{elnök} \to \texttt{elnökCím}$$

függőség, ami a BCNF feltétel megsértése. Azaz ismét dekompozíciót kell végrehajtanunk, ezúttal az előbbi függőséggel. A kapott három relációséma mindegyike BCNF-ben van:

{filmcím, év, stúdióNév}
{stúdióNév, elnök}
{elnök, elnökCím}

□

Általában addig kell alkalmazni a dekompozíciós szabályt, amíg minden kapott séma a BCNF feltételnek megfelelő nem lesz. Biztosak lehetünk abban, hogy az eljárás véget ér, ugyanis minden alkalommal a dekompozíciós szabályt alkalmazva R-re a kapott relációk kevesebb attribútumot fognak tartalmazni, mint R-ben volt. Ahogy láttuk a 3.17. példában, amint elérünk a két attribútumig, már biztosan BCNF-sémához jutunk; gyakran a nagyobb attribútumhalmazú relációk is BCNF-ben vannak. A módszer az alábbiakban látható:

3.20. algoritmus. BCNF-dekompozícó algoritmusa

BEMENET: Az R_0 reláció az S_0 függőséghalmazzal.

KIMENET: Az R_0 dekompozíciója relációk csoportjára, melyek mindegyike BCNF-ben van.

ELJÁRÁS: A következő lépések bármely R relációra és S függőséghalmazra alkalmazhatók rekurzívan. Kezdetben legyen $R = R_0$ és $S = S_0$.

1. Ellenőrizzük, R BCNF-ben van-e. Ha igen, akkor készen vagyunk, és $\{R\}$ a válasz.

2. Ha vannak BCNF-et megsértő függőségek, akkor $X \to Y$ legyen egy ilyen. Használjuk a 3.7. algoritmust X^+ kiszámításához. Legyen $R_1 = X^+$ az egyik relációséma, továbbá R_2-ben legyenek benne X attribútumai és azok az R-beli attribútumok, amelyek nincsenek X^+-ban.

3. Használjuk a 3.12. algoritmust az R_1 és R_2 függőségeinek meghatározásához, a kapottak legyenek rendre S_1 és S_2.

4. Rekurzívan bontsuk fel R_1-et és R_2-t ennek az algoritmusnak a használatával. A végeredmény a dekompozíciók eredményeinek uniója lesz.

□

3.3.5. Feladatok

3.3.1. feladat. A következő relációsémákra és funkcionális függőségi halmazokra:

a) $R(A, B, C, D)$ sémára és $AB \to C$, $C \to D$, $D \to A$ funkcionális függőségekre;

b) $R(A, B, C, D)$ sémára és $B \to C$, $B \to D$ funkcionális függőségekre;

c) $R(A,B,C,D)$ sémára és $AB \to C$, $BC \to D$, $CD \to A$, $AD \to B$ funkcionális függőségekre;

d) $R(A,B,C,D)$ sémára és $A \to B$, $B \to C$, $C \to D$, $D \to A$ funkcionális függőségekre;

e) $R(A,B,C,D,E)$ sémára és $AB \to C$, $DE \to C$, $B \to D$ funkcionális függőségekre;

f) $R(A,B,C,D,E)$ sémára és $AB \to C$, $C \to D$, $D \to B$, $D \to E$ funkcionális függőségekre

végezzük el az alábbiakat:

i) Jelezzünk minden BCNF-megsértést. Ne feledkezzünk el azokról a függőségekről sem, amelyek nincsenek az adott halmazban, de következnek belőle. Nem szükséges megadnunk azokat a megsértéseket, amelyek jobb oldalán egynél több attribútum áll.

ii) Bontsuk fel a relációkat, amelyeket szükséges, BCNF-ben levő relációkra.

3.3.2. feladat. A 3.3.4. alfejezetben említettük, hogy gyakoroljuk annak a lehetőségnek a használatát, hogy a függőségek jobb oldalát kiterjesztjük, amíg egy BCNF-megsértés lehetséges. Vizsgáljuk az R relációt, melynek sémája $\{A,B,C,D\}$, az érvényes függőségei pedig $A \to B$ és $A \to C$. Mindkettő egy-egy BCNF-megsértés, mert az R egyetlen kulcsa $\{A,D\}$. Tegyük fel, hogy az R dekompozícióját $A \to B$ szerint végezzük. Ugyanazt kapjuk végeredményként, ha kiterjesztjük a BCNF-et sértő függőséget $A \to BC$-re? Miért? Miért nem?

! **3.3.3. feladat.** Legyen R ugyanaz a reláció, mint a 3.3.2. feladatban, de most az $A \to B$ és $B \to C$ funkcionális függőségek legyenek érvényesek. Újból hasonlítsuk össze R felbontását, amikor az $A \to B$ funkcionális függőségek legyenek érvényesek. Újból hasonlítsuk össze R felbontását, amikor az $A \to BC$-t vesszük először.

! **3.3.4. feladat.** Tegyük fel, hogy van egy $R(A,B,C)$ relációsémánk az $A \to B$ funkcionális függőséggel. Azt is tegyük fel, hogy felbontottuk ezt $S(A,B)$ és $T(B,C)$ sémákra. Adjunk példát az R reláció egy olyan előfordulására, amelynek az S-re és T-re vett vetítéseit újból összekapcsolva (mint a 3.4.1. alfejezetben), nem ugyanazt a reláció-előfordulást adja vissza, azaz $\pi_{A,B}(R) \bowtie \pi_{B,C}(R) \neq R$.

3.4. Dekompozíció: a jó, a rossz és a csúf

Eddig láttuk, hogy a BCNF-re bontás előtt a relációsémában lehetségesek az anomáliák; a dekompozíció után a kapott eredményekben nem fordulhatnak elő anomáliák. Ez a „jó". A dekompozíciónak ugyanakkor van néhány rossz, sőt kimondottan „csúf" következménye is. Ebben a részben megvizsgálunk három különböző tulajdonságot, amelyekről szeretnénk, ha a dekompozícióra teljesülne.

1. *Anomáliák kiküszöbölése* dekompozícióval, ahogyan a 3.3. alfejezetben láttuk.

2. *Információ-visszaállíthatóság.* Vissza tudjuk-e állítani az eredeti relációt a dekompozícióval kapott relációk rekordjaiból?

3. *Függőségek megőrzése.* Ha ellenőrizzük a függőségek vetületét a dekompozícióval kapott relációkon, akkor garantálni tudjuk-e, hogy ha felépítjük az eredeti relációt a komponensek összekapcsolásával, akkor az eredmény kielégíti az eredeti függőségeket?

Kiderül, hogy a 3.20. BCNF-algoritmus teljesíti az 1. és 2. feltételeket, de nem szükségszerűen mindhármat. A 3.5. alfejezetben látni fogunk egy másik dekompozíciós módszert, amely biztosítja a 2. és 3. feltételeket, de nem feltétlenül az 1-et. Valójában nincs olyan módszer, amivel mindhárom biztosítható lenne egyszerre.

3.4.1. Információ visszanyerése a komponensekből

Mivel láttuk, hogy minden kétattribútumú reláció BCNF-ben van, felmerülhet, hogy miért kell a 3.20. algoritmus nehézségével foglalkoznunk? Miért nem mondjuk azt minden relációra, hogy felbontjuk olyan relációkra, melyek mindegyikének a sémája egy-egy attribútumpár R-ből? A válasz az, hogy az adatok a felbontással kapott relációkban nem biztos, hogy lehetővé teszik a relációk összekapcsolását és R visszaállítását, még akkor sem, ha mindegyik reláció R egy-egy vetülete. Ha R-et valóban visszakapjuk, akkor azt mondjuk, hogy a dekompozícióban *veszteségmentes összekapcsolás* van.

Ugyanakkor, ha a 3.20. algoritmussal végezzük a felbontást, ahol minden dekompozíció egy BCNF-et megsértő függőségből indul, akkor az eredeti reláció vetületeiként kapott rekordok összekapcsolhatók úgy, hogy csak az eredeti sorokat adják vissza. Itt mindjárt megmutatjuk, miért. Majd a 3.4.2. alfejezetben megadunk egy „chase" nevű algoritmust, amellyel ellenőrizhetjük, hogy a relációk vetülete bármely felbontásra lehetővé teszi-e a reláció visszaállítását újra-összekapcsolással.

A helyzet egyszerűsítése érdekében vizsgáljuk az $R(A, B, C)$ relációt és a $B \to C$ függőséget, ami a BCNF megsértése. A $B \to C$ függőség kettéosztja az attribútumokat az $R_1(A, B)$ és $R_2(B, C)$ relációkba.

Legyen t egy R-beli sor. Felírjuk t-t $t = (a, b, c)$ alakban, ahol a, b és c a t sor komponensei rendre az A, B és C attribútumokhoz. A t sor vetülete (a, b) az $R_1(A, B) = \pi_{A,B}(R)$-ben és (b, c) az $R_2(B, C) = \pi_{B,C}(R)$-ben. Amikor az $R_1 \bowtie R_2$ természetes összekapcsolást számítjuk ki, akkor ezek a rekordok összekapcsolódnak, mert megegyeznek a közös B komponensen (mivel mindkettő értéke b). Megadják a $t = (a, b, c)$ sort, azaz a kiinduló sort a kapcsolásban. Vagyis függetlenül attól, hogy milyen sorokból indultunk ki, a vetületek mindig összekapcsolhatók, hogy visszakapjuk a kiinduló sort.

Azonban a kiinduló sorok visszanyerése nem elegendő ahhoz, hogy biztosak lehessünk abban, hogy a dekompozícióval kapott relációk valóban az eredeti R relációt reprezentálják. Gondoljuk meg, mi történik akkor, amikor van két sorunk R-ben, például $t = (a, b, c)$ és $v = (d, b, e)$. Amikor levetítjük t-t $R_1(A, B)$-re, akkor $u = (a, b)$-t kapunk, és amikor levetítjük v-t $R_2(B, C)$-re, akkor $w = (b, e)$-t kapunk. Ezek a sorok szintén összeillenek a természetes összekapcsoláskor, és a kapott sor $x = (a, b, e)$ lesz. Lehetséges, hogy x egy hamis sor? Azaz lehet, hogy (a, b, e) nem az R egy sora?

Mivel feltettük, hogy a $B \to C$ érvényes R-re, a válasz „nem". Idézzük fel, hogy ez a függőség azt jelenti, hogy minden olyan sor, amely megegyezik a B komponensen, megegyezik a C komponensen is. Ez azt jelenti, hogy $c = e$, azaz a két attribútum, amelyet különbözőnek feltételeztünk, valójában ugyanaz. Tehát az (a, b, e) sor R-ben valójában (a, b, c), azaz $x = t$.

Mivel t szerepel R-ben, x-nek is benne kell lennie. Másként mondva, mivel a $B \to C$ függőség fennáll, a két rekord összekapcsolása nem vezethet hamis rekordhoz. Illetve minden természetes összekapcsolással megkapott rekord sora lesz az R relációnak.

Ez a megállapítás általánosan igaz. Feltettük, hogy A, B és C mind külön attribútumok voltak. De ugyanez az okoskodás mondható el akkor is, ha tetszőleges X, Y és Z attribútumhalmazról van szó. Azaz, ha $Y \to Z$ fennáll R-ben, melynek attribútumai $X \cup Y \cup Z$, akkor $R = \pi_{X \cup Y}(R) \bowtie \pi_{Y \cup Z}(R)$.

Összefoglalva:

- Ha a 3.20. algoritmus szerint végezzük a reláció dekompozícióját, akkor az eredeti reláció pontosan előállítható természetes összekapcsolással.

Hogy megértsük miért, a fenti érvelésről elmondhatjuk, hogy a rekurzív dekompozíció bármelyik lépése során a reláció ekvivalens a két komponensére vett vetületeinek összekapcsolásával. Ha ezeket a komponenseket tovább bontjuk, szintén visszaállíthatók lesznek a felbontással kapott relációk természetes összekapcsolásával. Azaz egy egyszerű indukcióval a bináris dekompozíciókon, lépésenként elmondhatjuk, hogy az eredeti reláció mindig azoknak a relációknak a természetes összekapcsolása, amelyekre felbontottuk. Bizonyíthatjuk továbbá, hogy a természetes összekapcsolás asszociatív és kommutatív, azaz a felbontott komponensek természetes összekapcsolásának sorrendje nem számít.

Az $Y \to Z$ függőség vagy a vele szimmetrikus $Y \to X$ alapvető. Ezek valamelyike nélkül nem tudnánk helyreállítani az eredeti relációt. Lássunk erre is példát.

3.21. példa. Vizsgáljuk a fenti $R(A, B, C)$ relációt, melyre a $B \to A$ és a $B \to C$ függőségek egyike sem teljesül. Továbbá R tartalmazza az alábbi két sort:

A	B	C
1	2	3
4	2	5

Az R reláció vetületei az $\{A, B\}$ és $\{B, C\}$ sémájú relációkra az alábbi $R_1 = \pi_{AB}(R) =$

A	B
1	2
4	2

és $R_2 = \pi_{BC}(R) =$

B	C
2	3
2	5

Mivel mind a négy rekord azonos B-értékkel rendelkezik (amely 2), mindkét reláció mindegyik sora összekapcsolható a másik reláció mindkét sorával. Amikor megpróbáljuk visszaállítani R-et, akkor a vetületrelációk természetes összekapcsolásával egy $R_3 = R_1 \bowtie R_2 =$

A	B	C
1	2	3
1	2	5
4	2	3
4	2	5

relációt kapunk. Az eredmény „túl sok", kapunk két hamis sort, az $(1, 2, 5)$-öt és a $(4, 2, 3)$-at, amelyek nem voltak benne az eredeti R relációban. □

3.4.2. Chase-teszt a veszteségmentes összekapcsoláshoz

A 3.4.1. alfejezetben megállapítottuk, hogy az $R(A, B, C)$ reláció dekompozíciójának az $\{A, B\}$ és $\{B, C\}$ sémákra az egyetlen $B \rightarrow C$ függőséggel van veszteségmentes összekapcsolása. Most tekintsünk egy általánosabb esetet. Felbontjuk az R relációt olyanokra, amelyek attribútumhalmazai $S_1, S_2, \ldots, S_k$. Adott egy F függőséghalmaz, amely fennáll R-ben. Igaz-e, hogy ha levetítjük R-et a dekompozícióval kapott relációkra, akkor visszakaphatjuk R-et a kapott relációk természetes összekapcsolásával? Azaz, igaz-e, hogy

$$\pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \cdots \bowtie \pi_{S_k}(R) = R?$$

Három fontos dologra kell emlékeznünk:

- A természetes összekapcsolás asszociatív és kommutatív. Nem számít, hogy milyen sorrendben kapcsoljuk össze a vetületeket, eredményként ugyanazt a relációt fogjuk kapni. Pontosabban az eredmény egy olyan t sorokból álló sorhalmaz, melyre minden $i = 1, 2, \ldots, k$-ra, t vetülete az S_i attribútumhalmazokra egy sor a $\pi_{S_i}(R)$-ben.

Az összekapcsolás az egyetlen módja az adatvisszanyerésnek?

Feltesszük, hogy az egyetlen lehetséges módszer az eredeti reláció felépítésének a vetületekből az, ha a természetes összekapcsolást használjuk. Talán lehetnek más algoritmusok is, amelyekkel előállíthatjuk az eredeti relációt, amely működhet olyan esetben is, amikor a természetes összekapcsolás sikertelenül végződik? Igazság szerint nincs másik út. A 3.21. példában láttuk, hogy az R és R_3 relációk különböző előfordulások, de mégis ugyanaz a vetületük az $\{A, B\}$ és $\{B, C\}$ attribútumhalmazokra, nevezetesen az R_1-nek és R_2-nek nevezett relációkra. Azaz adott R_1 és R_2 relációk esetén egyetlen algoritmus sem tud semmit mondani arról, hogy az eredeti reláció R vagy R_3.

Ráadásul ez a példa nem egyedi. Tekintsük az $X \cup Y \cup Z$ attribútumokból álló reláció bármely olyan dekompozícióját az $X \cup Y$ és az $Y \cup Z$ sémákra, ahol sem $Y \to X$, sem $Y \to Z$ nem áll fenn. Ehhez konstruálhatunk egy példát, amely hasonlít a 3.21. példában látotthoz, ahol az eredeti előfordulás nem határozható meg a vetületekből.

- Bármely t sor R-ben biztosan benne van $\pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \cdots \bowtie \pi_{S_k}(R)$-ben. Ennek oka az, hogy t vetületei az S_i halmazokra biztosan benne vannak $\pi_{S_i}(R)$-ben minden i-re, és ennek következtében, ahogy az előbb láttuk, t a kapcsolás eredménye.

- Következmény, hogy amikor az F-beli függőségek fennállnak R-en, $\pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \cdots \bowtie \pi_{S_k}(R) = R$ fennáll akkor és csak akkor, ha az összekapcsolás minden sora R-ben is benne van. Tehát csak ezt kell tesztelnünk, hogy bebizonyítsuk, a dekompozíció veszteségmentes összekapcsolású.

A *chase*-teszt a veszteségmentes kapcsoláshoz csupán egy szervezett módszer arra, hogy ellenőrizzük, egy $\pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \cdots \bowtie \pi_{S_k}(R)$-beli t sorról bizonyítható-e az F-beli függőségek használatával, hogy az R egy sora. Ha t szerepel az összekapcsolásban, akkor R-ben lenniük kell $t_1, t_2, \ldots, t_k$ soroknak úgy, hogy t az egyes $S_1, S_2, \ldots, S_k$ attribútumokra vett vetületek összekapcsolása minden $i = 1, 2, \ldots, k$-ra. Emiatt tudjuk, hogy t_i megegyezik t-vel az S_i attribútumain, de t_i-nek vannak ismeretlen értékei azokon a komponenseken, amelyek nincsenek S_i-ben.

Készíthetünk egy képet, amit *tablónak* nevezünk, és amelyen ábrázolhatjuk, hogy mit tudunk. Feltételezve, hogy R attribútumai $A, B, \ldots$, az $a, b, \ldots$ komponenseket használjuk t-hez. A t_i-khez ugyanazt a betűt használjuk, mint t-ben, hogy leírjuk az S_i-beli komponenseket, de alsó indexben i betűvel jelöljük azt a komponenst, amely nincs S_i-ben. Ezzel a módszerrel t_i megegyezik t-vel az S_i

attribútumain, de egyedi értéke van – azaz mindegyik csak egyszer fordulhat elő a tablóban – a többi helyen.

3.22. példa. Tegyük fel, hogy egy $R(A, B, C, D)$ relációnk van, amely felbontásra kerül az $S_1 = \{A, D\}$, $S_2 = \{A, C\}$ és $S_3 = \{B, C, D\}$ attribútumhalmazokra. Ennek a dekompozíciónak a tablója a 3.9. ábrán látható.

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

3.9. ábra. Tabló az R reláció felbontására $\{A, D\}$, $\{A, C\}$ és $\{B, C, D\}$-re

Az első sor megfelel az A és D attribútumhalmaznak. Jegyezzük meg, hogy az A és D attribútumokon szereplő komponensek a nem indexelt a és d betűk. Emellett a többi attribútumnak – b és c – adunk egy 1-es alsó indexet, hogy jelezzük, ezek tetszőleges értékek. A választás magyarázata, hogy az (a, b_1, c_1, d) R egy sorát jelenti, amely a $t = (a, b, c, d)$ sorból készült az $\{A, D\}$-re való vetítéssel, majd más sorokkal való összekapcsolással. Mivel a rekord B és C komponenseit a vetítéskor eldobtuk, nem tudunk semmit arról, hogy a sor milyen értékekkel rendelkezett ezeken az attribútumokon.

Hasonlóan a második sor index nélküli betűi az A és C attribútumokhoz tartoznak, míg a többi attribútumot 2-vel indexeljük. Az utolsó sor index nélküli elemei a $\{B, C, D\}$ komponensekben vannak, és a-t 3-mal indexeljük. Mivel minden sor saját számot használ az indexhez, a többször előforduló jelek csak indexeletlen betűk lehetnek. □

Emlékeztetünk rá, hogy a célunk az F függőséghalmaz felhasználásával bizonyítani, hogy t valóban szerepel R-ben. Ennek érdekében „kivadásszuk" („chaseeljük") a tablót az F-beli függőségeket alkalmazva, egyenlővé téve a szimbólumokat, amikor csak lehet. Ha egyszer azt kapjuk, hogy a sorok egyike t-vel egyezik meg (azaz a sorból eltűnik minden index nélküli szimbólum), akkor látjuk, hogy bármely t rekord, amely a vetületek összekapcsolásában szerepel, éppen egy rekordja R-nek.

A zűrzavar elkerülése érdekében két szimbólum egyenlővé tételekor, ha az egyik index nélküli, akkor a másik is azt az értéket kapja meg. Ha két indexelt szimbólumot kell egyenlővé tenni, akkor bármelyik indexet választhatjuk és megadhatjuk a másiknak. Mindemellett emlékezzünk rá, hogy szimbólumok egyenlővé tételekor az egyes jelek minden előfordulását le kell cserélni, nem csak néhány előfordulását.

3.23. példa. Folytassuk a 3.22. példát, és tegyük fel, hogy az $A \to B$, $B \to C$ és $CD \to A$ függőségek fennállnak. Kezdjük a 3.9. ábrán látható tablóval. Mivel az első két sor megegyezik az A attribútumon, az $A \to B$ függőség szerint a

B attribútumon is megegyeznek. Tehát $b_1 = b_2$. Bármelyiket lecserélhetjük a másikra, mivel mindkettő indexelt. Cseréljük most le b_2-t b_1-re. A kapott tabló:

A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

Most láthatjuk, hogy az első két sor megegyezik a B attribútumon is, tehát felhasználhatjuk a $B \to C$ függőséget, amellyel levezethetjük, hogy a C komponensei, c_1 és c megegyeznek. Mivel c indexeletlen, c_1-et cseréljük le c-re, így adódik:

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a_3	b	c	d

Következőként figyeljük meg, hogy az első és a harmadik sorok mind a C, mind a D oszlopban megegyeznek, azaz alkalmazhatjuk a $CD \to A$ függőséget, hogy levezessük, ezeknek a soroknak az A-értéke is azonos, azaz $a = a_3$. Lecseréljük a_3-t a-ra, mellyel adódik:

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a	b	c	d

Ezen a ponton láthatjuk, hogy az utolsó sor t-vel egyenlő lett, azaz (a, b, c, d). Ezzel beláttuk, hogy amennyiben R megfelel az $A \to B$, $B \to C$ és $CD \to A$ függőségeknek, ha levetítjük az $\{A, D\}$, $\{A, C\}$ és $\{B, C, D\}$ sémákra, majd újra összekapcsoljuk, akkor a kapott eredménynek benne kell lennie R-ben. Sőt pontosan megegyezik R-nek azzal a sorával, amelyet $\{B, C, D\}$-re vetítettünk. □

3.4.3. Miért működik a chase?

Két kérdést kell megválaszolni:

1. Amikor a chase előállít egy t-vel megegyező sort (azaz a tablóban van indexmentes sor), akkor ebből miért következik, hogy a kapcsolás veszteségmentes?

2. Ha az összes lehetséges függőség alkalmazása után sem találunk olyan sort, amelyben csak index nélküli változók vannak, miért nem lehet a kapcsolás veszteségmentes?

Az első kérdés megválaszolása egyszerű. A chase-eljárás maga egy bizonyítása annak, hogy a vetületsoroknak R-ből mindenképpen elő kell állítaniuk t-t a függőségekkel. Másrészt tudjuk, hogy minden R-beli sor elkészül, ha elvégezzük a vetítést és az összekapcsolást. Tehát a chase bebizonyítja, hogy a vetítés és az összekapcsolás eredménye pontosan R.

A második kérdéshez tegyük fel, hogy végül is egy olyan tablót állítunk elő, melyben nincs indexmentes sor, és ebben a tablóban már nem tudunk újabb függőségeket felhasználva szimbólumokat egyenlővé tenni. Gondoljunk a tablóra úgy, mint R egy előfordulására. Ez nyilvánvalóan kielégíti a megadott függőségeket, mert többet már nem lehet alkalmazni szimbólumok egyenlővé tételéhez. Tudjuk, hogy az i. sorban az indexmentes elemek S_i attribútumai között vannak, a dekompozíció i. relációjában. Így amikor levetítjük ezt a relációt S_i-re, majd vesszük a természetes összekapcsolást, akkor megkapjuk az index nélküli elemekből álló sort. Ez a sor nincs benne R-ben, tehát megállapíthatjuk, hogy az összekapcsolás nem veszteségmentes.

3.24. példa. Vizsgáljuk meg az $R(A, B, C, D)$ relációt a $B \to AD$ függőséggel és a javasolt felbontással $\{A, B\}$, $\{B, C\}$ és $\{C, D\}$ sémákra. Itt látható a kiinduló tabló:

A	B	C	D
a	b	c_1	d_1
a_2	b	c	d_2
a_3	b_3	c	d

Amikor alkalmazzuk az egyetlen függőséget, akkor levezethetjük, hogy $a = a_2$ és $d_1 = d_2$. Tehát az eredménytabló az alábbi:

A	B	C	D
a	b	c_1	d_1
a	b	c	d_1
a_3	b_3	c	d

További módosítást nem tudunk végrehajtani, mert nincs más függőség megadva, és nincsen olyan sor, amely teljesen indexmentes. Így ennek a dekompozíciónak nincs veszteségmentes összekapcsolása. Ezt az állítást beláthatjuk, ha felhasználjuk a fenti tablót, mint egy relációt három sorral. Amikor $\{A, B\}$-re vetítünk, $\{(a, b), (a_3, b_3)\}$ adódik. A vetület $\{B, C\}$-re $\{(b, c_1), (b, c), (b_3, c)\}$, és a vetület $\{C, D\}$-re $\{(c_1, d_1), (c, d_1), (c, d)\}$. Ha összekapcsoljuk az első két vetületet, akkor $\{(a, b, c_1), (a, b, c), (a_3, b_3, c)\}$ adódik. Ha ezt a relációt a harmadik vetülettel is összekapcsoljuk, akkor $\{(a, b, c_1, d_1), (a, b, c, d_1), (a, b, c, d), (a_3, b_3, c, d_1), (a_3, b_3, c, d)\}$ adódik. Megjegyezzük, hogy ez az összekapcsolás R-hez képest két további sort tartalmaz, sőt tartalmazza az (a, b, c, d) sort, ahogy szükséges. □

3.4.4. Függőségek megőrzése

Említettük, hogy néhány esetben lehetetlen a relációt BCNF-relációkra bontani úgy, hogy a kapott eredmény rendelkezzen veszteségmentes összekapcsolással és megőrizze a függőségeket is. Az alábbiakban egy olyan példa következik, melyben kompromisszumot kellett kötnünk a függőségek megőrzése és a BCNF között.

3.25. példa. Tegyük fel, hogy az alábbi `Vetítések` relációnk adott, a következő attribútumokkal:

1. `filmcím`, a film neve.
2. `mozi`, a mozi neve, ahol a filmet vetítik.
3. `város`, a város neve, ahol a mozi megtalálható.

Az (m, t, c) rekord jelentése, hogy az m című filmet jelenleg a t moziban játsszák, c városban.

Érthető, hogy feltételezhetjük az alábbi függőségeket:

$$\texttt{mozi} \rightarrow \texttt{város}$$
$$\texttt{filmcím város} \rightarrow \texttt{mozi}$$

Az első azt jelenti, hogy a mozi egyetlen városban van. A második nem teljesen nyilvánvaló, de azon a gyakorlaton alapul, hogy egyazon város két mozijában nem kötjük le ugyanazt a filmet egy időben.

Először keressük meg a kulcsokat. Egyetlen egyedülálló attribútum sem kulcs. Például a `filmcím` nem kulcs, mert a filmet játszhatják egyazon időben különböző városokban és különböző mozikban.[4] A `mozi` sem kulcs, mert bár a `mozi` funkcionálisan meghatározza a `város`-t, vannak többvásznas mozik, így egyszerre több filmet vetítenek. Végül a `város` sem kulcs, mert a városokban általában egynél több mozi van, és egynél több filmet játszanak.

Másrészt a három kétattribútumos részhalmazból kettő is kulcs. Világos, hogy a {`filmcím`, `város`} kulcs, mert a megadott függőség szerint ezek az attribútumok meghatározzák a `mozi`-t.

Ugyanígy igaz, hogy a {`mozi`, `filmcím`} is kulcs, mert a lezártja tartalmazza a `város`-t a `mozi` $\rightarrow$ `város` függőség következtében. A fennmaradó attribútumpár, a `város` és a `mozi` nem határozza meg funkcionálisan a `filmcím`-et a többvásznas mozik miatt, így nincs több kulcs. Összefoglalva, a két lehetséges kulcs:

{`filmcím`, `város`}
{`mozi`, `filmcím`}

[4] Ebben a példában feltesszük, hogy nincs egyszerre két „aktuális" film, amelyeknek azonos a címe, annak ellenére, hogy korábban elismertük, lehet két film azonos címmel, amelyek különböző évben készültek.

Azonnal látható a BCNF megsértése. Megadtunk egy funkcionális függőséget, a mozi → város-t, de a bal oldali mozi nem szuperkulcs. Ez arra csábít bennünket, hogy ezt a BCNF-et sértő függőséget felhasználva felbontsuk a relációt az alábbi két sémára:

{mozi, város}
{mozi, filmcím}

Ezzel a felbontással nincs semmi gond az alábbi függőséget illetően:

filmcím város → mozi

Lehetnek az aktuális relációk a felbontott sémákban olyanok, hogy kielégítik a mozi → város függőséget (amely ellenőrizhető a {mozi, város} relációban), de amikor elvégezzük az összekapcsolást, az olyan relációt eredményez, amely nem elégíti ki a filmcím város → mozi függőséget. Például az alábbi relációkról:

mozi	*város*
Guild	Menlo Park
Park	Menlo Park

és

mozi	*filmcím*
Guild	Z, a hangya
Park	Z, a hangya

azt gondolnánk, hogy megfelelnek a fenti relációra alkalmazottnak, de amikor összekapcsoljuk őket, két sort kapunk:

mozi	*város*	*filmcím*
Guild	Menlo Park	Z, a hangya
Park	Menlo Park	Z, a hangya

amelyek megsértik a filmcím város → mozi függőséget. □

3.4.5. Feladatok

3.4.1. feladat. Adott az $R(A, B, C, D, E)$ és relációkra való bontása a következő attribútumhalmazokkal: $\{A, B, C\}$, $\{B, C, D\}$ és $\{A, C, E\}$. Az alábbi függőséghalmazok mindegyikével végezzük el a chase-tesztet, hogy eldönthessük, veszteségmentes-e a dekompozíció. Azokra, amelyek nem veszteségmentesek, adjunk példát az R olyan előfordulására, melynél a dekompozíció és az újraösszekapcsolás az R-nél több sort eredményez.

a) $B \to E$ és $CE \to A$;

b) $AC \to E$ és $BC \to D$;

c) $A \to D$, $D \to E$ és $B \to D$;

d) $A \to D$, $CD \to E$ és $E \to D$.

! **3.4.2. feladat.** A 3.4.1. feladat összes függőséghalmazai közül melyiket őrzi meg a dekompozíció?

3.5. Harmadik normálforma

A 3.25. példában látott problémát megoldhatjuk úgy, hogy enyhítünk a BCNF feltételen annak érdekében, hogy alkalmanként megengedhető legyen, hogy ha egy relációsémát nem tudunk felbontani BCNF-relációkra, akkor ne végezzük el, vagy enyhítsük az FD-k ellenőrzését. Ennek az enyhítő feltételnek a neve „harmadik normálforma". Ebben a részben megadjuk a harmadik normálforma követelményeit, aztán megadunk egy algoritmust, amellyel elvégezhetjük a dekompozíciót, a 3.20. algoritmustól lényegesen eltérő módon annak érdekében, hogy megkapjuk a harmadik normálformát és megtartsuk mind a veszteségmentes összekapcsolást, mind pedig a függőségmegőrző tulajdonságokat.

3.5.1. A harmadik normálforma definíciója

Az R reláció *harmadik normálformá*ban (3NF) van akkor és csak akkor, ha:

- Valahányszor létezik R-ben egy $A_1A_2\cdots A_n \to B_1B_2\cdots B_m$ nem triviális függőség, akkor vagy az

$$\{A_1, A_2, \ldots, A_n\}$$

halmaz az R szuperkulcsa, vagy azokra az attribútumokra $B_1, B_2, \ldots, B_m$ közül, amelyek nincsenek az A-k között, teljesül, hogy egy kulcsnak az elemei (nem feltétlenül ugyanannak a kulcsnak).

Ha egy attribútum szerepel valamelyik kulcsban, akkor gyakran *elsődleges* attribútumnak (vagy más néven prímattribútumnak) nevezzük. Így a 3NF feltételét átfogalmazhatjuk: „minden nem triviális függőségre igaz, hogy bal oldala szuperkulcs, vagy jobb oldala csak elsődleges attribútumokat tartalmaz".

Megjegyezzük, hogy a 3NF feltétel és a BCNF feltétel közötti különbség a „vagy jobb oldala csak elsődleges attribútumokat tartalmaz" mondatrészben van, ami „felmenti" a `mozi` → `város`-hoz hasonló függőséget a 3.25. példában, mivel a jobb oldali `város` egy elsődleges attribútum.

Más normálformák

Ha van „harmadik normálforma", mi történt az első két „normálformával"? Ezeket valóban definiálták, de ma már keveset használjuk őket. Az *első normálforma* egyszerűen az a feltétel, hogy minden sor minden komponense atomi értékű legyen. A *második normálforma* kevésbé szigorú feltétel, mint a 3NF. Van „negyedik normálforma" is, ezzel később, a 3.6. alfejezetben találkozunk.

3.5.2. 3NF-szintetizáló algoritmus

Most kifejtjük és megindokoljuk, hogyan bontható fel egy R reláció olyan relációk halmazává, amelyekre teljesülnek az alábbiak:

a) A felbontásban szereplő relációk mindegyike 3NF-ben van.

b) A felbontásnak veszteségmentes összekapcsolása van.

c) A felbontásnak függőségmegőrző tulajdonsága van.

3.26. algoritmus. Harmadik normálformában lévő relációk előállítása veszteségmentes összekapcsolással és függőségmegőrzéssel

BEMENET: Egy R reláció és az R-re vonatkozó funkcionális függőségek F halmaza.

KIMENET: Az R felbontása relációk gyűjteményére, melyek mindegyike 3NF-ben van. A felbontás rendelkezik veszteségmentes összekapcsolással és függőségmegőrző tulajdonsággal.

ELJÁRÁS: Hajtsuk végre a következő lépéseket:

1. Keressük meg F egy minimális bázisát, amit hívjunk G-nek.
2. A G-ben szereplő összes $X \to A$ funkcionális függőségre használjuk XA-t sémaként a felbontás egy relációjához.
3. Amennyiben a 2. lépésben kapott relációk attribútumhalmazának egyike sem szuperkulcs R-ben, adjunk még egy relációt az eredményhez, melynek sémája kulcs lesz R-hez.

□

3.27. példa. Tekintsük az $R(A, B, C, D, E)$ relációt az $AB \to C$, $C \to B$ és $A \to D$ funkcionális függőségekkel. Első lépésként figyeljük meg, hogy a megadott funkcionális függőségek a minimális bázist jelentik. Ennek ellenőrzésére

egy kis munkát kell végeznünk. Először ellenőrizzük, hogy nem tudunk egyetlen függőséget sem elhagyni. Ennek megmutatására a 3.7. algoritmust használjuk, azaz funkcionális függőségek egyik párosa sem implikálja a harmadikat. Például vegyük az $\{A, B\}$ lezárását. Az első funkcionális függőség bal oldala csak a második és harmadik funkcionális függőségekben – azaz a $C \to B$-ben és $A \to D$-ben – kerül felhasználásra. Így a lezárás tartalmazza D-t, de nem tartalmazza C-t, így levonhatjuk azt a konklúziót, hogy az $AB \to C$ funkcionális függőséget nem implikálja a második és a harmadik funkcionális függőség. Hasonló konklúzióra jutunk, ha a második vagy a harmadik funkcionális függőséget próbáljuk elhagyni.

Azt is ellenőriznünk kell, hogy nem tudunk egyetlen attribútumot sem elhagyni a bal oldalról. Ebben az egyszerű esetben az egyetlen lehetőség az, hogy az első funkcionális függőségből az A-t vagy a B-t hagyjuk el. Például, ha az A-t hagyjuk el, akkor $B \to C$ marad. Azt kell megmutatni, hogy ez a $B \to C$ nem vezethető le a három eredeti funkcionális függőségből, amelyek az $AB \to C$, $C \to B$ és $A \to D$. Ezekkel a funkcionális függőségekkel a $\{B\}$ lezárása csak a B, így a $B \to C$ nem következik belőle. Így megvan a minimális bázisunk.

A 3NF előállítását azzal kezdjük, hogy vesszük az egyes funkcionális függőségek attribútumait, mint egy-egy sémát. Így az $S_1(A, B, C)$, $S_2(B, C)$ és $S_3(A, D)$ relációkat kapjuk. Sosem szükséges olyan relációt használnunk, amelynek sémája egy másik reláció részhalmaza, így az S_2 sémát eldobjuk.

Azt is meg kell vizsgálnunk, hogy szükséges-e egy olyan reláció hozzáadása, amelynek sémája egy kulcs. Ebben a példában R két kulccsal rendelkezik: $\{A, B, E\}$ és $\{A, C, E\}$, így ezeket tudjuk ellenőrizni. Ezen kulcsok egyike sem részhalmaza az eddig kiválasztott sémáknak. Így az egyiket hozzá kell adnunk, mondjuk az $S_4(A, B, E)$ sémát. Így R végső felbontása a következő: $S_1(A, B, C)$, $S_3(A, D)$ és $S_4(A, B, E)$. □

3.5.3. Miért működik a 3NF-szintetizáló algoritmus?

Három dolgot kell megmutatnunk: azt, hogy a veszteségmentes összekapcsolási, valamint a függőségmegőrző tulajdonságok megmaradtak, és azt, hogy a felbontás relációi 3NF-ben vannak.

1. *Veszteségmentes összekapcsolás.* Kezdjük egy olyan relációval, amely attribútumainak egy K halmaza szuperkulcs. Tekintsük a funkcionális függőségek szekvenciáját, ahogy azt a 3.7. algoritmusban használtuk a K kiterjesztéséhez K^+-ra. Mivel K egy szuperkulcs, így tudjuk, hogy K^+ az összes attribútumot jelenti. A funkcionális függőségeket a chase-tablóra alkalmazva azt kapjuk, hogy az indexelt szimbólumok a K-hoz tartozó sorban egyenlők lesznek a nem indexelt szimbólumokkal, ugyanabban a sorrendben, ahogyan az attribútumok a lezártba kerültek. Így ennek a tesztnek az eredményeként levonhatjuk azt a következtetést, hogy a felbontás veszteségmentes.

2. *Függőségmegőrzés.* A minimális bázis összes funkcionális függősége rendelkezik az összes attribútummal a felbontás valamelyik relációjában. Így az összes függőség ellenőrizhető a felbontás relációiban.

3. *Harmadik normálforma.* Amikor egy olyan relációt adunk hozzá a felbontáshoz, amelynek sémája egy kulcs, akkor ez a reláció biztosan 3NF-ben van. Ennek az az oka, hogy ennek a relációnak az összes attribútuma elsődleges kulcs, így nem sérti meg a 3NF-et erre a relációra. A minimális bázis funkcionális függőségeiből származó relációk esetében annak bizonyítása, hogy 3NF-ben vannak, túlmutat ennek a könyvnek a keretein. Az érvelés annak bemutatására szorítkozik, hogy a 3NF megsértése maga után vonná, hogy a bázis nem minimális.

3.5.4. Feladatok

3.5.1. feladat. Minden relációsémára és funkcionális függőséghalmazra, amely a 3.3.1. feladatban szerepelt:

i) Derítsük fel a 3NF-sértéseket.

ii) Ha szükséges, bontsuk fel a relációkat 3NF-ben lévő relációk halmazára.

3.5.2. feladat. Vizsgáljuk meg a `Kurzusok`(K, O, I, T, D, Cs) relációt, amelynek az attribútumai informálisan az alábbiak: kurzus, oktató, időpont, terem, diák, csoport. Legyenek a funkcionális függőségek: $K \to O$, $IT \to K$, $IO \to T$, $ID \to T$ és $KD \to Cs$. Intuitívan az első állítás azt mondja, hogy egy kurzusnak egyedi oktatója van, a második azt jelenti, hogy egy időpontban egy teremben egy kurzus lehet. A harmadik azt mondja, hogy egy tanár egy időpontban csak egy teremben lehet, a negyedik pedig ugyanezt mondja a diákokról. Az utolsó pedig azt jelenti, hogy egy diák egy kurzus egyetlen csoportjába jár.

a) Melyek a `Kurzusok` reláció kulcsai?

b) Igazoljuk, hogy a megadott funkcionális függőségek minimális bázist alkotnak.

c) Használjuk a 3NF-szintetizáló algoritmust, hogy megtaláljuk R veszteségmentes összekapcsolási és függőségmegőrző tulajdonságú dekompozícióit 3NF-relációkra. Van ezek között olyan reláció, amelyik nincs BCNF-ben?

3.5.3. feladat. Vizsgáljuk meg a `Részvények`(B, I, Be, R, M, O) relációt, melynek az attribútumai informálisan az alábbiak: bróker, iroda (a brókeré), befektető, részvény, mennyiség (a befektető által birtokolt részvényé) és osztalék (a részvényé). Legyenek a `Részvények`-re érvényes függőségek $R \to O$, $Be \to B$, $BeR \to M$ és $B \to I$. Ismételjük meg a 3.5.2. feladatot a `Részvények` relációra.

3.5.4. feladat. Igazoljuk a chase használatával, hogy a 3.27. példában látott dekompozíció veszteségmentes.

!! **3.5.5. feladat.** Tegyük fel, hogy módosítjuk a 3.20. algoritmust (BCNF-dekompozíció) úgy, hogy csak akkor bontjuk fel R-et, amikor nincs 3NF-ben, ahelyett hogy minden olyan esetben felbontanánk, amikor R nincs BCNF-ben. Adjunk ellenpéldát, amely igazolja, hogy a módosított algoritmus nem feltétlenül állítja elő a függőségmegőrző 3NF-felbontást.

3.6. Többértékű függőségek

A „többértékű függőség" olyan állítás, amely azt fejezi ki, hogy két attribútum vagy attribútumhalmaz független egymástól. Látjuk később, hogy ez a feltétel a funkcionális függőség fogalmának az általánosítása abban az értelemben, hogy minden funkcionális függőség implikálja a megfelelő többértékű függőséget. A független attribútumhalmazokkal kapcsolatosan vannak olyan helyzetek, amelyeket nem lehet a funkcionális függőségekkel megmagyarázni. Ebben az alfejezetben feltárjuk a többértékű függőség okait, és látni fogjuk, hogyan használhatjuk azokat az adatbázisséma tervezésében.

3.6.1. Attribútumfüggetlenségből származó redundancia

Előfordulhatnak olyan helyzetek, hogy olyan relációsémát tervezünk, amely bár BCNF-ben van, mégis marad benne egyfajta redundancia, amely nem a funkcionális függőséghez kapcsolódik. Leggyakrabban ez abból ered, hogy a kulcs két vagy több halmazértékű tulajdonságát is egyetlen relációval próbálunk reprezentálni.

3.28. példa. Ebben a példában feltesszük, hogy egy színésznek lehet több lakcíme is. A lakcímeket kétkomponensűnek tekintjük: a várost különválasztjuk az utca-házszámtól. A színészekhez és lakcímeikhez a szokásos attribútumokat is (a filmek címe és éve, amelyekben szerepelnek) hozzávesszük a relációhoz. A 3.10. ábra ennek a relációnak egy tipikus előfordulását mutatja.

név	*város*	*utca*	*filmcím*	*év*
C. Fisher	Hollywood	123 Maple St.	Csillagok háborúja	1977
C. Fisher	Malibu	5 Locust Ln.	Csillagok háborúja	1977
C. Fisher	Hollywood	123 Maple St.	A birodalom visszavág	1980
C. Fisher	Malibu	5 Locust Ln.	A birodalom visszavág	1980
C. Fisher	Hollywood	123 Maple St.	Jedi visszatér	1983
C. Fisher	Malibu	5 Locust Ln.	Jedi visszatér	1983

3.10. ábra. A lakcímek halmaza független a filmektől

A 3.10. ábra középpontjában Carrie Fisher két vélt lakcíme és a három legismertebb filmje áll. Nincs értelme összekapcsolni egy lakcímet egyik vagy másik filmmel. Így csak egyféleképpen tudjuk kifejezni, hogy a lakcímek és filmek a színészek független jellemzői, mégpedig minden lakcím előfordul minden filmmel. Ha pedig a lakcímeket és filmeket minden kombinációban megismételjük, akkor nyilvánvaló a redundancia. Például a 3.10. ábrán háromszor megismételjük Carrie Fisher lakcímét (minden filmjéhez külön-külön) és kétszer minden filmjét (minden lakcímhez egyszer).

A 3.10. ábrán javasolt reláció mégsem sérti meg a BCNF-et. Valójában egyáltalán nem is létezik nem triviális függőség. Például a `város` attribútumot nem határozza meg funkcionálisan a másik négy attribútum. Előfordulhat, hogy egy színésznek két lakása van különböző városokban, de ugyanolyan nevű utcában. Ekkor lenne két olyan sor, amely a `város`-on kívül minden más attribútumon megegyezne, a `város`-on pedig különbözne. Így a

$$\texttt{név utca filmcím év} \rightarrow \texttt{város}$$

funkcionális függőség nem érvényes a `Színészek` relációnkban. Az olvasóra bízzuk, hogy ellenőrizze a többi esetben is, hogy az öt attribútumból egyiket sem határozza meg funkcionálisan a többi négy. Mivel nincs nem triviális funkcionális függőség, ebből következik, hogy az egyetlen kulcsot az öt attribútum együttesen alkotja, és nem sértheti semmi a BCNF-et. □

3.6.2. Többértékű függőségek definíciója

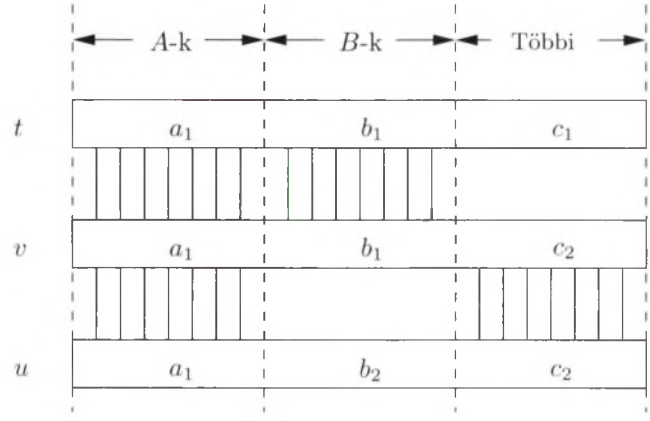
A *többértékű függőség* (angolul multivalued dependency, MVD-vel rövidítjük) valamely R relációra vonatkozó állítás, miszerint amikor rögzítjük egy attribútumhalmaz értékeit, akkor bizonyos más attribútumok értékei függetlenek lesznek a relációban szereplő összes többi attribútum értékeitől. Pontosabban kifejezve azt mondjuk, hogy az

$$A_1 A_2 \cdots A_n \twoheadrightarrow B_1 B_2 \cdots B_m$$

többértékű függőség fennáll R relációban, ha tekintve az R sorai közül azokat, amelyek minden egyes A-beli attribútumon ugyanazt a bizonyos értéket veszik fel, akkor ezekre a sorokra azt találjuk, hogy a B-ken felvett értékek halmaza független R-nek az A-któl és B-től eltérő összes attribútumán felvett értékek halmazától. Még pontosabban azt mondjuk, hogy ez a többértékű függőség érvényes, ha:

> Az R reláció minden olyan t és u sorpárjára, amelyek minden A-n megegyeznek, találhatunk R-ben olyan v sort, amely megegyezik:
>
> 1. t-vel és u-val az A-kon,
> 2. t-vel a B-ken és
> 3. u-val R minden olyan attribútumán, amely nincs az A-kban vagy a B-kben.

Megjegyezzük, hogy ebben a szabályban t-t és u-t felcserélhetjük, amelynek az a következménye, hogy létezik egy negyedik w sor, amely a B-ken megegyezik u-val és a többi eltérő attribútumokon pedig t-vel. Végeredményben az A-knak valamely rögzített értékeire a B-khez és a többi attribútumhoz rendelt értékeknek az összes lehetséges kombinációban elő kell fordulniuk a különböző sorokban. A 3.11. ábrában felvázoltuk, v hogyan viszonyul t-hez és u-hoz, amikor fennáll a többértékű függőség. Az A-k és a B-k nem feltétlenül egymást követően fordulnak elő az oszlopokban.



3.11. ábra. A többértékű függőség biztosítja v létezését

Általában feltehetjük, hogy egy többértékű függőségben az A-k és B-k (a függőség bal és jobb oldala) diszjunkt halmazok. Itt is megengedett persze, mint a funkcionális függőségeknél, hogy hozzávegyünk az A-ból néhány attribútumot a jobb oldalhoz, ha kívánjuk.

3.29. példa. A 3.28. példában láttunk egy többértékű függőséget, amelyet a jelöléseinkkel az alábbi formában tudunk kifejezni:

$$\texttt{név} \twoheadrightarrow \texttt{város utca}$$

Azaz minden színésznévre a lakcímek halmaza kapcsolatban van a színész mindegyik filmjével. Például nézzük meg, hogyan alkalmazhatjuk a többértékű függőség formális definícióját a 3.10. ábra első és negyedik sorára:

név	*város*	*utca*	*filmcím*	*év*
C. Fisher	Hollywood	123 Maple St.	Csillagok háborúja	1977
C. Fisher	Malibu	5 Locust Ln.	A birodalom visszavág	1980

Ha ezt a két sort ebben a sorrendben t-vel és u-val jelöljük, akkor a többértékű függőség azt mondja ki, hogy R-ben lennie kell olyan sornak, amelyben a név `C. Fisher`, a városon és utcán megegyezik az első sorral, t-vel, és az

egyéb attribútumain (`filmcím` és `év`) megegyezik az u-val. Valóban van ilyen sor, mégpedig a 3.10. ábra harmadik sora.

Hasonlóan vegyük most a fenti t sort másodiknak és u legyen az első. Ekkor a többértékű függőség azt mondja ki, hogy R-ben lennie kell olyan sornak, amely a `név`, `város`, `utca` attribútumokon megegyezik a másodikkal, és a `név`, `filmcím` és `év` attribútumokon megegyezik az elsővel. Ez a sor is létezik, méghozzá ez a 3.10. ábra második sora. □

3.6.3. Többértékű függőségekre vonatkozó szabályok

A 3.2. alfejezetben a funkcionális függőségekre tanult levezetési szabályokhoz hasonló számos szabály van a többértékű függőségekre is. Például a többértékű függőségre teljesülnek a következők:

- *Triviális többértékű függőség.* Az

$$A_1A_2\cdots A_n \twoheadrightarrow B_1B_2\cdots B_m$$

  többértékű triviális függőség fennáll bármely relációban, ha

$$\{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\}$$

  teljesül.

- A *tranzitivitási szabály*, amely szerint ha érvényesek

$$A_1A_2\cdots A_n \twoheadrightarrow B_1B_2\cdots B_m \quad \text{és} \quad B_1B_2\cdots B_m \twoheadrightarrow C_1C_2\cdots C_k$$

  többértékű függőségek egy bizonyos relációra, akkor

$$A_1A_2\cdots A_n \twoheadrightarrow C_1C_2\cdots C_k$$

  is érvényes. Mindazon C-ket, amelyek egyben A-k is, törölhetjük a jobb oldalról.

A többértékű függőség viszont nem tesz eleget a szétvághatósági/összevonhatósági szabály szétvághatóságra vonatkozó részének, mint ahogy a következő példa mutatja.

3.30. példa. Nézzük meg újból a 3.10. ábrát, amelyben korábban már láttuk, hogy teljesül az alábbi többértékű függőség:

$$\texttt{név} \twoheadrightarrow \texttt{város utca}$$

Ha a szétvághatósági szabály igaz lenne a többértékű függőségekre is, akkor azt várnánk, hogy

$$\texttt{név} \twoheadrightarrow \texttt{utca}$$

is teljesül. E többértékű függőség szerint minden színész lakcímében az utca független a többi attribútumtól, beleértve a város-t is. Azonban ez az állítás hamis. Tekintsük például a 3.10. ábra első két sorát. A feltételezett többértékű függőség arra engedne következtetni, hogy a sorok az utcákban váltakozhatnak, azaz a

név	*város*	*utca*	*filmcím*	*év*
C. Fisher	Hollywood	5 Locust Ln.	Csillagok háborúja	1977
C. Fisher	Malibu	123 Maple St.	Csillagok háborúja	1977

sorok is benne lennének a relációban. De ezek nem valódi sorok, hiszen például Locust Ln. 5 Malibuban van, nem pedig Hollywoodban. □

Vannak új szabályok is a többértékű függőségekre. Először:

- *Funkcionális függőség előléptetése.* Minden funkcionális függőség egyben többértékű függőség is. Azaz, amennyiben

$$A_1A_2\cdots A_n \to B_1B_2\cdots B_m$$

akkor

$$A_1A_2\cdots A_n \twoheadrightarrow B_1B_2\cdots B_m$$

A bizonyításhoz legyen R egy tetszőleges reláció, amelyben

$$A_1A_2\cdots A_n \to B_1B_2\cdots B_m$$

fennáll, és legyenek t és u az R-nek olyan sorai, amelyek az A-kon megegyeznek. Ahhoz, hogy megmutassuk $A_1A_2\cdots A_n \twoheadrightarrow B_1B_2\cdots B_m$ érvényességét, azt kell megmutatnunk, hogy R tartalmaz egy olyan v sort is, amely az A-kon megegyezik t-vel és u-val, a B-ken t-vel, a többi attribútumon pedig u-val. De v-nek vehetjük az u-t. Biztos, hogy az u megegyezik az A-kon t-vel és u-val, hiszen feltettük, hogy ez a két sor megegyezik az A-kon. Az $A_1A_2\cdots A_n \to B_1B_2\cdots B_m$ funkcionális függőség biztosítja, hogy u megegyezik a B-ken t-vel. Természetesen a többi attribútumon pedig u megegyezik önmagával. Így ha egy funkcionális függőség fennáll, akkor a megfelelő többértékű függőség is fennáll.

- *Komplementer-szabály.* Ha $A_1A_2\cdots A_n \twoheadrightarrow B_1B_2\cdots B_m$ többértékű függőség az R relációban, akkor R kielégíti az $A_1A_2\cdots A_n \twoheadrightarrow C_1C_2\cdots C_k$ többértékű függőséget is, ahol a C-k az R összes attribútumai közül éppen azok, amelyek nincsenek sem az A-k, sem a B-k között.

Ez azt jelenti, hogy a B-k közül két olyan sornak a cseréje, amelyek megegyeznek az A-kkal, ugyanolyan hatást vált ki, mint a C-k közötti csere.

3.31. példa. Nézzük meg újból a 3.10. ábrán látható relációt, amelyre feltettük, hogy teljesíti az alábbi többértékű függőséget:

$$\texttt{név} \twoheadrightarrow \texttt{város utca}$$

A komplementer-szabály szerint a

$$\texttt{név} \twoheadrightarrow \texttt{filmcím év}$$

többértékű függőségnek is teljesülnie kell ebben a relációban, mivel a `filmcím` és `év` attribútumok nem szerepeltek az első függőségben. A második függőség intuitívan azt jelenti, hogy minden színész esetén azoknak a filmeknek halmaza, amelyekben a színész játszott, független a színész lakcímétől. □

Az olyan többértékű függőség, amelynek jobb oldala a bal oldal egy részhalmaza, triviálisan fennáll minden reláció esetén. A komplementer szabály egy érdekes következménye azonban, hogy vannak egyéb olyan triviális többértékű függőségek is, amelyek nem néznek ki triviálisnak.

- *További triviális többértékű függőségek.* Amennyiben egy R reláció összes attribútuma az
$$\{A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m\}$$
halmaz, akkor $A_1A_2\cdots A_n \twoheadrightarrow B_1B_2\cdots B_m$ fennáll R-en.

Annak bizonyításához, hogy ezek a további triviális többértékű függőségek fennállnak, figyeljük meg, hogy ha veszünk két olyan sort, amelyek megegyeznek az $A_1, A_2, \ldots, A_n$ attribútumokon, és a komponenseiket felcseréljük a $B_1, B_2, \ldots, B_m$ attribútumokon, akkor ugyanazt a két sort kapjuk, bár ellenkező sorrendben.

3.6.4. Negyedik normálforma

Azokat a többértékű függőségek által okozott redundanciákat, amelyeket korábban a 3.6.1. alfejezetben találtunk, megszüntethetjük, ha ezeket a függőségeket felhasználjuk a felbontásokban. Ebben az alfejezetben bevezetünk egy új normálformát, az ún. „negyedik normálformát". Ebben a normálformában az összes „nem triviális" többértékű függőséget megszüntetjük úgy, ahogy tettük az összes olyan funkcionális függőséggel, amelyek megsértették a BCNF-et. Az eredményül kapott, felbontott relációkban nincs sem a funkcionális függőségekből származó olyan redundancia, mint amilyen a 3.3.1. alfejezetben szerepelt, sem a többértékű függőségekből származó redundancia, mint amilyent a 3.6.1. alfejezetben láttunk.

A „negyedik normálforma" feltétel lényegében olyan, mint a BCNF-feltétel, csak a funkcionális függőségek helyett többértékű függőségekre alkalmazzuk. Formálisan:

- Egy R reláció *negyedik normálformá*ban (4NF) van, ha valahányszor az
$$A_1A_2\cdots A_n \twoheadrightarrow B_1B_2\cdots B_m$$

nem triviális többértékű függőség fennáll, akkor $\{A_1, A_2, \ldots, A_n\}$ szuperkulcs.

Azaz, ha egy reláció 4NF-ben van, akkor minden nem triviális többértékű függőség valójában olyan funkcionális függőség, amelynek bal oldala szuperkulcs. Megjegyezzük, hogy a kulcs és szuperkulcs fogalmak csak a funkcionális függőségektől függnek, a többértékű függőségeket is hozzávéve nem változik a „kulcs" definíciója.

3.32. példa. A 3.10. ábra relációja megsérti a 4NF feltételt. Például, a

$$\texttt{név} \twoheadrightarrow \texttt{város utca}$$

nem triviális többértékű függőség, mégis a név önmagában nem szuperkulcs. Tulajdonképpen ehhez a relációhoz az egyetlen kulcs az összes attribútumból áll. □

A negyedik normálforma valóban a BCNF általánosítása. A 3.6.3. alfejezetben láttuk, hogy minden funkcionális függőség egyben többértékű függőség is. Így minden, ami megsérti a BCNF-et, megsérti a 4NF-et is. Másképpen fogalmazva, minden 4NF-ben levő reláció emiatt BCNF-ben van.

Vannak olyan BCNF-ben levő relációk, amelyek nem 4NF-beliek. A 3.10. ábra jó példa erre. Az egyetlen kulcs ehhez a relációhoz mind az öt attribútumból áll, és emiatt nincs nem triviális funkcionális függőség. Így ez biztos, hogy BCNF-ben van. Ugyanakkor a 3.32. példában láttuk, hogy nincs 4NF-ben.

3.6.5. Negyedik normálformára bontás

A 4NF dekompozíciós algoritmusa meglehetősen hasonlít a BCNF dekompozíciós algoritmusához.

3.33. algoritmus. Negyedik normálformára bontás

BEMENET: Az R_0 reláció és az S_0 funkcionális és többértékű függőségeket tartalmazó halmaz.

KIMENET: Az R_0 dekompozíciója olyan relációkra, amelyek mindegyike 4NF-ben van. A dekompozíció veszteségmentes összekapcsolási tulajdonsággal rendelkezik.

ELJÁRÁS: Végezzük el a következő lépéseket $R = R_0$-lal és $S = S_0$-lal:

1. Keressük a 4NF feltétel egy megsértését R-ben. Legyen ez

$$A_1 A_2 \cdots A_n \twoheadrightarrow B_1 B_2 \cdots B_m$$

ahol $\{A_1, A_2, \ldots, A_n\}$ nem szuperkulcs. Megjegyezzük, hogy ez a többértékű függőség lehet egy valódi többértékű függőség, vagy levezethető egy megfelelő $A_1 A_2 \cdots A_n \to B_1 B_2 \cdots B_m$ S-beli funkcionális függőségből, mivel tudjuk, hogy minden funkcionális függőség egyben többértékű függőség is. Ha nincs ilyen, akkor készen vagyunk, R önmagában egy megfelelő dekompozíció.

2. Ha találunk 4NF-sértést, akkor osszuk fel a 4NF feltételt megsértő R relációt két sémára:

   a) R_1, melynek a sémája tartalmazza az A-kat és a B-ket;

   b) R_2, melynek a sémája az A-kból és a sem A-ban, sem B-ben nem szereplő attribútumokból áll.

3. Keressük meg az R_1-ben és R_2-ben fennálló funkcionális és többértékű függőségeket (a 3.7. alfejezet bemutatja, hogyan kell ezt általános esetben végrehajtani, de gyakran a függőségek „vetítése" a legegyszerűbb). Rekurzívan dekomponáljuk R_1-et és R_2-t a vetített függőségeikkel.

□

3.34. példa. Folytassuk a 3.32. példát. Tapasztaltuk, hogy a

$$\texttt{név} \twoheadrightarrow \texttt{utca város}$$

a 4NF megsértése. A dekompozíciós szabály azt mondja, hogy helyettesítsük ezt az ötattribútumú sémát egy sémával, amely a fenti függőség attribútumait tartalmazza, és egy olyannal, amely tartalmazza a bal oldali `név` attribútumot és azokat, amelyek nem fordulnak elő a függőségben. Ezek az attribútumok a `filmcím` és az `év`, tehát a dekompozíció eredményeként a következő sémák adódnak:

$$\{\texttt{név, utca, város}\}$$
$$\{\texttt{név, filmcím, év}\}$$

Ezekben a sémákban nincs nem triviális többértékű (vagy funkcionális) függőség, tehát 4NF-ben vannak. Megjegyezzük, hogy a {`név, utca, város`} sémájú relációban a

$$\texttt{név} \twoheadrightarrow \texttt{utca város}$$

többértékű függőség triviális, mivel tartalmazza az összes attribútumot. Hasonlóan a {`név, filmcím, év`} relációban a

$$\texttt{név} \twoheadrightarrow \texttt{filmcím év}$$

többértékű függőség triviális. Ha egyik vagy mindkét kapott reláció nem 4NF-ben lenne, akkor folytatnunk kellene a nem 4NF-ben lévő sémá(k) felbontásával. □

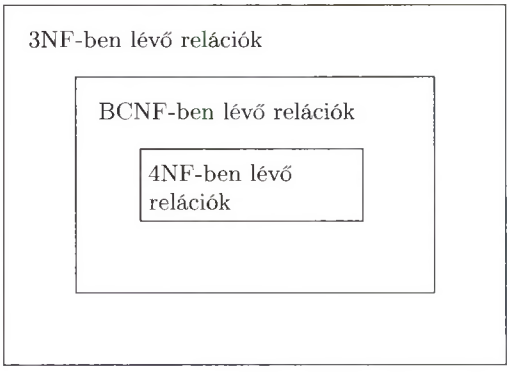
Ahogy a BCNF-felbontásnál, a felbontás minden egyes lépésében egyre kevesebb attributumból álló sémát kapunk, mint amilyenből kiindultunk. Ily módon végül olyan sémákhoz jutunk, melyeket már nem kell tovább felbontani, mivel már 4NF-ben vannak. A 3.4.1. alfejezetben adott felbontás helyességének igazolása átvihető a többértékű függőségekre is. Amikor felbontunk egy relációt az

$A_1A_2 \cdots A_n \twoheadrightarrow B_1B_2 \cdots B_m$ többértékű függőség miatt, ez a függőség elegendő ahhoz, hogy az eredeti reláció a felbontásban szereplő relációkból visszaállítható.

A 3.7. alfejezetben megadunk egy olyan algoritmust, melyet a többértékű függőségre alkalmazva 4NF-dekompozíciót szolgáltat, és igazolja, hogy a dekompozició veszteségmentes. Ugyanezen alfejezetben megmutatjuk ennek végrehajthatóságát, megjegyezve azonban, hogy ez időigényes, mert az MVD-knek a dekompozícióval kapott relációkra való vetítését kell végrehajtani. Ez a vetítés ahhoz szükséges, hogy eldöntsük, vajon szükséges-e további felbontás.

3.6.6. Normálformák közötti kapcsolatok

Említettük korábban, hogy a 4NF-ből következik a BCNF, amelyből következik a 3NF. A 3.12. ábrán szemléltetjük, hogy a normálformákat kielégítő függőségekkel rendelkező relációsémák halmazai között mi a kapcsolat. Ha egy reláció adott függőségekkel 4NF-ben van, akkor az egyben BCNF-ben és 3NF-ben is van. Ha egy reláció BCNF-ben van, akkor 3NF-ben is van.



3.12. ábra. A 4NF-ből következik a BCNF, amiből következik a 3NF

Jellemzői	*3NF*	*BCNF*	*4NF*
Megszünteti a funkcionális függőségekből eredő redundanciát	Nem	Igen	Igen
Megszünteti a többértékű függőségekből eredő redundanciát	Nem	Nem	Igen
Megőrzi a funkcionális függőségeket	Igen	Nem	Nem
Megőrzi a többértékű függőségeket	Nem	Nem	Nem

3.13. ábra. A normálformák és felbontásaik jellemzői

A normálformák összehasonlításának másik módja az, hogy az adott normálformára való felbontás eredményéül kapott relációk halmaza mit biztosít. Ezt a 3.13. ábrán összegeztük. A BCNF (és így a 4NF) megszünteti a redundanciát és más problémákat, amelyeket a funkcionális függőségek okoznak, ugyanakkor csak a 4NF szünteti meg azt a további redundanciát, amelyet olyan többértékű függőségek okoznak, amelyek nem funkcionális függőségek. Gyakran elegendő a 3NF ilyen redundancia megszüntetésére, de vannak olyan esetek, amikor nem. A BCNF nem biztosítja a funkcionális függőségek megőrzését, és egyik normálforma sem biztosítja a többértékű függőségek megőrzését, ám tipikus esetekben a függőségek is érvényben maradnak.

3.6.7. Feladatok

3.6.1. feladat. Legyen $R(A, B, C)$ egy reláció az $A \twoheadrightarrow B$ többértékű függőséggel. Ha tudjuk, hogy az (a, b_1, c_1), (a, b_2, c_2) és (a, b_3, c_3) sorok benne vannak R aktuális előfordulásában, akkor milyen további soroknak kell még R-ben lenniük?

3.6.2. feladat. Vegyük azt a relációt, amelyben feljegyezzük a személyek nevét, személyi számát és születési dátumát. Továbbá a személy összes gyerekének a nevét, személyi számát és születési dátumát, és amennyiben autóval vagy autókkal rendelkezik a személy, az autóinak a sorozatszámát és gyártmányát. Pontosabban ennek a relációnak a sorai az összes olyan

$$(n, szsz, szd, gyn, gyszsz, gyszd, as, agy)$$

sor, amelyre

1. n a személy neve, akinek a személyi száma $szsz$;
2. szd az n születési dátuma;
3. gyn az n egyik gyerekének neve;
4. $gysz$ a gyn személyi száma;
5. $gyszd$ a gyn születési dátuma;
6. as az n egyik autójának sorozatszáma;
7. agy az as sorozatszámú autó gyártmánya.

Erre a relációra:

a) Adjuk meg azokat a funkcionális és többértékű függőségeket, amelyektől elvárjuk, hogy érvényesek legyenek.

b) Adjunk egy javaslatot a reláció 4NF-ben való felbontására.

3.6.3. feladat. A következő relációsémákra és funkcionális függőségi halmazokra nézve:

a) $R(A, B, C, D)$ sémára és $A \twoheadrightarrow B$, $A \twoheadrightarrow C$ többértékű függőségekkel;

b) $R(A, B, C, D)$ sémára és $A \twoheadrightarrow B$, $B \twoheadrightarrow CD$ többértékű függőségekkel;

c) $R(A, B, C, D)$ sémára, $AB \twoheadrightarrow C$ többértékű függőséggel és $B \to D$ funkcionális függőséggel;

d) $R(A, B, C, D, E)$ sémára és $A \twoheadrightarrow B$ és $AB \twoheadrightarrow C$ többértékű függőségekkel, továbbá $A \to D$ és $AB \to E$ funkcionális függőségekkel

végezzük el az alábbiakat:

i) Keressük meg az összes 4NF-et megsértő függőséget.

ii) Bontsuk fel a fenti relációsémákat 4NF-ben lévő relációsémákra.

3.6.4. feladat. Adjunk informális érveket arra, miért nem vártuk a 3.28. példa öt attribútumának egyikétől sem, hogy a másik négy funkcionálisan meghatározza.

3.7. Egy algoritmus többértékű függőségek megkeresésére

Az érvelések, melyek többértékű függőségekre vagy többértékű függőségek és funkcionális függőségek együttesére vonatkoznak, sokkal bonyolultabbak, mint azok, amelyek csak funkcionális függőségekre vonatkoznak. Funkcionális függőségekre a 3.7. algoritmust használhatjuk, és eldönthetjük vele, hogy egy függőség következik-e több megadott függőségből vagy nem. Ebben az alfejezetben először megmutatjuk, hogy a lezárási algoritmus valójában azonos a chase-eljárással, amelyet a 3.4.2. alfejezetben tanulmányoztunk. A chase mögötti ötlet kiterjeszthető többértékű függőségekre úgy, ahogyan a funkcionális függőségekre. Lesz egy eszközünk, amivel megoldhatjuk az összes olyan problémát, amely a többértékű függőségekkel és a funkcionális függőségekkel kapcsolatos, mint például annak eldöntése, hogy egy többértékű függőség következik-e megadott függőségekből, vagy mi lesz a többértékű és funkcionális függőségek vetülete a dekompozícióval kapott relációkra.

3.7.1. A chase és a lezárás

A 3.2.4. alfejezetben láttuk, hogyan kell egy adott X attribútumhalmazból kiszámítani a lezártját, X^+-t, amely az összes olyan attribútumot tartalmazza, amely funkcionálisan függ X-től. Így tesztelhetjük, hogy egy $X \to Y$ funkcionális függőség következik funkcionális függőségek egy F halmazából, az X lezárásával F szerint, illetve megnézhetjük, hogy $Y \subseteq X^+$ teljesül-e. Tekinthetjük

a lezárást a chase egy variációjával, ahol a kiinduló tabló és a cél különböznek attól, amit a 3.4.2. alfejezetben láttunk.

Tegyük fel, hogy egy olyan tablóval kezdünk, amelynek két sora van. Ezek a sorok megegyeznek X attribútumain és eltérnek az összes többi attribútumon. Ha alkalmazzuk az F-beli funkcionális függőségeket, akkor pontosan azokban az oszlopokban fogjuk egyenlővé tenni a szimbólumokat, amelyek $X^+ - X$-ben vannak. Így a chase-alapú teszt annak eldöntésére, hogy $X \to Y$ következik-e F-ből, az alábbiakban foglalható össze:

1. Kezdjük egy olyan tablóval, aminek két olyan sora van, amely megegyezik X attribútumain és különbözik a többin.

2. Hajtsuk végre a chase-eljárást az F függőséghalmazt használva.

3. Ha a végső tabló megegyezik Y összes oszlopán, akkor $X \to Y$ fennáll, különben pedig nem.

3.35. példa. Ismételjük meg a 3.8. példát, ahol az alábbi relációnk volt:

$$R(A, B, C, D, E, F)$$

az $AB \to C$, $BC \to AD$, $D \to E$ és $CF \to B$ függőségekkel. Azt szeretnénk eldönteni, hogy $AB \to D$ fennáll-e. Induljunk ki ebből a tablóból:

A	B	C	D	E	F
a	b	c_1	d_1	e_1	f_1
a	b	c_2	d_2	e_2	f_2

Alkalmazhatjuk az $AB \to C$-t $c_1 = c_2$ bizonyítására; legyen mondjuk mindkettő c_1. Az eredménytabló:

A	B	C	D	E	F
a	b	c_1	d_1	e_1	f_1
a	b	c_1	d_2	e_2	f_2

Következőnek alkalmazzuk a $BC \to AD$-t $d_1 = d_2$ bizonyítására, majd utána $D \to E$-t, hogy belássuk: $e_1 = e_2$. Ezen a ponton a tabló:

A	B	C	D	E	F
a	b	c_1	d_1	e_1	f_1
a	b	c_1	d_1	e_1	f_2

Innen nem tudunk továbblépni. Mivel a két sor már megegyezik a D oszlopon, tudjuk, hogy $AB \to D$ következik a megadott függőségekből. □

3.7.2. A chase kiterjesztése többértékű függőségekre

A chase során funkcionális függőségeknél alkalmazott eljárás alkalmazható többértékű függőségek bizonyítására is. Amikor megpróbálunk belátni egy funkcionális függőséget, azt kérdezzük, hogy következtethetünk-e arra, hogy két nem feltétlenül azonos értéknek valójában meg kell egyeznie. Amikor alkalmazzuk az $X \to Y$ funkcionális függőséget, olyan sorpárokat keresünk a tablóban, amelyek megegyeznek X összes oszlopán, és arra törekszünk, hogy Y minden oszlopában az értékek megegyezőek legyenek.

Azonban a többértékű függőségek nem jelentik azt, hogy bizonyos szimbólumok azonosak. $X \to\to Y$ inkább azt állítja, hogy ha találunk két olyan sort a tablóban, amelyek megegyeznek X-en, akkor készíthetünk két újabb sort, megcserélve az Y attribútumokon elhelyezkedő komponenseiket; a kapott két új sornak a relációban szerepelnie kell, ezáltal a tablóban is. Hasonlóan, ha funkcionális és többértékű függőségekből le szeretnénk vezetni egy $X \to\to Y$ többértékű függőséget, egy kétsoros tablóval kell kezdenünk, amely két olyan sort tartalmaz, amely megegyezik az X attribútumokon és különbözik azokon, amelyek X-en kívüliek. Alkalmazzuk a megadott funkcionális függőségeket, hogy egyenlővé tegyünk szimbólumokat, majd alkalmazzuk a többértékű függőségeket is, azaz megcseréljük az értékeket bizonyos attribútumokon a két előforduló sorban annak érdekében, hogy új sorokat adjunk a tablóhoz. Ha észrevesszük, hogy az eredeti sorok egyikében az Y attribútumokat kicseréljük egy másik eredeti sorból ugyanazokkal, akkor beláttuk a függőséget.

Eljött a figyelmeztetés helye ebben a lényegesen összetettebb chase-eljárásban. Mivel szimbólumokat egyenlővé tudunk tenni, és ki tudjuk cserélni más szimbólumokkal, nem biztos, hogy felismerjük, hogy elkészítettünk egyet a kívánt sorból, mert néhány eredeti szimbólumot esetleg újakra cseréltünk. A legegyszerűbb út ennek a problémának a megelőzésére, ha a célsort az inicializáláskor definiáljuk, és később nem változtatjuk a szimbólumait. Azaz legyen a célsor egy olyan sor, amelynek minden egyes komponense egy nem indexelt betű. Legyen két kezdősor az $X \to\to Y$ függőség teszteléséhez, amely nem indexelt betűket tartalmaz X attribútumain. Legyen az első sor olyan, hogy nincs indexük az Y attribútumain és legyen a második olyan, hogy nem indexelt betűk vannak az X-en és Y-on kívüli összes attribútumon. A két sorban fennmaradó helyeket töltsük ki új szimbólumokkal, amelyek mindegyike csak egyszer fordul elő. Amikor egyenlővé teszünk indexelt és nem indexelt szimbólumokat, akkor az indexelt helyére tegyük a nem indexeltet, ahogyan a 3.4.2. alfejezetben is tettük. Ezután, ha alkalmazzuk a chase-eljárást, csak azt a kérdést kell megválaszolnunk, hogy előfordul-e az a sor a tablóban, amelynek minden eleme indexeletlen.

3.36. példa. Tegyük fel, hogy adott az $R(A, B, C, D)$ reláció és $A \to B$, $B \to\to C$ függőségek. Azt szeretnénk belátni, hogy $A \to\to C$ fennáll R-ben. Kezdjük egy kétsoros tablóval, amely $A \to\to C$-t reprezentálja:

A	B	C	D
a	b_1	c	d_1
a	b	c_2	d

Megjegyezzük, hogy a célsorunk az (a, b, c, d). A tabló mindkét sora az A attribútumon index nélküli elemet tartalmaz. Az első sorban index nélküli elem van a C oszlopban, míg a második sorban a fennmaradó helyeken.

Először az $A \to B$ függőséget fogjuk alkalmazni annak bizonyítására, hogy $b = b_1$. Tehát le kell cserélnünk az indexelt b_1-et az index nélküli b-re. A keletkező tabló:

A	B	C	D
a	b	c	d_1
a	b	c_2	d

Következőnek a $B \twoheadrightarrow C$ többértékű függőséget alkalmazzuk, mivel a két sor megegyezik a B oszlopon. A C oszlopokat cserélgetjük, hogy új sorokat adhassunk a tablóhoz, ami az alábbihoz vezet:

A	B	C	D
a	b	c	d_1
a	b	c_2	d
a	b	c_2	d_1
a	b	c	d

Ezzel megkapjuk a csupa index nélküli szimbólumokból álló sort, ami bizonyítja, hogy $A \twoheadrightarrow C$ fennáll az R relációban. Nézzük meg, hogy a tablómanipulációk tényleg bizonyítják, hogy $A \twoheadrightarrow C$ fennáll. A bizonyítás a következő: „Adott két sor, amelyek megegyeznek A-n. $A \to B$ miatt meg kell egyezniük B-n is. Mivel megegyeznek B-n, a $B \twoheadrightarrow C$ miatt megcserélhetjük a C-komponenseiket, és a kapott sorok R-beliek lesznek. Azaz, ha két sor megegyezik A-n, akkor azok a sorok, amelyeket a C attribútumok megcserélésével kapunk, szintén R-beliek lesznek, azaz $A \twoheadrightarrow C$." □

3.37. példa. A funkcionális és többértékű függőségekkel kapcsolatosan van egy meglepő szabály, amely azt mondja ki, hogy ha az $X \twoheadrightarrow Y$ többértékű függőség fennáll, továbbá van egy olyan funkcionális függőség, amelynek a jobb oldala Y nem feltétlenül teljes részhalmaza, amit nevezzünk Z-nek, akkor $X \to Z$ is fennáll. A chase-eljárást fogjuk használni arra, hogy egy egyszerű példáját bizonyítsuk ennek a szabálynak. Legyen a megadott relációnk $R(A, B, C, D)$ az $A \twoheadrightarrow BC$ többértékű függőséggel és a $D \to C$ funkcionális függőséggel. Azt állítjuk, hogy $A \to C$.

Mivel egy funkcionális függőséget próbálunk belátni, az index nélküli elemeket tartalmazó célrekord elérésével nem kell foglalkoznunk. Bármely két olyan rekordból kiindulhatunk, amelyek megegyeznek A-n, és különböznek az összes többi összetevőben. Például:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2

A célunk annak belátása, hogy $c_1 = c_2$.

Az egyetlen dolog, amivel kezdhetünk, hogy alkalmazzuk az $A \twoheadrightarrow BC$ többértékű függőséget, mivel a két sor megegyezik A-n, de semmilyen más oszlopon nem. Amikor megcseréljük a B és C attribútumokat ebben a két sorban, két újabb sort kell felvennünk:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2
a	b_2	c_2	d_1
a	b_1	c_1	d_2

Most olyan párokat kaptunk, amelyek megegyeznek a D oszlopban, tehát alkalmazhatjuk a $D \rightarrow C$ funkcionális függőséget. Például az első és a harmadik sornak is d_1 a D-értéke, tehát alkalmazhatjuk a függőséget $c_1 = c_2$ levezetésére. Ez volt a célunk, azaz bizonyítani, hogy $A \rightarrow C$ fennáll. A tabló az alábbi:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_1	d_2
a	b_2	c_1	d_1
a	b_1	c_1	d_2

Végül semmilyen más módosítás nem lehetséges a megadott függőségekkel. Ugyanakkor ez nem is probléma, mert a szükséges bizonyítást már elvégeztük. □

3.7.3. Miért működik a chase a többértékű függőségekre?

Az okoskodás lényegében most is ugyanaz, mint amit az előbb láttunk. A chase minden lépése, legyen az szimbólumok egyenlővé tétele vagy új sorok generálása, egy igaz észrevétel az R sorairól, amelyet az adott lépésben alkalmazott funkcionális vagy többértékű függőséggel tudunk igazolni. Azaz a chase pozitív eredménye mindig bizonyítja, hogy a megadott funkcionális és többértékű függőségek fennállnak R-ben.

Amikor a chase hibával végződik – azaz a célsor (többértékű függőségeknél) vagy az elvárt egyenlőségek (funkcionális függőségeknél) nem állnak elő –, akkor a befejező lépéskor létrejött tabló egy ellenpélda. Kielégíti a megadott függőségeket, különben tudtunk volna még változtatásokat végrehajtani. Ugyanakkor nem elégíti ki a belátni próbált függőséget.

Van egy másik probléma is, ami nem jött elő akkor, amikor csak funkcionális függőségekkel használtuk a chase-t. Amióta a többértékű függőségek új sorokat

adnak a tablóhoz, honnan tudhatjuk, hogy a chase-eljárás be fog fejeződni? Adhatunk örökké hozzá új sorokat úgy, hogy soha nem érjük el a célunkat, de nem lehetünk biztosak abban, hogy még néhány lépés múlva elérhetjük? Szerencsére ez nem történhet meg. Ez azért van így, mert sosem készítünk új szimbólumokat. Induláskor legfeljebb két különböző szimbólumunk van a k oszlop mindegyikében. Azaz sosem lehet 2^k-nál több sorunk a tablóban, ha k az oszlopok száma. A chase többértékű függőségekkel eltarthat exponenciális ideig, de sosem fut a végtelenségig.

3.7.4. Többértékű függőségek vetítése

Idézzük fel, hogy a többértékű függőségek bizonyítása azért kellett, mert ez vezetett el a relációk 4NF-felbontásához. A feladat befejezéséhez képesnek kell lennünk arra, hogy a megadott függőségeket levetítsük a két relációsémára, amelyet a dekompozíció első lépésében kapunk. Csak ekkor tudhatjuk meg, hogy a relációk 4NF-ben vannak, vagy esetleg újabb dekompozícióra van szükség.

Legrosszabb esetben ki kell próbálnunk minden lehetséges funkcionális és többértékű függőséget a felbontott relációkra. A chase-tesztet alkalmaztuk az eredeti reláció összes attribútumán. Mivel a cél egy többértékű függőség esetében egy olyan sor előállítása a tablóban, amely index nélküli betűket tartalmaz a dekompozíció egyik relációjának összes oszlopában, ebben a sorban a többi helyen bármilyen szimbólum szerepelhet. A funkcionális függőség esetében a cél ugyanez: a szimbólumok egyezése egy megadott oszlopban.

3.38. példa. Adott az $R(A, B, C, D, E)$ reláció, amit felbontunk, és az egyik, dekompozícióval létrejövő reláció legyen $S(A, B, C)$. Tegyük fel, hogy az $A \twoheadrightarrow CD$ függőség fennáll R-ben. Implikál ez a függőség bármilyen függőséget S-ben? Azt állítjuk, hogy $A \twoheadrightarrow C$ fennáll S-ben csakúgy, mint $A \twoheadrightarrow B$ is (a komplementer-szabály miatt). Lássuk be, hogy $A \twoheadrightarrow C$ fennáll S-ben. Az alábbi tablóval kezdünk:

A	B	C	D	E
a	b_1	c	d_1	e_1
a	b	c_2	d	e

Alkalmazzuk az $A \twoheadrightarrow CD$ függőséget, és cseréljük meg a C és D komponenseket ebben a két sorban, hogy két új sort kapjunk:

A	B	C	D	E
a	b_1	c	d_1	e_1
a	b	c_2	d	e
a	b_1	c_2	d	e_1
a	b	c	d_1	e

Megjegyezzük, hogy az utolsó sor az S attribútumain, azaz A-n, B-n és C-n, csak indexeletlen szimbólumokat tartalmaz. Ez elegendő ahhoz, hogy belássuk, $A \twoheadrightarrow C$ fennáll S-re. □

Gyakran a funkcionális és többértékű függőségek megkeresésének a vetületrelációkban nem kell teljesen kimerítőnek lennie. Alább láthatunk néhány egyszerűsítést.

1. Biztosan nem kell ellenőrizni a triviális funkcionális és többértékű függőségeket.

2. Funkcionális függőségek esetében nem kell azokra szorítkoznunk, amelyeknek egy attribútuma van a jobb oldalon, a kombinációs szabály miatt.

3. Egy funkcionális vagy egy többértékű függőség, amelynek bal oldala nem tartalmazza egyetlen adott függőség bal oldalát sem, biztosan nem áll fenn, mivel sehogyan sem tudjuk elkezdeni vele a chase-eljárást. Azaz a két sor, melyekkel a tesztet kezdenénk, nem változik a megadott függőségekkel.

3.7.5. Feladatok

3.7.1. feladat. Használjuk a chase-tesztet annak eldöntésére, hogy az alábbi függőségek fennállnak-e az $R(A, B, C, D, E)$ reláción az $A \twoheadrightarrow BC$, $B \rightarrow D$ és $C \twoheadrightarrow E$ függőségek mellett?

a) $A \rightarrow D$

b) $A \twoheadrightarrow D$

c) $A \rightarrow E$

d) $A \twoheadrightarrow E$

! **3.7.2. feladat.** Ha a 3.7.1. feladatban látható R relációt levetítjük az $S(A, C, E)$-re, akkor mely nem triviális funkcionális és többértékű függőségek érvényesek S-ben?

! **3.7.3. feladat.** Lássuk be a következő szabályokat a többértékű függőségekre. Minden egyes esetben a bizonyítás megadható a chase-teszttel, de egy kicsit általánosabban kell gondolkozni, mint a látott példákban, mivel az X, Y, Z tetszőleges attribútumhalmazok, továbbá vannak egyéb, név nélküli attribútumok a relációban, amelyekre a függőségek érvényesek.

a) Az unió szabálya. Ha X, Y és Z attribútumhalmazok, továbbá $X \twoheadrightarrow Y$ és $X \twoheadrightarrow Z$, akkor $X \twoheadrightarrow (Y \cup Z)$.

b) A metszet szabálya. Ha X, Y és Z attribútumhalmazok, továbbá $X \twoheadrightarrow Y$ és $X \twoheadrightarrow Z$, akkor $X \twoheadrightarrow (Y \cap Z)$.

c) A különbség szabálya. Ha X, Y és Z attribútumhalmazok, továbbá $X \twoheadrightarrow Y$ és $X \twoheadrightarrow Z$, akkor $X \twoheadrightarrow (Y - Z)$.

d) A közös attribútumok elhagyása a jobb oldalról. Ha $X \twoheadrightarrow Y$ fennáll, akkor $X \twoheadrightarrow (Y - X)$ is fennáll.

! **3.7.4. feladat.** Adjunk ellenpéldaként olyan relációkat, amelyek megmutatják, hogy a következő szabályok a többértékű függőségekre *nem* igazak. *Útmutatás:* alkalmazzuk a chase-tesztet, és figyeljük meg, mi történik.

a) Ha $A \twoheadrightarrow BC$, akkor $A \twoheadrightarrow B$.

b) Ha $A \twoheadrightarrow B$, akkor $A \rightarrow B$.

c) Ha $AB \twoheadrightarrow C$, akkor $A \twoheadrightarrow C$.

3.8. Összefoglalás

- *Funkcionális függőségek:* A funkcionális függőség egy állítás, amely azt mondja, hogy ha egy reláció két sora megegyezik néhány megadott attribútumon, akkor megegyezik attribútumok egy másik halmazán is.

- *Relációk kulcsa:* Egy reláció szuperkulcsa egy olyan attribútumhalmaz, amely funkcionálisan meghatározza a reláció többi attribútumát. A kulcs olyan szuperkulcs, melynek egyik valódi részhalmaza sem szuperkulcs.

- *Funkcionális függőségek bizonyítása:* Számos olyan szabály van, amellyel beláthatjuk, hogy egy $X \rightarrow A$ funkcionális függőség fennáll minden olyan előfordulásban, amely megfelel bizonyos funkcionális függőségeknek. Az $X \rightarrow A$ igazolásához számoljuk ki X lezártját úgy, hogy a funkcionális függőségek felhasználásával addig bővítjük X-et, amíg nem tartalmazza A-t.

- *Minimális bázis funkcionális függőségek egy halmazához:* Bármely függőséghalmazhoz létezik legalább egy minimális bázis, amely az eredetivel ekvivalens funkcionális függőségekből álló halmaz (azaz egyik halmazból következik a másik). A függőségek jobb oldalain egyetlen attribútum áll, egyetlen szabály sem hagyható el az ekvivalencia megtartásával, és a bal oldalakról sem hagyható el egyetlen attribútum sem az ekvivalencia megtartása mellett.

- *Boyce–Codd normálforma:* Egy reláció BCNF-ben van, ha a nem triviális függőségek csak olyanok, melyekben egy szuperkulcs meghatároz egy vagy több más attribútumot. A BCNF legnagyobb előnye, hogy kiküszöböli azokat a redundanciákat, amelyeket a funkcionális függőségek jelenléte okoz.

- *Veszteségmentes összekapcsolással rendelkező dekompozíció:* A dekompozíció egy másik hasznos tulajdonsága, hogy az eredeti reláció a felbontásban szereplő relációk természetes összekapcsolásával előállítható. Bármely

dekompozíció visszaad legalább annyi sort, amennyivel elindultunk, de a gondatlanul elvégzett dekompozíció olyanokat is képes előállítani, amelyek nem voltak az eredeti relációban.

- *Függőségmegőrző dekompozíció:* A dekompozíció egy másik elvárt tulajdonsága, hogy minden, az eredeti relációban fennálló függőséget ellenőrizhessünk a felbontott relációkban fennálló függőségek ellenőrzésével.

- *Harmadik normálforma:* Néha a BCNF-dekompozíció elveszíti a függőségmegőrző tulajdonságát. A BCNF egy változata, a 3NF megenged olyan $X \rightarrow A$ függőségeket akkor is, ha X nem szuperkulcs, feltéve, hogy A egy kulcs része. A 3NF nem garantálja, hogy megszünteti a funkcionális függőségekből adódó redundanciát, de többnyire megteszi.

- *A chase:* Eldönthetjük, hogy egy dekompozíciónak megvan-e a veszteségmentes összekapcsolási tulajdonsága, ha kitöltünk egy táblázatot (tablót) – ami azon sorok halmaza, amelyek az eredeti reláció sorait reprezentálják. A táblázat kitöltését a fennálló funkcionális függőségek alkalmazásával végezzük úgy, hogy belátjuk egyes szimbólumpárok egyenlőségét. A dekompozíció veszteségmentes a megadott függőséghalmaz mellett akkor és csak akkor, ha a chase egy olyan sorhoz vezet, amely azonos azzal a sorral, amelynek a létezését feltettük a felbontás utáni relációk összekapcsolásakor.

- *3NF-szintetizáló algoritmus:* Ha vesszük a funkcionális függőségek egy minimális bázisát, beletesszük mindet egy relációba, és hozzáadjuk a kulcsokat, ha szükséges, akkor a kapott eredmény a 3NF-re bontás, amelynek megvan a veszteségmentes összekapcsolási és a függőségmegőrző tulajdonsága is.

- *Többértékű függőségek:* A többértékű függőség egy olyan állítás, amely azt mondja, hogy két attribútumhalmaz a relációban olyan értékeket tartalmaz, amelyek az összes lehetséges kombinációban előfordulnak.

- *Negyedik normálforma:* A többértékű függőségek szintén okozhatnak redundanciát a relációban. A 4NF hasonló a BCNF-hez, de kiküszöböli azokat a nem triviális többértékű függőségeket, amelyek bal oldala nem szuperkulcs. A relációk 4NF-re bonthatók anélkül, hogy információt veszítenénk.

- *Többértékű függőségek bizonyítása:* Funkcionális és többértékű függőségeket bizonyíthatunk függőségek egy halmaza alapján a chase-eljárással. Egy kétsoros tablóból indulunk ki, amely reprezentálja azt a függőséget, amelyet bizonyítani próbálunk. A funkcionális függőségeket szimbólumok egyenlővé tételére alkalmazzuk, míg a többértékű függőségeket a tablóhoz új sorok hozzáadására, amelyekben a megfelelő komponensek felcserélve szerepelnek.

3.9. Irodalomjegyzék

A harmadik normálforma leírását [6] adja. Ez az írás bemutatja a funkcionális függőségek ötletét csakúgy, mint a kiinduló relációs elképzelést. A Boyce–Codd normálforma egy későbbi írásban ([7]) jelenik meg.

A többértékű függőségeket és a negyedik normálformát Fagin definiálta [9]-ben. Ugyanakkor a többértékű függőségek ötlete ettől függetlenül előfordult [8]-ban és [11]-ben.

Armstrong tanulmányozta először a funkcionális függőségek levezetési szabályait [2]-ben. Az itt tárgyalt funkcionális függőségekre vonatkozó szabályokat (beleértve az „Armstrong-axiómák" névvel illetetteket is) és a többértékű függőségek bizonyításához szükséges szabályokat [3]-ban találjuk.

A funkcionális függőségek tesztelése egy attribútumhalmaz lezárásának kiszámításával [4]-ből származik csakúgy, mint az, hogy a minimális bázisból levezethető a 3NF-dekompozíció. Az, hogy ez a dekompozíció veszteségmentesen összekapcsolható és függőségmegőrző tulajdonságú, [5]-ből való.

A tablóteszt a veszteségmentes összekapcsolhatósági teszthez és a chase [1]-ben szerepelnek. További információkat és az ötlet történetét [10] tartalmazza.

[1] A. V. Aho, C. Beeri, J. D. Ullman, „The theory of joins in relational databases," *ACM Transactions on Database Systems* **4**:3, pp. 297–314, 1979.

[2] W. W. Armstrong, „Dependency structures of database relationships," *Proceedings of the 1974 IFIP Congress*, pp. 580–583.

[3] C. Beeri, R. Fagin, J. H. Howard, „A complete axiomatization for functional and multivalued dependencies," *ACM SIGMOD Intl. Conf. on Management of Data*, pp. 47–61, 1977.

[4] P. A. Bernstein, „Synthesizing third normal form relations from functional dependencies," *ACM Transactions on Database Systems* **1**:4, pp. 277–298, 1976.

[5] J. Biskup, U. Dayal, P. A. Bernstein, „Synthesizing independent database schemas," *ACM SIGMOD Intl. Conf. on Management of Data*, pp. 143–152, 1979.

[6] E. F. Codd, „A relational model for large shared data banks," *Comm. ACM* **13**:6, pp. 377–387, 1970.

[7] E. F. Codd, „Further normalization of the data base relational model," in *Database Systems* (R. Rustin, ed.), Prentice-Hall, Englewood Cliffs, NJ, 1972.

[8] C. Delobel, „Normalization and hierarchical dependencies in the relational data model," *ACM Transactions on Database Systems* **3**:3, pp. 201–222, 1978.

[9] R. Fagin, „Multivalued dependencies and a new normal form for relational databases," *ACM Transactions on Database Systems* **2**:3, pp. 262–278, 1977.

[10] J. D. Ullman, *Principles of Database and Knowledge-Base Systems, Volume I*, Computer Science Press, New York, 1988.

[11] C. Zaniolo, M. A. Melkanoff, „On the design of relational database schemata," *ACM Transactions on Database Systems* **6**:1, pp. 1–47, 1981.

4. fejezet

Magas szintű adatbázismodellek

Vizsgáljuk meg azt a folyamatot, ahogy egy új adatbázist – mint például a filmadatbázisunkat – létrehozunk. Ezt a folyamatot a 4.1. ábra szemlélteti. A tervezési fázissal kezdjük, amelyben a következő kérdéseket tesszük fel és válaszoljuk meg: milyen információkat kell tárolni, mely információelemek kapcsolódnak egymáshoz, milyen megszorításokat (például kulcs vagy a hivatkozási épség megszorítása) kell figyelembe venni és így tovább. Ebben a fázisban kell a lehetőségeket értékelni, az elképzeléseket összeegyeztetni – ez sokáig tarthat. A 4.1. ábra azt a folyamatot mutatja be, ahogy az elgondolásokat magas szintű tervvé alakítjuk.

4.1. ábra. Az adatbázis-modellezés és implementálás eljárása

Mivel a kereskedelmi adatbázisrendszerek nagy többsége a relációs modellt használja, élhetünk azzal a feltevéssel, hogy a tervezési fázisban ezt a modellt is figyelembe vegyük. A gyakorlatban azt tapasztaljuk, hogy gyakran egyszerűbb, ha a magas szintű modell megalkotásával kezdjük, majd ezt alakítjuk át a relációs modell tervévé. Annak, hogy így járjunk el, az elsődleges oka az, hogy a relációs modell csak egyetlen – a reláció – fogalmát tartalmazza, és nem foglalkozik egy sor másik, a valós világhoz sokkal közelebb álló modellel. A relációs koncepció egyszerűsége a modell nagy erőssége, amely az adatbázis-műveletek megvalósításának hatékonyságában jelenik meg. Ez az erősség az előzetes tervezéskor viszont gyengeséggé válik, és ez az, amiért általában hasznos, ha a magas szintű tervezéssel kezdjük.

A tervezési módszerek megnevezésében gyakran utalnak az alkalmazott módszerre is. Az első és legrégebbi módszer az „egyed-kapcsolat diagram", a 4.1. alfejezetben mi is ezzel kezdjük a módszerek bemutatását. Az aktuális irányzat az UML (Unified Modeling Language – Egységesített Modellező Nyelv) használata, melyet eredetileg az objektumorientált szoftvertervezéshez szántak, de az adatbázissémák leírására is alkalmassá tették. Ezt a modellt a 4.7. alfejezetben fogjuk bemutatni. Végül a 4.9. alfejezetben az ODL-t (Object Description Language – Objektum Leíró Nyelv) tanulmányozzuk, amely az adatbázisoknak osztályok és ezek objektumai összességéből álló rendszerként való leírására alkalmas.

A 4.1. ábrán látható következő fázis a magas szintű terv átalakítása relációs tervvé. Erre a lépésre csak akkor kerül sor, ha a magas szintű tervet már elfogadtuk. Bármelyik magas szintű modellt is használtuk, teljesen mechanikus lehetőségünk van arra, hogy a magas szintű tervet olyan relációs adatbázissémává alakítsuk, amely a kereskedelmi adatbázisrendszerekben alkalmazható lesz. Az E/K-diagramok relációs adatbázissémává való átalakításával a 4.5. és 4.6. alfejezetekben foglalkozunk. Az UML átalakítását a 4.8., az ODL használatát a 4.10. alfejezetekben tárgyaljuk.

4.1. Az egyed-kapcsolat (E/K) modell elemei

Az *E/K-modell*ben az adatok szerkezetét grafikusan ábrázoljuk, ún. egyed-kapcsolat diagramként. Három alapvető eleme lehet egy ilyen diagramnak:

1. Egyedhalmazok,
2. Attribútumok,
3. Kapcsolatok.

A következőkben ezeket részletezzük.

4.1.1. Egyedhalmazok

Az *egyed* valamilyen típusú absztrakt objektum. Hasonló egyedek összessége *egyedhalmaz*t alkot. Az egyed fogalma bizonyos értelemben hasonlít az objektumorientált programozás objektumfogalmához; hasonló viszony mondható el egy egyedhalmaz és objektumok egy osztálya között. Ugyanakkor az E/K-modell statikus fogalom, mely csupán az adatok szerkezetéről szól, a rajtuk végezhető műveletekről nem. Így tehát az osztályoktól eltérően, az egyedhalmazokhoz nem tartoznak metódusok.

4.1. példa. Példaként tekintsük a filmekről szóló példaadatbázisunkat. Minden film egy-egy egyed, és a filmek összessége egyedhalmazt alkot. Hasonlóan a színészek is egyedek, összességük egyedhalmazt alkot. Más típusú egyed a stúdió, és így a stúdiók halmaza lesz a harmadik egyedhalmaz, mely a további példákban szerepelni fog. □

Az E/K-modell változatai

Az E/K egyes változataiban az attribútumok típusai lehetnek:

1. atomiak, mint ahogy az itt bemutatott változatban,
2. rekordok, adott számú atomi komponensből felépítve (a C nyelvű „struct"-hoz hasonlóan),
3. atomi vagy rekord típusú elemekből álló halmazok, adott elemtípussal.

Például egy attribútum típusa lehetne párok halmaza, melyek első eleme egész szám, második eleme karaktersorozat.

4.1.2. Attribútumok

Az egyedhalmazokhoz *attribútumok* tartoznak, melyek az egyedek tulajdonságait írják le. A *Filmek* egyedhalmaz attribútuma lehet például a filmek *címe* vagy percekben mért *hossza*. Abban semmi meglepő nincs, hogy a *Filmek* egyedhalmaz attribútumai hasonlítanak a `Filmek` reláció attribútumaihoz. Az egyedhalmazokat általában relációkkal valósítjuk meg, azonban a végső relációs terv nem mindegyik relációja származik egyedhalmazból.

Az E/K-modell itt bemutatott változatában az attribútumok értékeit atomiaknak tekintjük (például karaktersorozatok, egész, illetve lebegőpontos számok). Az E/K-modell egyes változataiban korlátozott mértékben lehetőség van arra, hogy az attribútumoknak belső struktúrát adjunk meg (lásd az ide vonatkozó keretes írást).

4.1.3. Kapcsolatok

A *kapcsolatok* két vagy több egyedhalmazt kötnek össze egymással. Például a *Filmek* és a *Színészek* egyedhalmazok között felírható a *SzerepelBenne* kapcsolat: ha egy m filmben egy s színész szerepel, akkor az m filmegyed az s színészegyeddel részt vesz a *SzerepelBenne* kapcsolatban. A kapcsolatok leggyakrabban binárisak, azaz két egyedhalmazt kötnek össze. Az E/K-modell azonban lehetővé teszi, hogy tetszőleges számú egyedhalmazt kapcsoljunk össze egyetlen kapcsolattal. Az ilyen, többágú kapcsolatokkal majd a 4.1.7. alfejezetben foglalkozunk.

4.1.4. Egyed-kapcsolat diagramok

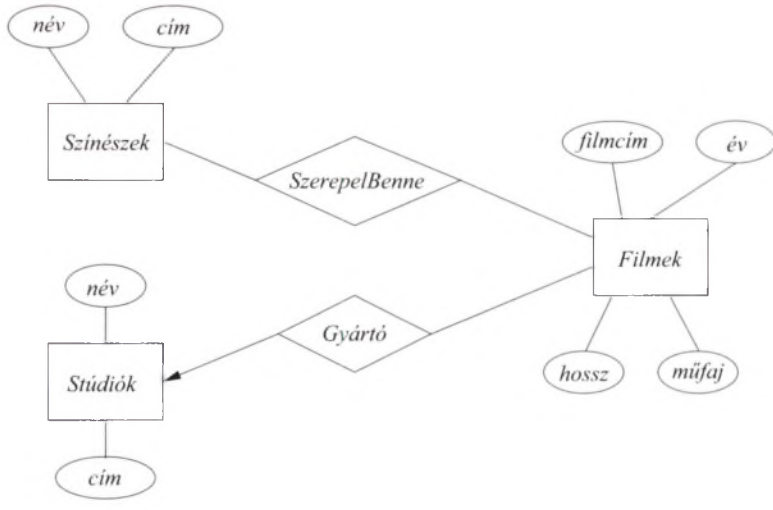
Az *egyed-kapcsolat diagram* egy gráf, melynek csúcspontjai egyedhalmazoknak, attribútumoknak és kapcsolatoknak felelnek meg. E különböző típusú elemeket

különféle alakzatokkal ábrázoljuk, az alábbiak szerint:

- az egyedhalmazt téglalappal,
- az attribútumot oválissal,
- a kapcsolatot pedig rombusszal jelöljük.

Az egyedhalmazokat összekötjük attribútumaikkal, a kapcsolatokat pedig a bennük részt vevő egyedhalmazokkal. Ezek az összekötések a gráf élei.

4.2. példa. A 4.2. ábrán egy egyszerű, filmekről szóló adatbázis E/K-diagramja látható. Az egyedhalmazok: *Filmek*, *Színészek*, *Stúdiók*.



4.2. ábra. A filmadatbázis egyed-kapcsolat diagramja

A *Filmek* egyedhalmaznak négy attribútuma van: *filmcím*, *év* (készítés ideje), *hossz*, *műfaj*. A másik két egyedhalmaz (*Színészek* és *Stúdiók*) két-két attribútuma megegyezik: *név* és *cím*, jelentésük értelemszerű. Ezenkívül két kapcsolat is látható a diagramon:

1. A *SzerepelBenne* kapcsolat hozzárendeli a filmekhez a bennük szereplő színészeket. Másik irányból tekintve egy-egy színészhez rendeli hozzá azokat a filmeket, melyekben az illető szerepel.

2. A *Gyártó* kapcsolat az egyes filmeket összekapcsolja a gyártójukkal. A 4.2. ábrán a *Stúdiók* egyedhalmaz irányába mutató nyíl azt fejezi ki, hogy minden filmhez csak egy gyártó stúdió tartozik. Az ehhez hasonló egyértelműségi megszorítást a 4.1.6. alfejezetben tárgyaljuk.

□

4.1.5. Az E/K-diagram előfordulásai

Az egyed-kapcsolat diagram az adatbázis *sémá*jának, azaz szerkezetének jelölésére szolgál. Egy E/K-diagram által leírt adatbázist aktuális adataival együtt az adatbázis *előfordulás*ának nevezzük. Minthogy az E/K-diagram az adatbázisnak csak a terve, nem pedig a megvalósítása, így az előfordulás nem egy létező dolog abban az értelemben, mint ahogy az adatbázisrendszerekben a relációk előfordulása létezik. Gyakran mégis hasznos, ha úgy ábrázoljuk, mintha létező dolog lenne.

Egy előfordulásban például egy adott egyedhalmazra nézve konkrét egyedek véges halmaza szerepel. Az egyedek mindegyike konkrét értékkel rendelkezik az összes attribútumon. Egy n ágú R kapcsolatnak, mely az $E_1, E_2, \ldots, E_n$ egyedhalmazokat köti össze, az előfordulásban $(e_1, e_2, \ldots, e_n)$ sorok véges halmaza felel meg, melyekben minden e_i-t a neki megfelelő E_i egyedhalmaz aktuális előfordulásából választunk ki. Minden egyes ilyen sort úgy értünk, hogy R „összekapcsolja" a benne szereplő egyedeket.

E sorok halmaza az R aktuális előfordulásának *kapcsolathalmaz*a. Gyakran segít, ha a kapcsolathalmazt táblázatként ábrázoljuk. A táblázat oszlopait a kapcsolatban részt vevő egyedhalmazok neveivel címkézzük meg, és a táblázat egy-egy sorát az előfordulásban szereplő egy-egy sor alkotja. Látjuk majd, hogy amikor a kapcsolatot relációvá alakítjuk, akkor az eredményreláció nem pontosan az, mint a kapcsolathalmaz.

4.3. példa. A *SzerepelBenne* kapcsolat egy előfordulását alkotó párok, táblázatos formában például az alábbiak szerint ábrázolhatók:

Filmek	*Színészek*
Elemi ösztön	Sharon Stone
Emlékmás	Arnold Schwarzenegger
Emlékmás	Sharon Stone

A kapcsolathalmaz elemei a táblázat sorai. Például a (Elemi ösztön, Sharon Stone) pár a *SzerepelBenne* kapcsolat aktuális előfordulásának egy eleme. □

4.1.6. Bináris E/K-kapcsolatok típusai

Általában egy bináris kapcsolat az egyik egyedhalmaz egy eleméhez tetszőleges számú egyedet kapcsolhat a másik egyedhalmazból. Ugyanakkor erre a „számosságra" (multiplicitásra) vonatkozóan gyakran teszünk megszorítást. Tegyük fel, hogy R egy kapcsolat az E és F egyedhalmazok között. Ekkor:

- Ha E minden eleme R-en keresztül legfeljebb egy F-beli elemhez kapcsolódhat, akkor R-et *sok-egy* kapcsolatnak nevezzük E-ből F-be. Vegyük észre, hogy ez esetben F egyedei tetszőleges számú E-beli egyedhez kapcsolódhatnak. Hasonlóan, ha F minden egyede R-en keresztül legfeljebb egy E-beli egyedhez kapcsolódhat, akkor R sok-egy kapcsolat F-ből E-be (más szóval *egy-sok* kapcsolat E-ből F-be).

- Ha R sok-egy kapcsolat egyszerre E-ből F-be és F-ből E-be is, akkor R-et *egy-egy* kapcsolatnak nevezzük. Az egy-egy kapcsolat bármely egyedhalmazának egy eleme a másik egyedhalmaz legfeljebb egy eleméhez kapcsolódhat.
- Ha R nem sok-egy kapcsolat E-ből F-be és F-ből E-be sem az, akkor R-et *sok-sok* kapcsolatnak nevezzük.

Mint a 4.2. példában említettük, a kapcsolatok típusát nyilak használatával jelölhetjük az E/K-diagramban. Ha egy kapcsolat sok-egy típusú az E egyedhalmaz és az F egyedhalmaz között, akkor egy nyíl mutat az F-be. A nyíl azt jelenti, hogy minden E-beli egyedhez legfeljebb egy F-beli egyed kapcsolódik. Ha E felé viszont nem nyíl, hanem egyszerű vonal vezet, akkor egy F-beli egyedhez sok E egyed kapcsolódhat.

4.4. példa. A fentiek értelmében, ha egy-egy kapcsolat van E és F egyedhalmazok között, akkor nyíl mutat mindkét egyedhalmazba. Például a 4.3. ábrán látható két egyedhalmaz a *Stúdiók* és *Elnökök*, illetve az *Irányít* kapcsolat (az attribútumokat elhagytuk). Feltételezhetjük, hogy egy elnök csak egy stúdiót vezet és egy stúdiónak csak egy elnöke van, így ez egy-egy kapcsolat, amelyet a két nyíl jelez.

4.3. ábra. Példa egy-egy kapcsolatra

Emlékeztetünk, hogy a nyíl „legfeljebb egy"-et jelent, azaz nem garantálja, hogy minden egyed kapcsolatban is áll a másik egyedhalmaz valamelyikével, amerre a nyíl mutat. A 4.3. ábrán például egy elnök bizonyára mindig hozzá van rendelve egy stúdióhoz, mivel máskülönben nem lehetne elnök. Fordított irányban azonban lehetséges, hogy egy stúdiónak egy adott időszakban nincs elnöke, így az *Irányítótól* az *Elnökök* felé mutató nyíl valóban „legfeljebb egy"-et jelent, nem pedig „pontosan egy"-et. Ezt a különbséget részletesebben a 4.3.3. alfejezetben tárgyaljuk. □

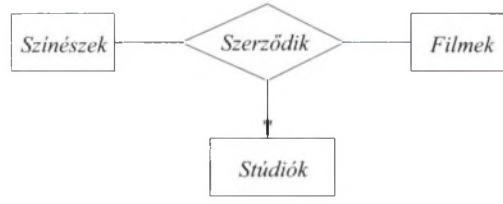
4.1.7. Sokágú kapcsolatok

Az E/K-modellben lehet olyan kapcsolatokat definiálni, amelyek több egyedhalmazt kapcsolnak össze. A gyakorlatban a hármas és magasabb fokú kapcsolatok ritkák, bár néha szükségesek ahhoz, hogy kifejezzük a dolgok valódi állását. A sokágú kapcsolatokat az E/K-diagramban a kapcsolatot reprezentáló rombuszból kiinduló és a kapcsolatban részt vevő egyedekhez vivő vonalakkal jelöljük.

4.5. példa. A 4.4. ábrán a *Szerződik* kapcsolatban részt vesznek a stúdiók, a színészek és a filmek. Ez a kapcsolat reprezentálja, hogy egy stúdió mely színésszel, melyik filmre kötött szerződést. Mint korábban a 4.1.5. alfejezetben

Következtetési lehetőségek kapcsolattípusok viszonya alapján

Figyeljük meg, hogy a sok-egy kapcsolat speciális esete a sok-sok kapcsolatnak, és az egy-egy kapcsolat speciális esete a sok-egy kapcsolatnak. Azaz a sok-sok kapcsolatok hasznos tulajdonságaival rendelkeznek a sok-egy kapcsolatok, és a sok-egy kapcsolatok hasznos tulajdonságaival pedig rendelkeznek az egy-egy kapcsolatok. Például a sok-egy kapcsolatokhoz alkalmazott adatstruktúra ugyanúgy működik az egy-egy kapcsolatoknál, habár ugyanez nem biztos, hogy működik a sok-sok kapcsolatok esetén.



4.4. ábra. Egy háromágú kapcsolat

tárgyaltuk, egy E/K-kapcsolat tekinthető egy kapcsolathalmaznak, amelynek az elemei a részt vevő egyedhalmazok egyedeiből képzett n-esek. Így a *Szerződik* kapcsolat a következő alakú hármasokkal írható le: (stúdió, színész, film).

A sokágú kapcsolatokban, ha egy nyíl egy E egyedhalmazra mutat, akkor az azt jelenti, hogy ha kiválasztunk a többi egyedhalmazból egy-egy egyedet, akkor a kapcsolatban ezekhez az egyedekhez legföljebb egy egyed tartozik az E egyedhalmazból. (Megjegyezzük, hogy ez a szabály általánosítja a kétágú kapcsolatoknál bevezetett jelölést.) Ezt funkcionális függőségnek is tekinthetjük, a függőség jobb oldalán E, bal oldalán az összes többi egyedhalmazok állnak.

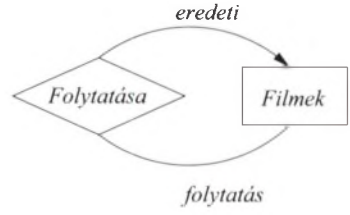
A 4.4. ábrán a *Stúdiók* egyedhalmazra mutat egy nyíl, ami azt jelenti, hogy csak egy stúdió létezik, amely egy színésszel egy adott filmre szerződést kötött. Azonban nem mutat nyíl a *Színészek* egyedhalmazra vagy a *Filmek* egyedhalmazra sem. Azaz egy stúdió szerződést köthet több színésszel is egy filmre, illetve egy színésszel több filmre is szerződhet. □

4.1.8. Szerepek a kapcsolatokban

Lehetséges, hogy egy egyedhalmaz kétszer vagy többször is szerepel egy kapcsolatban. Ha így van, akkor annyi vonalnak kell jönnie a kapcsolatból az egyedhalmazhoz, ahányszor az részt vesz a kapcsolatban. Minden vonal az egyedhalmaz egy másik *szerepét* mutatja a kapcsolatban. Ezért meg kell címkézni a kapcsolat és az egyedhalmaz közötti éleket, ezeket nevezzük „szerepeknek".

Korlátozások a nyíl jelölésre a sokágú kapcsolatokban

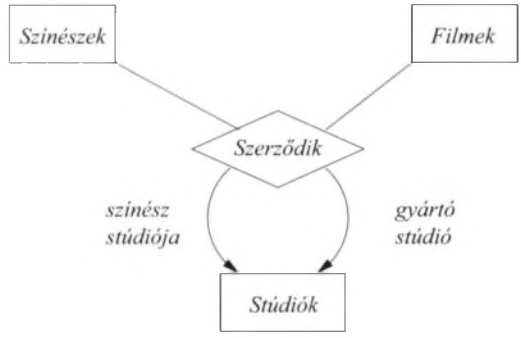
Nincs elegendő lehetőség a nyíl és nem nyíl választásban a három vagy több résztvevős kapcsolatok esetében. Így nem tudjuk leírni az összes lehetőséget nyilakkal. Például, a 4.4. ábrán a stúdió valójában csak a filmnek függvénye, és nem a színésznek és a filmnek együtt, mivel csak egy stúdió gyárt egy adott filmet. Azonban a jelölésünkben ezt nem tudjuk megkülönböztetni. Egy háromágú kapcsolat esetén az egyik egyedhalmazra mutató nyíl azt jelenti, hogy ennek az egyedhalmaznak az egyedei a másik két egyedhalmaz egyedeinek függvényei. Amennyiben minden lehetséges szituációt kezelni akarunk, funkcionális függőségek halmazát kell megadnunk a kapcsolatban részt vevő egyedhalmazok között.



4.5. ábra. Egy kapcsolat szerepekkel

4.6. példa. A 4.5. ábrán a *Folytatása* kapcsolat a *Filmek* egyedhalmaznak önmagával való kapcsolata. Két film között lehet kapcsolat úgy, hogy az egyik folytatása a másiknak. A kapcsolatban álló két film megkülönböztetésére az éleket a szerepekkel megcímkézzük. Az egyik, amely az *Eredeti* szerepben van és az eredeti filmet mutatja, a másik, amely *Folytatás* szerepben van és a folytatást reprezentálja. Feltesszük, hogy egy filmnek lehet több folytatása, de minden folytatásnak csak egy eredetije van. Így kapunk a 4.5. ábrán egy sok-egy kapcsolatot. □

4.7. példa. Záró példánkban egyaránt szerepel sokágú kapcsolat, és egy egyedhalmaz több szerepben. A 4.6. ábrán a 4.5. példában bevezetett *Szerződik* kapcsolat egy bonyolultabb verziója látható. Most a *Szerződik* kapcsolatban kétszer szerepel a stúdió, egyszer a színész és egyszer a film. Ez azt reprezentálja, hogy egy stúdió, amelyiknek van egy színésszel szerződése (általánosságban, nem egy adott filmre vonatkozóan), megengedi, hogy a színésze szerződést kössön egy másik stúdióval egy filmre. Így a kapcsolat a következő alakú négyesekkel írható le: (stúdió1, stúdió2, színész, film), amely azt jelenti, hogy a stúdió2 szerződést köt a stúdió1-gyel arra, hogy stúdió2 szerepelteti a stúdió1 színészét az általa gyártott filmben.



4.6. ábra. Egy négyágú kapcsolat

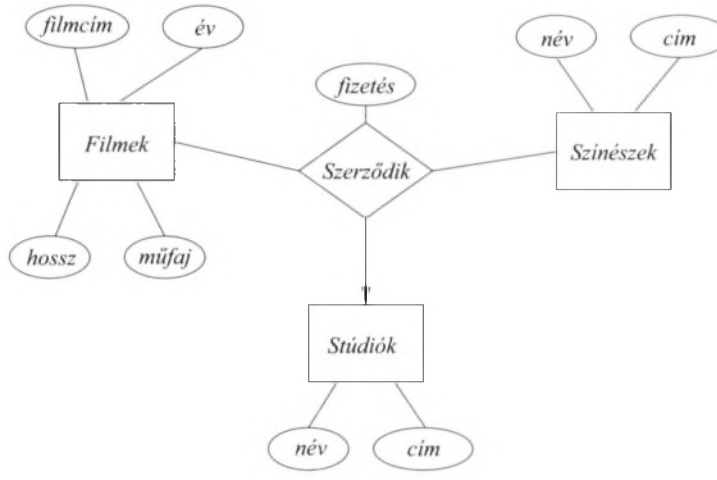
A 4.6. ábrán láthatjuk, hogy a nyilak mindkét szerep esetében a *Stúdiók* egyedhalmazra mutatnak, akkor is, amikor a színész „tulajdonosa", és akkor is, amikor filmgyártó szerepben van a *Stúdiók* egyedhalmaz. Azonban továbbra sincsenek nyilak a *Színészek* és a *Filmek* egyedhalmazokra. Az ok a következő: Ha adott egy színész, egy film és egy stúdió, amely a filmet készíti, akkor csak egy stúdió lehet, amely a „tulajdonosa" a színésznek. (Feltesszük, hogy egy színész pontosan egy stúdióval kötött szerződést.) Hasonlóan csak egy stúdió lehet a gyártó, ha adott a színész, a film és a színész stúdiója. Megjegyezzük, hogy mindkét esetben, a többi egyedek közül egy is elegendő ahhoz, hogy egyértelműen meghatározzuk a stúdió egyedet. Például a gyártó stúdió meghatározásához elegendő a filmet ismerni. Ez azonban nem változtat a sokágú kapcsolat specifikációján.

Tehát a *Színészek* és a *Filmek* egyedhalmazokra nem mutatnak nyilak. Ha adott egy színész, a stúdiója és a gyártó stúdió, akkor ezekhez több szerződés is tartozhat más-más filmekkel. Így ez a három komponens nem határozza meg egyértelműen a filmet. Hasonlóan egy gyártó stúdió szerződést köthet egy másik stúdióval több színészre is egy adott filmhez. Így a színészt sem határozza meg egyértelműen a másik három komponens. □

4.1.9. Kapcsolatok attribútumai

Néha kényelmes, sőt elengedhetetlen, ha attribútumokat rendelhetünk egy kapcsolathoz, és nem a benne részt vevő valamelyik egyedhalmazhoz. Például vizsgáljuk meg a 4.4. ábrát, amelyen modelleztük a színészek és a stúdiók szerződéseit valamilyen filmre.[1] Szeretnénk nyilvántartani az adott szerződéshez tartozó fizetést is. Azonban ez nem rendelhető a színészhez, mert a színész valószínűleg különböző filmek esetében különböző összeget kap. Hasonlóan, a stúdióhoz sem rendelhető (különböző színészeknek valószínűleg különböző összeget fizet) és a

[1] Itt visszatérünk a 4.5. példában szereplő, háromágú *Szerződik* kapcsolatra a 4.7. példa négyágú változata helyett.



4.7. ábra. Egy kapcsolat attribútummal

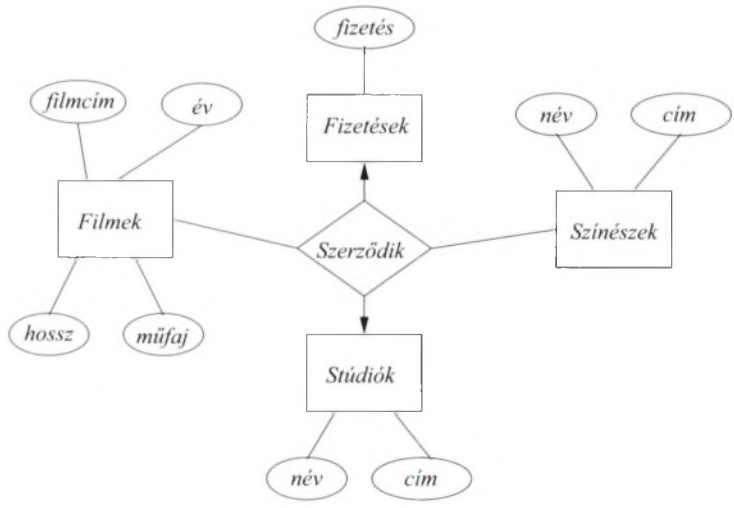
filmhez sem rendelhető (különböző színészek egy adott filmben is különböző fizetést kaphatnak).

Azonban, ha a fizetést a (színész, film, stúdió) hármashoz rendeljük a *Szerződik* kapcsolat kapcsolathalmazában, akkor az éppen jó lesz. A 4.7. ábra ugyanaz, mint a 4.4. ábra, kiegészítve attribútumokkal. A kapcsolatnak van *fizetés* attribútuma, az egyedhalmazoknak pedig ugyanazok az attribútumai, mint a 4.2. ábrán.

Általában bármely kapcsolathoz attribútumot vagy attribútumokat rendelhetünk. Ezen attribútumok értékét a kapcsolathalmazhoz rendelt reláció sora funkcionálisan meghatározza. Egyes esetekben az attribútumok a relációban szereplő egyedhalmazok egy részhalmaza által meghatározottak lehetnek, de feltehetően egyik egyszerű egyedhalmaz által sem meghatározott (vagy az attribútumnak az adott egyedhalmazhoz való rendelése nagyobb zavart okozna). A 4.7. ábrán például a fizetés valójában a film és a színész egyedek által meghatározott, a stúdió egyedet pedig a film határozza meg.

Nem szükséges a kapcsolatokhoz attribútumokat rendelni. Helyette felvehetünk egy új egyedhalmazt, amely egyedeinek attribútumai éppen a kapcsolathoz rendelt attribútumok. Ha ezt az egyedhalmazt is belevesszük a kapcsolatba, akkor elhagyhatjuk a kapcsolat attribútumait. Ugyanakkor hasznos konvenció a kapcsolatoknak attribútumokat adni, ezért ezt a jövőben is fogjuk alkalmazni, ahol indokolt.

4.8. példa. Tekintsük a 4.7. ábra E/K-diagramját, amelyen a fizetés a *Szerződik* kapcsolat attribútuma. Hozzunk létre egy *Fizetések* egyedhalmazt *fizetés* attribútummal. A *Fizetések* lesz a negyedik egyedhalmaz a *Szerződik* kapcsolatban. A teljes E/K-diagram a 4.8. ábrán látható. Megjegyezzük, hogy a 4.8. ábrán a *Fizetések* egyedhalmazhoz nyíl vezet. Ez a nyíl azt fejezi ki, hogy tudjuk, hogy



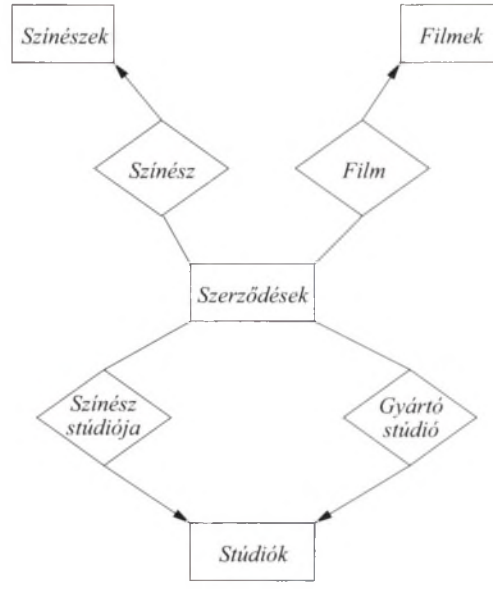
4.8. ábra. Kapcsolat attribútumának megszüntetése egy új egyed bevezetésével

a fizetés a kapcsolatban szereplő többi egyed által meghatározott. Általában, amikor a kapcsolat attribútumát átalakítjuk egy újonnan bevezetett egyedhalmaz attribútumává, akkor ehhez az egyedhalmazhoz nyíl vezet. □

4.1.10. Sokágú kapcsolatok átalakítása binárissá

Egyes adatmodellek csak bináris kapcsolatokat engednek meg. Ilyenek például az UML (4.7. alfejezet) és az ODL (4.9. alfejezet). Annak ellenére tehát, hogy az E/K-modellben nincs ilyen megszorítás, érdemes megemlíteni, hogy bármely sokágú kapcsolat átalakítható sok-egy típusú bináris kapcsolatok gyűjteményére egy új egyedhalmaz bevezetésével, melynek az egyedei tulajdonképpen a sokágú kapcsolat kapcsolathalmazának elemei. Ezt az egyedhalmazt *kapcsoló egyedhalmaz*nak nevezzük. Ezután sok-egy kapcsolatokat készítünk a kapcsoló egyedhalmaz és az eredeti kapcsolatban részt vevő egyedhalmazok között. Ha egy egyedhalmaz több szerepben is előfordult az eredeti kapcsolatban, akkor most minden szerep egy új kapcsolat lesz.

4.9. példa. A 4.6. ábrán lévő négyágú *Szerződik* kapcsolatot kicserélhetjük egy *Szerződések* egyedhalmazra. Mint a 4.9. ábrán látható, négy kapcsolatban vesz részt. Ha a *Szerződik* kapcsolat kapcsolathalmazának van egy (stúdió1, stúdió2, színész, film) eleme, akkor a *Szerződések* egyedhalmaznak is van egy e egyede. Ez az egyed a *Színész* kapcsolaton keresztül kapcsolódik a *Színészek* egyedhalmaz fenti színész egyedéhez. Ugyanígy a *Film* kapcsolaton keresztül éri el a *Filmek* egyedhalmaz a fenti film egyedét. A *Stúdiók* egyedhalmaz stúdió1 és stúdió2 egyedét pedig *Színész stúdiója* és a *Gyártó stúdió* kapcsolatokon keresztül azonosítja.



4.9. ábra. Egy sokágú kapcsolat kiváltása egy egyedhalmazzal és bináris kapcsolatokkal

Feltettük, hogy a *Szerződések* egyedhalmaznak nincs attribútuma, a többi egyedhalmaz attribútumait pedig a 4.9. ábrán nem tüntettük fel. Természetesen vehetnénk fel attribútumokat a *Szerződések* egyedhalmazhoz is, mint például az aláírás dátuma stb. □

4.1.11. Alosztályok az E/K-modellben

Gyakran előfordul, hogy egy egyedhalmaz egyedei között egyeseknek olyan speciális tulajdonságaik vannak, amelyekkel nem rendelkezik a halmaz minden egyede. Így célszerű speciális egyedhalmazokat, ún. *alosztály*okat definiálni, ahol minden alosztálynak vannak speciális attribútumai és/vagy kapcsolatai. Ha egy egyedhalmazt alosztályival speciális kapcsolat köt össze, azt „az-egy" (angolul *isa*) kapcsolatnak nevezzük (például „egy A az egy B" állítás kifejezi a speciális „az-egy" kapcsolatot A-ból B-be). Ezt a kapcsolatot más néven *öröklési kapcsolat*nak is nevezzük, és a továbbiakban így hivatkozunk rá. *(A ford.)*

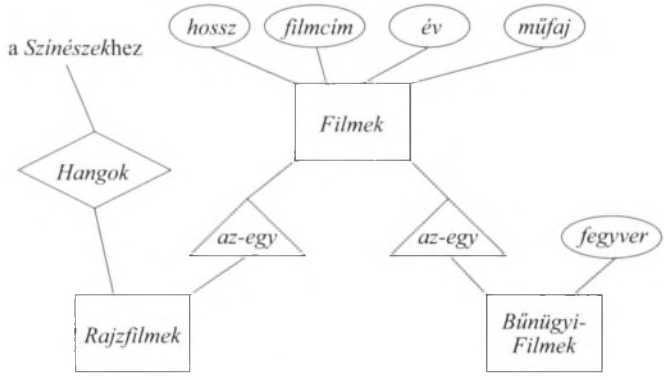
Az öröklési (az-egy) kapcsolatot a hagyományostól eltérően háromszöggel jelöljük, ezzel is kifejezve e kapcsolattípus különlegességét. A háromszög egyik oldalát az alosztállyal kötjük össze, ellenkező oldali csúcsát pedig az ősosztállyal (szuperosztállyal). Minden öröklési kapcsolat egy-egy kapcsolat, de az ezt kifejező nyilakat nem tüntetjük fel külön a diagramon.

4.10. példa. Azok között a filmek között, amelyeket tárolunk a példa adatbázisunkban, vannak rajzfilmek, bűnügyi filmek, kalandfilmek, vígjátékok és sok

A párhuzamos kapcsolatok különbözhetnek

A 4.9. ábrán láthatunk egy finomságot is a kapcsolatokra vonatkozóan. A *Szerződések* és a *Stúdiók* egyedhalmazt kétféle kapcsolat is összeköti egymással: a *Színész stúdiója* és a *Gyártó stúdió*. Ez nem jelenti azt, hogy a két kapcsolat tartalmilag megegyezik, ugyanazok a párok fordulnának elő bennük. Ez esetben ennek épp az ellenkezője igaz: nem valószínű, hogy egy szerződést mindkét kapcsolat ugyanahhoz a stúdióhoz rendelne, mivel akkor a stúdió saját magával kötne szerződést.

Általánosságban nem baj, ha egy E/K-diagramon két egyedhalmaz között többféle kapcsolatot definiálunk. Normális esetben e kapcsolatok tartalmilag is különbözni fognak az adatbázisban, jelentésüknek megfelelően.



4.10. ábra. Alosztályok az E/K-diagramban

egyéb speciális típusú film. Ezen filmtípusok mindegyike definiálható a *Filmek* egyedhalmaz alosztályaként. Például legyen az alábbi két alosztály: *Rajzfilmek* és *BűnügyiFilmek*. Egy rajzfilmnek a *Filmek* egyedhalmaznál megadott attribútumokon és kapcsolatokon kívül van még egy *Hangok* nevű kapcsolata, mely hozzárendeli azokat a színészeket, akik a filmben beszélnek, de maguk nem jelennek meg. Ilyen színészek csak a rajzfilmekhez rendelhetők. A bűnügyi filmekhez pedig tartozik egy további attribútum: a *fegyver*. A *Filmek*, *Rajzfilmek* és *BűnügyiFilmek* egyedhalmazok viszonyait a 4.10. ábra mutatja. □

Elméletileg az öröklési kapcsolattal összekötött egyedhalmazok tetszőleges struktúrát alkothatnak, de mi teszünk egy megszorítást: az ilyen leszármazási struktúrák fát kell alkossanak, melynek egy *gyökéreleme* van (például ilyen a 4.10. ábrán a *Filmek* egyedhalmaz). Ez a legáltalánosabb egyedhalmaz, a gyökértől távolodva pedig egyre speciálisabbak következnek.

Alosztályok objektumorientált rendszerekben

Lényeges hasonlóság van az E/K öröklési („az-egy" típusú) kapcsolata és az objektumorientált nyelvek alosztályai között. Az E/K-modellben az öröklési kapcsolat bizonyos értelemben egy alosztályt kapcsol össze ősosztályával (szuperosztályával). Ugyanakkor alapvető különbség is van: míg egy egyednek lehetnek reprezentánsai több egyedhalmazban, amelyek fát alkothatnak, addig egy objektum mindig csak egy osztályba vagy alosztályba tartozik bele.

A különbséget abból érzékelhetjük, ahogyan a 4.11. példában a *Roger nyúl a pácban* című filmet kezeltük. Objektumelvű megközelítésben egy negyedik egyedosztályra volna szükségünk, a *„BűnügyiRajzFilmek"*-re, amely örökölné a *Filmek*, a *Rajzfilmek* és a *BűnügyiFilmek* attribútumait és kapcsolatait. Az E/K-modellben erre a negyedik osztályra nincs szükség, mivel épp ezt érjük el, ha az említett filmet besoroljuk a *Rajzfilmek* és a *Bűnügyi-Filmek* egyedhalmazba is, és a film összetevőit a megfelelő egyedhalmazban adjuk meg.

Tegyük fel, hogy van egy egyedhalmazokból álló fa, amely öröklési kapcsolatokkal van felépítve. Ekkor egy egyed komponensekből tevődik össze, mivel a fában egy elemet valójában felmenőivel együtt kell értenünk (egészen a gyökérig). Pontosabban, ha egy e egyednek van egy E egyedhalmazhoz tartozó c komponense, és a fában E szülője az F egyedhalmaz, akkor e-nek kell legyen egy d komponense is az F egyedhalmazban. Továbbá a c-d párnak szerepelnie kell az E-t F-fel összekötő öröklési kapcsolatban. Az e egyed attribútumait a komponensek attribútumainak összessége alkotja, és e részt vesz mindazokban a kapcsolatokban, melyekben valamely komponense részt vesz.

4.11. példa. Tekintsük a 4.10. ábrát. Egy film, amely nem rajzfilm és nem is bűnügyi film, csak egy komponensből áll, mely a *Filmek* egyedhalmazhoz tartozik, és ennek megfelelően csak négy attribútuma lesz (valamint két kapcsolatban vehet részt: *SzerepelBenne* és *Gyártó* – ezeket nem tüntettük fel a 4.10. ábrán).

Egy rajzfilm, mely nem bűnügyi film, két komponensből áll: egyik a *Filmek*, másik a *Rajzfilmek* egyedhalmaznak az eleme. Egy ilyen film rendelkezik az előbb említett négy attribútummal, és a *Filmek* kapcsolatai mellett a *Hangok* kapcsolat is hozzátartozik. Hasonlóan, egy bűnügyi film két komponensből áll (*Filmek* és *BűnügyiFilmek*), és így összesen öt attribútuma van, beleértve a *fegyver*-t is.

Egy olyan egyed, mint a *Roger nyúl a pácban*, amely egyszerre rajzfilm és bűnügyi film, a komponenseit három egyedhalmazból, a *Filmek*, a *Rajzfilmek* és a *BűnügyiFilmek* egyedhalmazokból gyűjti össze. A három komponens az öröklődési kapcsolaton keresztül kapcsolódik egy egyedben. Együtt a három komponens adja a *Roger nyúl a pácban* egyed attribútumait és kapcsolatait.

Négy attribútumot és két kapcsolatot ad a *Filmek* egyedhalmaz, a *fegyver* attribútumot adja a *BűnügyiFilmek* egyedhalmaz és a *Hangok* kapcsolatot adja a *Rajzfilmek* egyedhalmaz. □

4.1.12. Feladatok

4.1.1. feladat. Tervezzünk egy bank részére adatbázist, amely tartalmazza az ügyfeleket és azok számláit. Az ügyfelekről tartsuk nyilván nevüket, címüket, telefonszámukat és TAJ-számukat. A számláknak legyen számlaszámuk, típusuk (például takarékbetét-számla, folyószámla stb.) és egyenlegük. Továbbá meg kell jelölni azokat az ügyfeleket, akiknek van számlájuk. Adjuk meg az E/K-diagramját ennek az adatbázisnak. Alkalmazzunk nyilakat a kapcsolatokban a multiplicitások jelölésére.

4.1.2. feladat. Módosítsuk a 4.1.1. feladatra adott megoldásunkat az alábbiak szerint:

a) Változtassuk meg a diagramot úgy, hogy egy számlának csak egy tulajdonosa lehessen.

b) Változtassuk tovább a diagramot úgy, hogy egy ügyfélnek csak egy számlája lehessen.

! *c)* A 4.1.1. feladat eredeti diagramját változtassuk meg úgy, hogy egy ügyfélhez rendelhessünk lakcímeknek (ország-város-utca hármassal megadva) és telefonszámoknak egy halmazát. Ügyeljünk arra, hogy az E/K-modellben nem megengedett, hogy attribútumoknak valamilyen összetett típusa (például halmaz) legyen.

! *d)* Módosítsuk tovább a diagramunkat úgy, hogy minden ügyfélhez tartozhasson lakcímek egy halmaza, és minden egyes lakcímhez telefonszámoknak egy halmaza.

4.1.3. feladat. Adjuk meg az E/K-modelljét egy olyan adatbázisnak, amely csapatokat, játékosokat és azok szurkolóit tartja nyilván:

1. Minden csapatról tároljuk a nevét, játékosait, csapatkapitányát (ő is egy játékos) és a mezük színét.

2. Minden játékosnak legyen neve.

3. Minden rajongóról tartsuk nyilván a nevét, kedvenc csapatát, kedvenc játékosát és kedvenc színét.

Vigyázzunk, a színek halmaza nem lehet a csapatok egy attribútumának típusa. Hogyan lehet ezzel a megszorítással együtt megfelelő modellt készíteni?

4.1.4. feladat. Tegyük fel, hogy szeretnénk hozzávenni a 4.1.3. feladat E/K-diagramjához a *Vezeti* kapcsolatot, ami két játékos közötti viszonyt fejez ki egy csapaton belül. Ez a kapcsolat (játékos1, játékos2, csapat) hármasok halmaza úgy, hogy a játékos2 volt a csapat kapitánya, amikor együtt játszott a játékos1-gyel.

a) Rajzoljuk meg ezt a módosítást az E/K-diagramon.

b) Cseréljük ki új egyedhalmazra és bináris kapcsolatokra a fenti hármas kapcsolatot.

! *c)* Az új bináris kapcsolatok ugyanolyanok, mint a korábban meglévő kapcsolatok? Feltételezhetjük, hogy a két játékos különböző, azaz egy kapitány nem vezeti saját magát.

4.1.5. feladat. Módosítsuk a 4.1.3. feladatot úgy, hogy a játékosokról jegyezzük fel, korábban mely csapatokban játszottak, beleértve a belépés és kilépés dátumát is.

! **4.1.6. feladat.** Tegyük fel, hogy családfa-adatbázist tervezünk az egyetlen, *Személyek* egyedhalmazzal. Egy személyről a következő információkat kívánjuk tárolni: neve (attribútumként), anyja, apja és gyerekei (kapcsolatokként).

! **4.1.7. feladat.** Módosítsuk a 4.1.6. feladatban lévő adatbázistervet a következő speciális személytípusokkal:

1. Nők.
2. Férfiak.
3. Szülők.

Megkülönböztethetünk más típusokat is, hogy a kapcsolatok a megfelelő alosztályokhoz kapcsolódjanak.

4.1.8. feladat. Alternatív megoldás lehet a 4.1.6. feladatban meghatározott információk reprezentálásának a *Család* hármas kapcsolat. Ez a kapcsolat (személy, anya, apa) hármasok halmaza, ahol egy személyt és szüleit tároljuk. Természetesen minden adat a *Személyek* egyedhalmazból kerül ki.

a) Rajzoljuk meg ezt a diagramot. Helyezzük el a nyilakat a megfelelő helyekre.

b) Cseréljük ki a *Család* hármas kapcsolatot egy egyedhalmazra és bináris kapcsolatokra. Ismét helyezzük el a nyilakat a kapcsolatok típusának megfelelően.

4.1.9. feladat. Tervezzünk adatbázist egy tanulmányi osztály számára. Ez az adatbázis tartalmazza a hallgatókat, oktatókat, tanszékeket és kurzusokat. Ezenkívül tartsuk nyilván, hogy a hallgatók milyen kurzusokat vettek fel, az adott kurzust mely oktató oktatja, a hallgatók jegyeit, a kurzusoknál az oktató munkáját segítő hallgatókat, egy adott kurzust mely tanszék ajánlotta, és minden olyan információt, amely a fentiek megvalósításához szükséges. Megjegyezzük, hogy ez a feladat nagy szabadságot enged a korábbiakhoz képest. Dönteni kell a kapcsolatok típusáról (sok-sok, sok-egy vagy egy-egy), az alkalmas típus megválasztásáról, illetve arról, hogy milyen segédinformációkat használunk.

! **4.1.10. feladat.** Informálisan két E/K-diagramról azt mondhatjuk, hogy „ugyanazt az információt hordozzák", ha bármely valós helyzetre igaz, hogy az egyik diagram előfordulásából kiszámítható a másik diagram előfordulása, mely ugyanazt a helyzetet írja le. Tekintsük a 4.6. ábrát. A négyágú kapcsolat szétbontható egy háromágú és egy kétágú kapcsolattá, figyelembe véve, hogy minden egyes filmhez egy gyártó stúdió tartozik. Adjunk meg egy E/K-diagramot, melyben nem szerepel négyágú kapcsolat, és ugyanazt az információt hordozza, mint a 4.6. ábra.

4.2. Tervezési alapelvek

Még nagyon sok részletet kell megismernünk az E/K-modellel kapcsolatban, de már eleget tudunk ahhoz, hogy elkezdjük megvizsgálni a lényeges pontjait annak, hogy mitől jó egy terv és mit kell elkerülni. Ebben az alfejezetben néhány hasznos tervezési elvet ajánlunk az Olvasó figyelmébe.

4.2.1. Valósághű modellezés

Az első és legfontosabb, hogy a tervnek pontosan meg kell felelnie az alkalmazás specifikációjának. Azaz, az egyedhalmazoknak és attribútumaiknak tükrözniük kell a valóságot. Nem lehet a *Színészek* egyedhalmaznak attribútuma a *hengerekSzáma*, hiszen ez inkább az *Autók* egyedhalmazhoz tartozik. Akármilyen kapcsolatok jönnek is létre, azoknak értelmeseknek kell lenniük az alapján, amit ismerünk a valós világ modellezendő részéről.

4.12. példa. Ha definiálunk egy *SzerepelBenne* kapcsolatot a *Színészek* és a *Filmek* között, annak sok-sok kapcsolatnak kell lennie. Az ok az, hogy a valós világban megfigyelhetjük, hogy színészek több filmben is szerepelhetnek, és a filmekben egynél több színész is szerepelhet. Tehát nem lenne korrekt a *SzerepelBenne* kapcsolatot sok-egy vagy egy-egy kapcsolatnak definiálni. □

4.13. példa. Másfelől nem mindig magától értetődő, hogyan modellezzük a valóságot E/K-diagrammal. Tegyük fel, hogy van egy *Tantárgyak* és egy *Oktatók* egyedhalmazunk, köztük az *Oktat* kapcsolattal. Kérdés, hogy e kapcsolat sok-egy típusú-e a *Tantárgyak* felől az *Oktatók*hoz. A válasz az adatbázist létrehozó szervezet szándékain és működési szabályain múlik. Elképzelhető, hogy

az iskolában egy tárgyat csak egy valaki oktathat. Még akkor is, ha többen oktatnak egy tárgyat, szükséges lehet közülük pontosan egy valakit kijelölni, mint a tárgy felelősét. Mindkét esetben az *Oktat* kapcsolatot sok-egy típusúnak kell megadnunk a *Tantárgyak* egyedhalmazból az *Oktatók* egyedhalmazba.

Elképzelhető az is, hogy az iskolában egy-egy tárgyat többen oktatnak, és szeretnénk, ha több oktatót is hozzá lehetne rendelni egy tárgyhoz az adatbázisban. Sőt az *Oktat* kapcsolat nem csak azt fejezheti ki egy tárgy esetében, hogy éppen ki oktatja, hanem akár azt, hogy kik voltak ez idáig a tárgy oktatói, vagy akár azt, hogy képességei alapján ki oktathatja. Ezt nem lehet eldönteni pusztán a kapcsolat nevének ismeretében. Ezekben az esetekben az a helyes, ha a kapcsolatot sok-sok típusúnak adjuk meg. □

4.2.2. Redundancia elkerülése

Óvatosnak kell lennünk, hogy minden csak egyszer szerepeljen. A 3.3. alfejezetben tárgyalt, a redundanciákkal és anomáliákkal kapcsolatos problémák tipikusan olyan problémák, amelyek az E/K-tervezés során keletkeznek. Ezentúl az E/K-modellezésben megjelenik több új eljárás, amelyek redundanciát és más anomáliákat okozhatnak.

Például használtunk egy *Gyártó* kapcsolatot a filmek és a stúdiók között. A kapcsolat mellé még felvehetnénk egy *stúdióNév* attribútumot is a *Filmek* egyedhalmazba. Ez két okból is veszélyes.

1. Adattárolás szempontjából ugyanazon gyártó stúdió esetén a kétszeri előfordulás több helyet igényel, mint az egyszeri.

2. Mivel módosíthatnánk a kapcsolatot az attribútum módosítása nélkül vagy fordítva, ezzel az update-anomália (módosítási anomália) lehetőségét építenénk be.

A 4.2.4. és 4.2.5. alfejezetekben bővebben foglalkozunk az anomáliák elkerülésével.

4.2.3. Egyszerűség

Ne vegyünk fel több elemet az adatbázistervünkbe, mint amennyi feltétlenül szükséges.

4.14. példa. Tegyük fel, hogy a *Filmek* és a *Stúdiók* közötti kapcsolat helyett „filmtársaságokat" vennénk, amelyek egy filmnek a gyártói lennének. Ekkor fel kell vennünk egy *Társaságok* egyedhalmazt, amely egy-egy kapcsolatban lenne a filmekkel. Ez a *Képvisel* kapcsolat mutatná meg minden filmhez egyértelműen azt a társaságot, amelyik gyártotta a filmet. Végül egy sok-egy kapcsolat lenne a *Társaságok* és a *Stúdiók* között, ahogyan a 4.11. ábra mutatja.

Tulajdonképpen a 4.11. ábrán lévő struktúra a valós világot reprezentálja, mivel lehetséges, hogy egy filmtől a gyártó stúdióhoz a *Társaságok*on keresztül juthatunk el. Azonban a *Társaságok* nem szolgálnak semmi egyéb hasznos

4.11. ábra. Egy terv szükségtelen egyedhalmazzal

célt, így jobb, ha elhagyjuk őket. A film-stúdió kapcsolatot kezelő, illetve használó programok bonyolultabbak, több helyet foglalnak és több hibalehetőséget rejtenek magukban a *Társaságok* egyedhalmaz használatával. □

4.2.4. A megfelelő kapcsolatok megválasztása

Az egyedhalmazokat többféleképpen köthetjük össze kapcsolatokkal. Általában azonban nem jó megoldás, ha hozzáadjuk a tervhez az összes lehetséges kapcsolatot. Ez redundanciához, módosítási és törlési anomáliához vezethet, amikor is valamely kapcsolat esetén más kapcsolatokból kikövetkeztethető, hogy mely egyedek melyekkel együtt vesznek benne részt. Két példával érzékeltetjük a problémát és azt, hogy mit tehetünk. Az első példában több kapcsolat ugyanazt az információt reprezentálhatja, míg a másodikban az egyik kapcsolat levezethető a többiből.

4.15. példa. Tekintsük a 4.7. ábrát, ahol a filmek, a színészek és a stúdiók a háromágú *Szerződik* kapcsolattal vannak összekötve. A 4.2. ábrához képest az a különbség, hogy innen kihagytuk a bináris *SzerepelBenne* és *Gyártó* kapcsolatokat. Szükségünk van-e ezekre a kapcsolatokra a *Filmek* és *Színészek*, valamint a *Filmek* és *Stúdiók* egyedhalmazok között? A válasz az, hogy nem tudjuk; attól függ, milyen feltételezéssel élünk a kérdéses háromágú kapcsolatról.

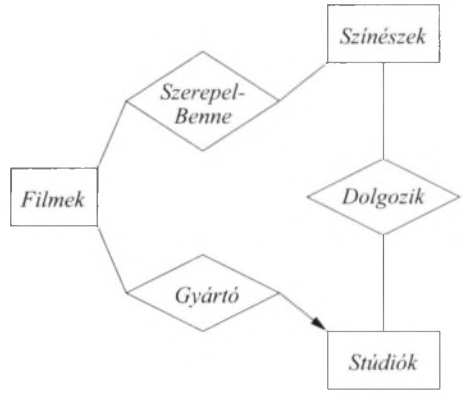
Nem kizárt, hogy a *SzerepelBenne* kapcsolat egyértelműen következik a *Szerződik* kapcsolatból. Ha egy színész csak akkor szerepelhet egy filmben, ha szerződése van arra a filmre nézve a gyártó stúdióval, akkor valóban nincs szükség a *SzerepelBenne* kapcsolatra. A színész-film párokat a *Szerződik* színész-film-stúdió hármasaiból megkapjuk. Ha viszont egy színész szerződés nélkül is szerepelhet – vagy olyan szerződéssel szerepel, amely nem ismert az adatbázisban –, akkor lehetnek olyan színész-film párok a *SzerepelBenne* kapcsolatban, melyek nem fordulnak elő a *Szerződik* színész-film-stúdió hármasaiban. Ez esetben meg kell tartanunk a *SzerepelBenne* kapcsolatot.

Hasonlót mondhatunk a *Gyártó* kapcsolatról. Ha minden filmre vonatkozóan van szerződése a gyártó stúdiónak legalább egy szereplővel, akkor a *Gyártó* kapcsolat felesleges. De ha lehetséges, hogy egy stúdió gyárt egy filmet és éppen nincs hozzá szerződött szereplő az adatbázisban (vagy tényleg nincs, vagy nem ismert az adatbázis számára), akkor nem hagyhatjuk el a *Gyártó* kapcsolatot.

Összességében tehát ránézésre nem tudjuk megmondani egy kapcsolatról, hogy redundáns-e. Az adatbázis megrendelőitől kell megtudakolni a viszonyokat, hogy mire számíthatunk, és csak ezek alapján dönthető el ésszerűen, hogy legyen-e *SzerepelBenne*, illetve *Gyártó* kapcsolat a tervben. □

4.16. példa. Tekintsük újra a 4.2. ábrát. E diagramon nem kötöttük össze kapcsolattal a *Színészek* és a *Stúdiók* egyedhalmazokat. A *SzerepelBenne* és a *Gyártó* kapcsolatok kompozíciója által viszont kapcsolatba kerülnek. Így azt mondhatjuk, hogy egy színész azokkal a stúdiókkal van kapcsolatban, melyek filmjeiben szerepel.

Lenne-e értelme még egy *Dolgozik* kapcsolatnak is a *Színészek* és a *Stúdiók* között, a 4.12. ábra szerint? További ismeretek nélkül ezt nem tudjuk eldönteni. Először is, mit jelentene ez a kapcsolat? Ha azt, hogy a színész szerepel a stúdió egyik filmjében, akkor szükségtelen a kapcsolat, mivel a fentiek szerint ez következtethető a *SzerepelBenne* és a *Gyártó* kapcsolatból is.



4.12. ábra. További kapcsolat a *Színészek* és a *Stúdiók* között

Az is lehetséges viszont, hogy olyan további információk is rendelkezésre állnak színészek és stúdiók munkaviszonyáról, melyek nem egy filmen keresztül származtathatók. Ez esetben a színészek és stúdiók közti közvetlen kapcsolat hasznos lehet, nem redundáns. Az is lehet, hogy a színészek és a stúdiók között egy teljesen más értelmű kapcsolat van. Ez jelentheti például azt, hogy egy színész filmektől függetlenül szerződésben áll egy stúdióval. A 4.7. példában már előfordult, hogy egy színész szerződött egy stúdióval, de egy másik stúdió filmjében is dolgozik. Ez esetben a *Dolgozik* kapcsolat független a *SzerepelBenne* és a *Gyártó* kapcsolattól, így nem is redundáns. □

4.2.5. A megfelelő típusú elem megválasztása

Néha lehetőségünk van egy elem típusát többféle módon is megválasztani úgy, hogy még mindig a valós világot reprezentálja. Ezen lehetőségek közül a leggyakoribb, hogy azt kell eldönteni, attribútumot vagy egyedhalmazt és kapcsolatot használjunk-e. Általában egy attribútumot egyszerűbb implementálni, mint egy egyedhalmazt vagy egy kapcsolatot. Azonban ha mindent attribútumnak választunk, akkor abból rendszerint problémák származnak.

4.17. példa. Vizsgáljunk meg egy jellegzetes problémát. A 4.2. ábrán jól tettük-e, hogy a stúdiókat egy egyedhalmaznak választottuk? Lehetnének-e a stúdiónevek és címek a filmek attribútumai, és így a *Stúdiók* egyedhalmazt elhagyhatnánk? Egyik probléma, hogy a stúdiók címét minden filmnél meg kell ismételni. Ezzel módosítási anomáliát teremtünk, ha a stúdió címét egy filmnél kicseréljük, de a többinél nem, és a törlési anomália is megjelenik, amikor egy adott stúdió által gyártott utolsó filmet töröljük.

Ha azonban nem tároljuk a stúdiók címét, akkor már nem káros a stúdiók nevét a `Filmek` egyedhalmaz egy attribútumaként tárolni. Ekkor nem lesz redundancia a címek ismétléséből. Az, hogy a stúdiók nevét meg kell mondani minden filmnél, nem valódi redundancia, mivel ezt valahol úgyis meg kell mondani, és ez erre egy alkalmas megoldás. □

Elvonatkoztatva a 4.17. példa konkrétumaitól, megadunk néhány feltételt arra vonatkozóan, mikor érdemes inkább attribútumot és nem egyedhalmazt használni. Tegyük fel, hogy E egy egyedhalmaz. E az alábbi feltételekkel alakítható át attribútummá, vagy több egyedhalmaz attribútumaivá:

1. Minden kapcsolat esetén, melyben E részt vesz, nyíl kell mutasson E felé; azaz mindig az „egy" oldalán kell álljon egy sok-egy típusú kapcsolatban és többágú általánosításában.

2. Ha E egyedhalmaznak egynél több attribútuma van, akkor egyik attribútum sem függhet a többitől, mint ahogy például a *Stúdiók* esetén a *cím* függ a *név*től. Ezt úgy is kifejezhetjük, hogy E egyetlen kulcsa az összes attribútumaiból álló halmaz.

3. Egynél többször nem szerepelhet E egy kapcsolatban.

Ha ezek a feltételek teljesülnek, az E egyedhalmazt az alábbi módon helyettesíthetjük:

a) Ha egy másik, F egyedhalmaz egy R nevű, sok-egy kapcsolaton keresztül kapcsolódik E-hez, akkor R-et elhagyjuk és E attribútumait F kapja meg (alkalmas átnevezésekkel, ha F meglévő attribútumaival ütköznek E attribútumnevei). Így tehát minden egyes F-beli egyed a hozzá egyértelműen kapcsolódó, eredeti E-beli egyed attribútumaival bővül[2], mint ahogyan a film objektumok attribútumként megkapják a gyártó stúdiójuk nevét, ha a stúdió címét nem akarjuk tárolni.

b) Ha E egy többágú, R nevű kapcsolatban vesz részt, akkor E attribútumait R kapja meg és E-t töröljük a kapcsolatból. Erre példa a 4.8. ábra visszaalakítása az eredeti, 4.7. diagramra, amikor is eltávolítjuk az egyetlen attribútummal rendelkező *Fizetések* egyedhalmazt.

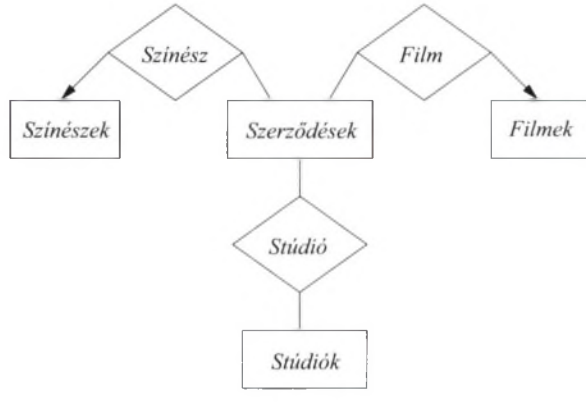
[2] Abban az esetben, ha egy F-beli egyed nem kapcsolódik E-beli egyedhez, akkor – jelölve ennek hiányát – az F új attribútumain speciális, „null" értéket kap. A *b)* esetben hasonlóan járhatunk el az R új attribútumainál.

4.18. példa. Vizsgáljuk meg a sokágú kapcsolatok használata és a kapcsoló egyedhalmaz bináris kapcsolatokkal való használata közti különbséget. Láttuk a négyágú *Szerződik* kapcsolatot egy színész, egy film és két stúdió között a 4.6. ábrán, amelyet mechanikusan átkonvertáltunk egy *Szerződések* egyedhalmazzá (4.9. ábra). Vajon jól választottunk?

Annak alapján, ahogy a feladatot megfogalmaztuk, mindkét lehetőségünk megvan. Azonban ha változtatunk egy kicsit a problémán, akkor már majdnem biztos, hogy a kapcsoló egyedhalmazt kell választanunk. Tegyük fel, hogy a szerződések egy színészt, egy filmet, de a stúdiók tetszőleges halmazát tartalmazhatja. Ez a szituáció bonyolultabb, mint ami a 4.6. ábrán látható, ahol stúdiók csak kétféle szerepben lehetnek. Ebben az esetben bármennyi stúdió szerepelhet a kapcsolatban, például egy, amely a gyártást végzi, egy, amely a speciális effektusokat, egy, amely a terjesztést stb. Így nem tudunk a stúdiókhoz szerepeket rendelni.

Itt a *Szerződik* kapcsolat kapcsolathalmaza a következő alakú hármasokból kell, hogy álljon: (színész, film, stúdiók halmaza), és a *Szerződik* kapcsolat nemcsak a *Színészek* és a *Filmek* egyedhalmazok között van, hanem egy új egyedhalmaz is részt vesz a kapcsolatban, amelynek az egyedei a stúdiók halmazai. Ez a megközelítés ugyan lehetséges, azonban a stúdiók halmaza mint alapegyed természetellenesnek tűnik, és így nem javasoljuk.

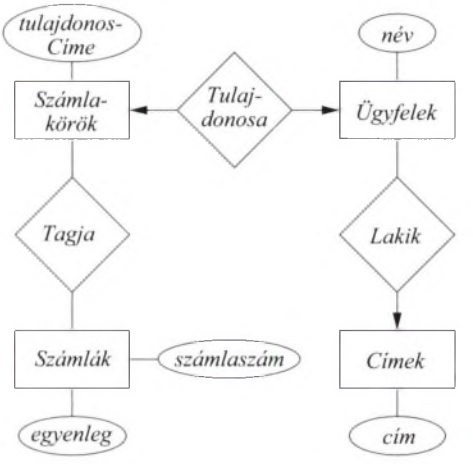
Jobb megközelítés, ha a szerződéseket mint egyedhalmazt tekintjük. Mint a 4.9. ábrán, egy szerződésegyedhez kapcsolódik egy színész, egy film és a stúdióknak egy halmaza, azonban most nincs korlát a stúdiók számára. Így a szerződések és stúdiók közötti kapcsolat egy sok-sok kapcsolat, ellentétben a sok-egy kapcsolatokkal, amikor a szerződések egy valódi kapcsoló egyedhalmaz volt. A 4.13. ábra mutatja az E/K-diagramot. Látható, hogy egy szerződéshez tartozik egy színész, egy film és akárhány stúdió. □



4.13. ábra. A *Szerződések*, mint olyan kapcsoló egyedhalmaz, amely egy színészhez, egy filmhez és a stúdiók egy halmazához kapcsolódik

4.2.6. Feladatok

4.2.1. feladat. A 4.14. ábrán egy banki adatbázis terve látható, amely az ügyfeleket és számláikat tartja nyilván. Mivel egy-egy ügyfélnek több számlája is lehet, és egy-egy számlához több tulajdonos is tartozhat, minden ügyfélhez „számlakört" rendelünk, és a számlák egy vagy több számlakörhöz tartozhatnak. Tegyük fel, hogy a különböző kapcsolatok és attribútumok jelentése megegyezik azzal, amit a nevük sugall. Bíráljuk felül a tervet. Mely szabályokat sérti? Miért? Milyen módosításokat javasolunk?



4.14. ábra. Egy banki adatbázis nem megfelelő terve

4.2.2. feladat. Tekintsük a 4.3. ábrán látható, *Stúdiók* és *Elnökök* egyedhalmazokat, valamint a köztük lévő kapcsolatot. A fel nem tüntetett további attribútumokra vonatkozóan milyen feltételek között vonhatnánk mindezeket össze egyetlen egyedhalmazzá?

4.2.3. feladat. Ha a 4.7. ábrán a *Stúdiók* egyedhalmazból eltávolítjuk a *cím* attribútumot, hogyan helyettesíthetjük az egyedhalmazt egyetlen attribútummal? Hová kerülne ez az attribútum?

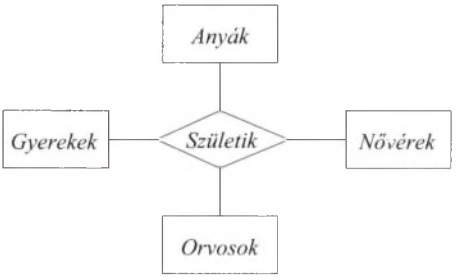
4.2.4. feladat. Adjunk meg úgy attribútumokat a 4.13. ábra alábbi egyedhalmazaihoz, hogy ezen egyedhalmazok attribútummá alakíthatók legyenek:

a) Színészek,

b) Filmek,

! *c) Stúdiók.*

!! **4.2.5. feladat.** Ebben és a következő feladatokban a születés E/K-modelljének két lehetséges verzióját vizsgáljuk meg. Egy születésnél van egy gyerek (ikrek születését két szüléssel reprezentáljuk), egy anya, valamennyi nővér és valamennyi orvos. Ezért tegyük fel, hogy *Gyerekek*, *Anyák*, *Nővérek* és *Orvosok* egyedhalmazaink vannak. Ezenkívül vegyünk egy *Születik* kapcsolatot, amely a négy egyedhalmazt kapcsolja össze a 4.15. ábrán javasolt módon. Megjegyezzük, hogy a *Születik* kapcsolat kapcsolathalmazának egy eleme a következő alakú (gyerek, anya, nővér, orvos). Ha több mint egy nővér és/vagy orvos segít egy szülést, akkor több elem lesz ugyanahhoz a gyerekhez és anyához a kapcsolathalmazban, a nővérek és orvosok összes lehetséges kombinációjának megfelelően.



4.15. ábra. A születések megvalósítása sokágú kapcsolattal

A következő feltevéseket szeretnénk beépíteni a tervünkbe. Mondjuk meg, hogy milyen nyilakat vagy egyéb elemeket kell felvenni az E/K-diagramba, hogy teljesüljenek az alábbi feltételek:

a) Minden gyereknek pontosan egy anyja van.

b) Egy gyerek-nővér-orvos hármashoz pontosan egy anya tartozik.

c) Egy adott gyerek és anya kombinációhoz pontosan egy orvos tartozik.

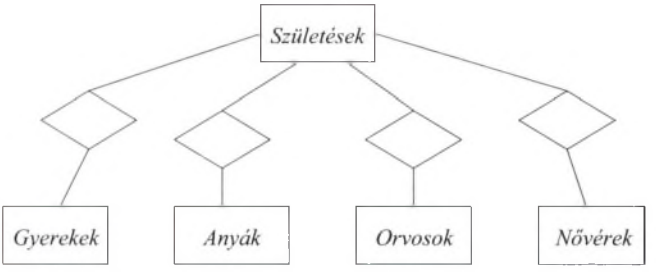
! **4.2.6. feladat.** A 4.2.5. feladat problémájának egy másik megközelítése, ha a *Gyerekek*, *Anyák*, *Nővérek* és *Orvosok* négy egyedhalmazt egy *Születések* egyedhalmazon keresztül kapcsoljuk össze négy bináris kapcsolat segítségével, és a kapcsolatok mindig a *Születések* egyedhalmaz és valamelyik másik egyedhalmaz között vannak, ahogyan az a 4.16. ábrán látható. Használjunk nyilakat (jelezve, hogy bizonyos kapcsolatok sok-egy kapcsolatok) a következő feltételek megvalósításához:

a) Minden gyerek egy születésnek az eredménye és minden születés pontosan egy gyerekhez tartozik.

b) Az *a)* feltételt egészítsük ki azzal, hogy minden gyereknek pontosan egy anyja van.

c) Az *a)* és *b)* feltételeket egészítsük ki azzal, hogy minden születés pontosan egy orvoshoz tartozik.

Milyen tervezési hibákkal terheltek az egyes esetek?



4.16. ábra. A születéseket egy egyedhalmazzal adjuk meg

!! **4.2.7. feladat.** Tegyük fel, hogy megváltoztatjuk a nézőpontunkat és megengedjük, hogy egy szülés alatt több gyereket is szülhessen egy anya. Hogyan lehetne ekkor megvalósítani azt, hogy minden gyereknek pontosan egy anyja van a 4.2.5. feladatban leírt és a 4.2.6. feladatban megadott megközelítések esetében külön-külön?

4.3. Megszorítások modellezése

Az E/K-modell számos lehetőséget ad arra, hogy a tervezés alatt álló adatbázis adataira vonatkozó szokásos típusú előírásokat adjunk meg. A relációs modellhez hasonlóan itt is meg tudjuk adni, hogy egy attribútum vagy attribútumok egy halmaza az egyedhalmaz kulcsa. Láttuk azt is, hogy egy kapcsolatból az egyedhalmazhoz vezető nyíl a „funkcionális függőség" kifejezésére szolgál. Arra is van mód, hogy kifejezzük a hivatkozásiépség-megszorítást, amikor egy egyedhalmaz egy egyedének egy másik egyedhalmazbeli egyedhez kell kapcsolódnia.

4.3.1. Kulcsok az E/K-modellben

Egy E egyedhalmaz *kulcs*a egy vagy több attribútum K halmaza úgy, hogy tetszőleges két, egymástól különböző e_1 és e_2 egyed nem egyezhet meg a K-beli attribútumok mindegyikén. Ha K több attribútumot is tartalmaz, akkor e_1 és e_2 megegyezhet azok egy részén, de nem mindegyiken. Három fontos szempontra hívjuk fel a figyelmet:

- Minden egyedhalmaznak kell legyen kulcsa, még az az-egy-hierarchiában részt vevő és a gyenge egyedhalmazok (lásd 4.4. alfejezet) esetében is, ilyenkor a kulcs más egyedhalmazhoz tartozik.

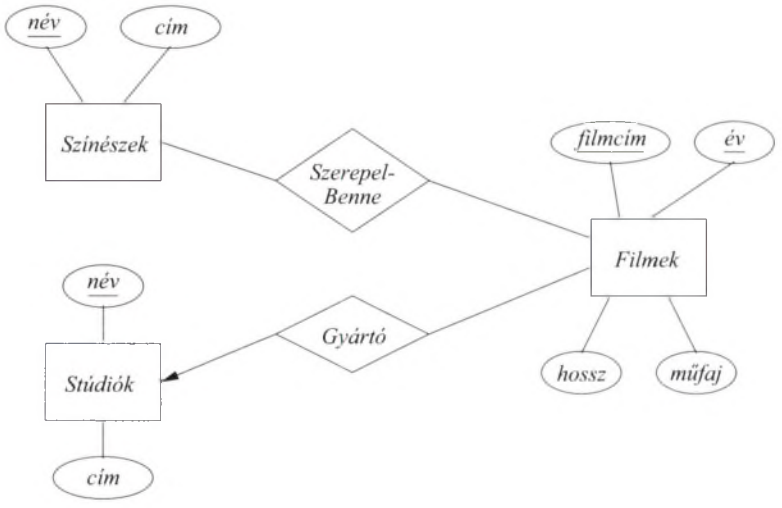
- Egy egyedhalmaz több lehetséges kulccsal is rendelkezhet. Szokás szerint ilyenkor választunk közülük egy „elsődleges kulcsot" és úgy tekintjük, mintha csak az volna kulcs.

- Ha egy egyedhalmaz egy öröklési („az-egy" típusú) kapcsolatok által meghatározott hierarchiában szerepel, akkor a gyökérben lévő egyedhalmaznak kell rendelkeznie a kulcshoz szükséges attribútumokkal; és minden egyed kulcsát a gyökérhalmazbeli komponense határozza meg, függetlenül attól, hogy milyen további komponensei vannak a hierarchiában.

A filmes példánkban a *filmcím* és *év* attribútumok halmazát használjuk a *Filmek* kulcsának, arra a megfigyelésre alapozva, hogy valószínűtlen az, hogy ugyanabban az évben két ugyanolyan című film készülne. Azt is eldöntöttük, hogy a *Színészek* esetében a *név* kellően biztonságos kulcs, feltehetjük ugyanis, hogy egyik színész sem kívánja másik színész nevét használni (inkább „művésznevet" választanak maguknak).

4.3.2. Kulcsok jelölése az E/K-modellben

Az E/K-diagramokban aláhúzással jelöljük az egyedhalmazok kulcsattribútumait. Például a 4.17. ábrán a 4.2. ábra diagramja látható, a filmek, színészek és stúdiók kulcsainak aláhúzásával. A *név* a *Színészek* egyattribútumos kulcsa, és hasonlóan a *Stúdiók* kulcsa is a saját *név* attribútuma. A *Filmek* kulcsa pedig két attribútumból áll: *filmcím* és *év*.



4.17. ábra. Kulcsok jelölése aláhúzással az E/K-diagramon

Ha egy egyedhalmaznak több kulcsa is van, azt nem jelöljük, csak az elsődleges kulcsot. Ha egyszerre több attribútum van aláhúzva (mint ahogy a

4.17. ábrán a *Filmek* esetén), akkor azok együtt alkotják a kulcsot. Érdemes megemlíteni, hogy egyes, szokatlan esetekben előfordulhat, hogy a kulcsnak nem minden attribútuma tartozik magához az egyedhalmazhoz. Erre a 4.4. alfejezetben térünk vissza („gyenge" egyedhalmazok).

4.3.3. Hivatkozások épsége

Gondoljunk vissza a 2.5.2. alfejezetben tárgyalt hivatkozásiépség-megszorításokra. Ez a megszorítás azt mondja, hogy az egyik egyedhalmazban szereplő értéknek elő kell fordulnia egy (adott) másik egyedhalmazban is. Vizsgáljuk meg például a 4.2. ábra *Gyártó* kapcsolatát, mely sok-egy típusú a *Filmek* irányából a *Stúdiók* felé. A sok-egy kapcsolat pusztán annyit követel meg, hogy ne lehessen olyan film, amelyet egynél több stúdió gyárt. Azt viszont nem, hogy minden filmhez tartozzon gyártó stúdió, és ha tartozik is, benne kell legyen az adatbázisban, a *Stúdiók* egyedhalmaz elemeként. A hivatkozások épsége azt követeli meg a *Gyártó* esetében, hogy a (kapcsolat által hivatkozott) stúdió egyed minden filmhez létezzen az adatbázisban.

Az E/K-diagramokban használt nyíl jelölés alkalmas arra is, hogy mutassa meg egy kapcsolatról, hogy az elvárja a hivatkozás épségének megőrzését. Tegyük fel, hogy R egy kapcsolat E és F egyedhalmazok között. Kerek nyílvéget fogunk használni annak jelölésére, hogy egy kapcsolat nemcsak sok-egy vagy egy-egy kapcsolat, hanem megköveteli az egy oldalon a hivatkozás épségét, azaz E minden egyedéhez kell, hogy létezzen egy vele kapcsolatban álló F egyedhalmazbeli egyed. Ugyanez az ötlet használható, amikor R több egyedhalmaz közötti kapcsolat.

4.19. példa. A 4.18. ábra mutatja a hivatkozásiépség-megszorításokat a *Filmek*, *Stúdiók* és *Elnökök* egyedhalmazok között. Ezeket az egyedhalmazokat és kapcsolatokat a 4.2. és 4.3. ábrákon vezettük be. Láthatjuk, hogy a *Stúdiók* egyedhalmazhoz megy egy kerek nyíl a *Gyártó* kapcsolatban. Ez azt jelenti, hogy minden filmhez kell hogy tartozzon egy és csak egy gyártó stúdió, melynek benne kell lennie a *Stúdiók* egyedhalmazban.

4.18. ábra. Hivatkozásiépség-megszorításokat tartalmazó E/K-diagram

Hasonlóan kerek nyilat találunk a *Irányít* kapcsolatban szintén a *Stúdiók* oldalán. Ez azt jelenti, hogy minden elnök vezet egy stúdiót, amely létezik a *Stúdiók* egyedhalmazában.

Megjegyezzük, hogy a *Irányít* kapcsolat *Elnökök* oldala nem kerek nyíl maradt. Ennek oka a következő ésszerű feltevés a stúdiók és elnökeik közötti kapcsolatról. Ha egy stúdió megszűnik, akkor az elnökét nem lehet többé (stúdió)elnöknek nevezni, így elvárható, hogy ez az elnök törlődjön az *Elnökök* egyedhalmazból. Ezért van a kerek nyíl a *Stúdiók* oldalon. Másrészről, ha egy

elnök törlődik az adatbázisból, akkor a stúdió még tovább létezhet. Ezért helyeztünk el hagyományos nyilat az *Elnökök* oldalon, ami azt jelenti, hogy minden stúdiónak legfeljebb egy elnöke van, de lehet olyan időszak, amikor éppen nincs elnöke. □

4.3.4. Egyéb megszorítások

Az E/K-modellben a kapcsolat fokára vonatkozó megszorítással korlátozhatjuk a kapcsolatban részt vevő egyedek számát. Például egy film egyedhez nem kapcsolódhat 10-nél több színész egyed a *SzerepelBenne* kapcsolatban. Az E/K-modellben a kapcsolat éleire írhatjuk rá a korlátot, ami azt jelenti, hogy a kapcsolatban részt vevő egyedek száma korlátozott az egyedhalmazra nézve. A 4.19. ábra mutatja, hogyan lehet E/K-modellben megadni ezt a megszorítást. Másik példaként tekinthetjük, hogy a nyíl annak a megszorításnak a szinonimája, hogy „≤ 1", a kerek nyíl pedig, amelyet a 4.18. ábrán láttunk, az „$= 1$" szinonimája.

4.19. ábra. A filmenkénti maximális szereplők számát szabályozó megszorítás

4.3.5. Feladatok

4.3.1. feladat. Adjuk meg a következő feladatokhoz készült E/K-diagramokban *i)* a kulcsokat és *ii)* a hivatkozásiépség-megszorításokat:

a) 4.1.1. feladat,

b) 4.1.3. feladat,

c) 4.1.6. feladat.

! **4.3.2. feladat.** Az E/K-modell kapcsolatainak ugyanúgy lehetnek kulcsai, mint az egyedhalmazoknak. Legyen R egy kapcsolat az $E_1, E_2, \ldots, E_n$ egyedhalmazok között. Ekkor R *kulcsa* az $E_1, E_2, \ldots, E_n$ egyedhalmazok attribútumainak olyan K halmaza, hogy bárhogy választunk két különböző elemet R kapcsolathalmazából, ezek az összes K-beli attribútumokon nem egyezhetnek meg. Tegyük fel, hogy $n = 2$, azaz R egy bináris kapcsolat. Minden i-re, K_i attribútumhalmaz legyen az E_i egyedhalmaz kulcsa. Adjuk meg R legkisebb lehetséges kulcsát E_1 és E_2 segítségével az alábbi feltevések mellett:

a) R sok-sok kapcsolat.

b) R sok-egy kapcsolat úgy, hogy E_2 az egy oldal.

c) R sok-egy kapcsolat úgy, hogy E_1 az egy oldal.

d) R egy-egy kapcsolat.

!! **4.3.3. feladat.** Vizsgáljuk meg ismét a 4.3.2. feladatban lévő problémát, de n bármilyen szám lehessen, ne csak 2. Használva azt az információt, hogy R milyen nyilakkal kapcsolódik az egyedhalmazokhoz, adjuk meg, hogyan lehetne megtalálni R legkisebb K lehetséges kulcsát K_i-k segítségével.

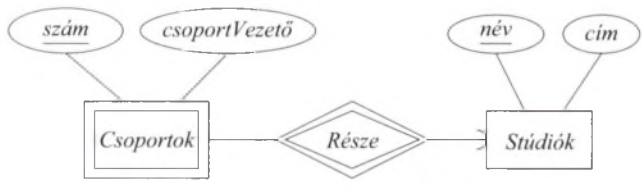
4.4. Gyenge egyedhalmazok

Esetenként előfordulhat, hogy egy egyedhalmaz kulcsában szereplő attribútumok közül néhány, vagy esetleg az összes, más egyedhalmaznak attribútuma. Az ilyen egyedhalmazokat *gyenge egyedhalmaz*oknak nevezzük.

4.4.1. A gyenge egyedhalmazok bevezetésének okai

Két elvi oka lehet a gyenge egyedhalmazok bevezetésének. Egyik, amikor az egyedhalmazok az-egy-hierarchiát alkotnak (a 4.1.11. alfejezetben tárgyaltuk). Ha például az E egyedhalmaz egyedei részei az F egyedhalmaz egyedeinek, akkor lehetséges, hogy az E-beli egyedek nevei nem egyértelműek; ha viszont az E-beli egyedhez tartozó (azt tartalmazó) F-beli egyed nevét hozzávesszük, akkor már igen. Néhány példán keresztül érzékeltetjük a problémát.

4.20. példa. Tegyük fel, hogy egy filmstúdiónál több csoport van, amelyek filmkészítéssel foglalkoznak. Egy adott stúdió esetében ezeket a csoportokat egy-egy szám jelöli, azaz 1. csoport, 2. csoport stb. Azonban egy másik stúdió is ugyanezt a jelölést használhatja csoportjaira, így a *szám* attribútum nem kulcsa a csoportoknak. Tehát ahhoz, hogy egyértelmű legyen egy csoport neve, szükség van a stúdió nevére is és a csoport számára is. Ezt ábrázolja a 4.20. ábra. A duplán keretezett téglalap jelzi a gyenge egyedhalmazt, a duplán keretezett rombusz egy sok-egy típusú kapcsolat, és a gyenge egyedhalmaz kulcsának megállapítását segíti. A jelölésekkel részletesebben a 4.4.3. alfejezetben foglalkozunk. A *Csoportok* gyenge egyedhalmaz, kulcsa a saját *szám* attribútuma és a *Stúdiók* egyedhalmaz *név* attribútuma, amelyhez a *Csoportok* egyedhalmaz a *Része* sok-egy kapcsolaton keresztül kapcsolódik. □



4.20. ábra. A csoportok gyenge egyedhalmaza és kapcsolatai

4.21. példa. A fajokat nemzetségükkel és fajukkal jelölik. Például az emberek a *Homo sapiens* fajhoz tartoznak, ahol a *Homo* a nemzetség neve, a *sapiens* a faj neve. Általában egy nemzetség fajokat tartalmaz, amelyeknek a neve a nemzetség nevével kezdődik és a faj nevével fejeződik be. Sajnos, maguk a fajok nevei nem egyértelműek; két vagy több nemzetségben is lehet ugyanolyan fajnév. Így egy fajt egyértelműen a nemzetség és a faj neve jelöli. A nemzetséghez a faj egy *Tagja* kapcsolaton keresztül kapcsolódik, a 4.21. ábrán javasolt módon. Tehát a *Fajok* egy gyenge egyedhalmaz, amelynek a kulcsa a nemzetséggel egészül ki. □

4.21. ábra. A fajok gyenge egyedhalmaza

A gyenge egyedhalmazok másik forrása a 4.1.10. alfejezetben bemutatott eljárás, amelynek segítségével a sokágú kapcsolatokat átalakítjuk binárissá[3]. Ezeknek az egyedhalmazoknak gyakran nincs is attribútuma. A kulcsuk olyan attribútumokból áll, amelyek a hozzá kapcsolódó egyedhalmazok kulcsattribútumai.

4.22. példa. A 4.22. ábrán láthatjuk a *Szerződések* kapcsoló egyedhalmazt, amely a 4.5. példában lévő *Szerződik* háromágú kapcsolatot helyettesíti. A *Szerződések* egyedhalmaznak van egy *fizetés* attribútuma, de ez az attribútum nem vesz részt a kulcsában. Egy szerződés kulcsa a stúdió nevéből, a színész nevéből, a film címéből és évéből áll. □

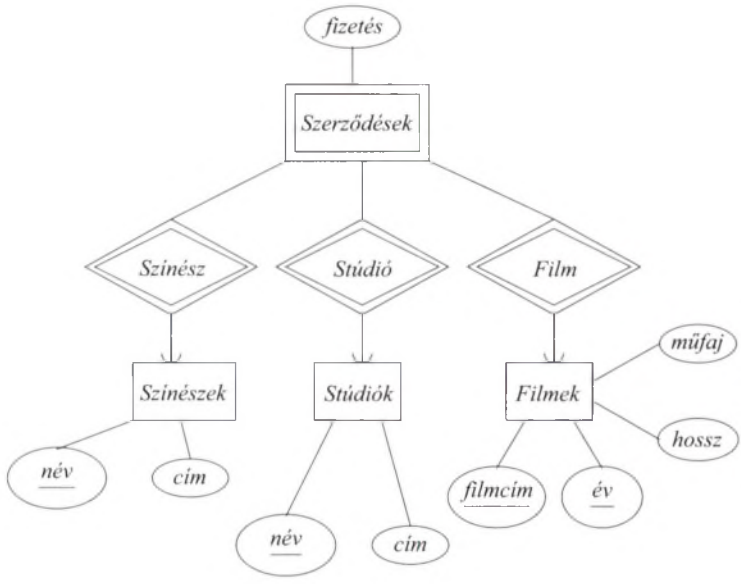
4.4.2. Gyenge egyedhalmazokra vonatkozó követelmények

Egy gyenge egyedhalmaz kulcsa nem származhat akárhonnan. Ha E egy gyenge egyedhalmaz, akkor kulcsa az alábbiakból tevődik össze:

1. Nulla vagy opcionálisan néhány saját attribútuma;

2. Más egyedhalmazok kulcsattribútumai, melyeknek E-hez sok-egy kapcsolattal kell kapcsolódniuk. E kapcsolatokat az *E kiteljesítő kapcsolat*ainak nevezhetjük (supporting relationships).

[3] Megjegyezzük, hogy bár az E/K-modellben nem kötelező megszüntetni a sokágú kapcsolatokat, egyes adatbázis-tervezési modellekben a sokágú kapcsolatokat át kell alakítani binárissá.



4.22. ábra. A kapcsoló egyedhalmaz gyenge

Ahhoz, hogy egy E és F egyedosztályt összekötő R kapcsolat E egyik kiteljesítő kapcsolata legyen, az alábbi feltételeknek kell teljesülniük:

a) R-nek bináris sok-egy kapcsolatnak[4] kell lennie, amelynek az „egy" oldala az F egyedhalmaznál van.

b) R-ben hivatkozási épségnek kell teljesülnie E-től F irányába, azaz minden E-beli egyedhez az adatbázisban léteznie kell R-en keresztül kapcsolódó F-beli egyednek. Más szavakkal, R-ből F-be lekerekített nyílnak kell vezetnie.

c) F azon attribútumai, amelyek benne vannak E kulcsában, benne vannak F kulcsában is.

d) Ha F maga is gyenge egyedhalmaz, akkor vegyük F azon kulcsattribútumait, amelyek benne vannak E kulcsában. Ezek kulcsattribútumai kell legyenek egy vagy több olyan G egyedhalmaznak is, amelyhez F kiteljesítő kapcsolattal kötődik. Rekurzíve, ha G gyenge, akkor egyes kulcsattribútumait G is máshonnan kapja és így tovább.

e) Ha több kiteljesítő kapcsolat is vezet E-ből F-be, akkor mindegyiken keresztül hozzájárulnak F kulcsának attribútumai E kulcsának kialakításához. Megjegyezzük, hogy egy E-beli e egyed más-más F-beli egyeddel

[4] Emlékezzünk arra, hogy az egy-egy kapcsolat a sok-egy kapcsolat speciális esete. Tehát amikor azt mondjuk, hogy sok-egy kapcsolat, akkor ebbe mindig beleértjük az egy-egy kapcsolatot is.

kapcsolódhat a különböző támogató kapcsolatokon keresztül. Így több különböző F-beli egyed kulcsa segíthet azonosítani az e E-beli egyedet.

Az intuitív ok, amiért ezek a feltételek szükségesek, a következő: vizsgáljunk meg egy egyedet egy gyenge egyedhalmazból, mondjuk egy csoportot a 4.20. példából. Minden csoport egyedi. Elvben meg tudjuk különböztetni a csoportokat egymástól még akkor is, ha ugyanaz a számuk, de különböző stúdióhoz tartoznak. Ez azonban elég nehéz, mert a számuk önmagában nem elegendő. Az egyetlen mód, hogy további információt kapcsoljunk a csoportokhoz, ha van egy determinisztikus eljárás, amivel további értékeket kapunk a csoport egyértelműsítéséhez. Csak a következő értékek összessége lesz egyedi egy absztrakt csoport egyedre vonatkozóan:

1. A *Csoportok* egyedhalmaz attribútumainak értéke;

2. A csoport egyed egy kapcsolatán keresztül elérhetők másik, egyedi egyedből származó, szintén egyedi értékek. Az említett kapcsolat sok-egy típusú (vagy speciális esetben egy-egy típusú) kell legyen, a vett érték pedig a másik egyedhalmaz kulcsának része.

4.4.3. Gyenge egyedhalmazok jelölése

A következő jelöléseket használjuk gyenge egyedhalmazokkal kapcsolatban.

1. Ha egy egyedhalmaz gyenge, akkor dupla kerettel jelezzük. Ez a jelölés látható a 4.20. és 4.22. ábrákon (*Csoportok*, illetve *Szerződések*).

2. A gyenge egyedhalmazok sok-egy típusú kiteljesítő kapcsolatait dupla kerettel jelöljük. Ez látható a *Része* kapcsolat esetében a 4.20. ábrán és a 4.22. ábra összes kapcsolata esetében.

3. Ha egy egyedhalmaz bármely attribútuma része a kulcsnak, akkor aláhúzással jelöljük. Például a 4.20. ábrán a csoportok száma részt vesz a kulcsban, bár nem a teljes kulcs.

Az előző jelöléseket a következő szabályban összegezhetjük:

- Ha egy E egyedhalmazt dupla kerettel jelölünk, akkor az gyenge. Ha E aláhúzott attribútumaihoz (ha vannak) hozzávesszük az E-hez dupla keretű kapcsolattal kötődő egyedhalmazok kulcsattribútumait, akkor ezek összessége E kulcsát képezi.

Ne feledjük, hogy a kettős rombusz csak a kiterjesztő kapcsolatok jelölésére szolgál. Egy gyenge egyedhalmaznak lehetnek olyan sok-egy kapcsolatai, melyek nem kiterjesztő kapcsolatok, ezeket egyszerű rombusz jelöli.

4.23. példa. A 4.22. ábrán a *Stúdió* kapcsolat nem feltétlenül kiterjesztő kapcsolata a *Szerződések*nek, ugyanis minden filmhez egyedi gyártó stúdió tartozik, melyet – az ábrán fel nem tüntetett – sok-egy kapcsolat határoz meg a *Filmek* és a *Stúdiók* között. Így tehát egy film és egy színész neve már egyértelműen azonosít egy szerződést, mely a filmbeli szereplésre vonatkozik valamely stúdióval. Jelöléseinkben ez annak felel meg, hogy a 4.22. ábrán a *Stúdió* kapcsolatot elegendő lenne hagyományos (szimpla vonalas) rombusszal feltüntetni. □

4.4.4. Feladatok

4.4.1. feladat. Egyik módja, hogy nyilvántartsuk az egyetemi hallgatókat és az általuk szerzett különböző érdemjegyeket, ha veszünk három egyedhalmazt: egyet a hallgatóknak, egyet a kurzusoknak és egyet a „kurzusfelvételeknek". A kurzusfelvételnek megfelelő egyedhalmaz egy kapcsoló egyedhalmaz a hallgatók és a kurzusok között, és nemcsak azt reprezentálja, hogy egy hallgató mely kurzusokat vette fel, hanem a kurzusra kapott érdemjegyet is. Rajzoljunk E/K-diagramot a fenti esethez, jelöljük be a gyenge egyedhalmazokat és a kulcsokat. Az érdemjegy része-e a kurzusfelvételt reprezentáló egyedhalmaz kulcsának?

4.4.2. feladat. Módosítsuk a 4.4.1. feladat megoldását úgy, hogy a hallgató összes részteljesítési érdemjegyét tartsuk nyilván, ami egy adott kurzus elvégzéséhez kapcsolódik. Ismét jelöljük a gyenge egyedhalmazokat és a kulcsokat.

4.4.3. feladat. A 4.2.6 *a)–c)* feladatokhoz készült E/K-diagramokban jelöljük be a gyenge egyedhalmazokat, kiterjesztő kapcsolataikat és a kulcsokat.

4.4.4. feladat. Rajzoljunk E/K-diagramot a következő esetekre. Az E/K-diagramokban jelöljük a gyenge egyedhalmazokat és a kulcsokat.

a) Egyedhalmazok: *Kurzusok*, *Tanszékek*. Egy kurzust egy tanszék hirdet meg, de csak egy számmal azonosítja. A különböző tanszékek adhatják ugyanazokat a számokat a kurzusaiknak. Minden tanszék neve egyedi.

! *b)* Egyedhalmazok: *Ligák*, *Csapatok*, *Játékosok*. Ligák nevei egyediek. Nincs olyan liga, amelyben két egyforma nevű csapat lenne. Nincs olyan csapat, amelyben két egyforma kódszámú játékos lenne. Azonban különböző csapatok esetében lehetnek ugyanolyan kódszámú játékosok és különböző ligákon belül lehetnek ugyanolyan nevű csapatok.

4.5. E/K-diagram átírása relációs modellé

Első megközelítésben egy E/K-terv átalakítása egy relációs adatbázissémára egyszerűen a következő:

- Minden egyedhalmazt írjunk át az attribútumaival definiált relációvá;

- Egy kapcsolatot pedig helyettesítsünk egy olyan relációval, amelynek az attribútumai a kapcsolatban álló egyedhalmazok kulcsainak felelnek meg.

Annak ellenére, hogy ez a két szabály az esetek nagy többségét kezeli, van még néhány speciális eset, amelyekkel foglalkoznunk kell, beleértve az alábbiakat:

1. A gyenge egyedhalmazokat nem lehet simán relációkká alakítani.
2. Az öröklési („az-egy") kapcsolatokat és alosztályokat óvatosan kell kezelni.
3. Néha jól tesszük, ha két relációt összevonunk, különösen egy E egyedhalmazhoz tartozó relációt egy olyan relációval, mely az E és néhány egyedhalmaz közötti sok-egy kapcsolathoz tartozik.

4.5.1. Egyedhalmazok átírása relációkká

Először vegyünk egy olyan egyedhalmazt, amely nem gyenge. A gyenge egyedhalmazok kezelésére szükséges módosításokat később a 4.5.4. alfejezetben fogjuk megnézni. Minden nem gyenge egyedhalmazhoz létrehozunk egy relációt ugyanezzel a névvel és ugyanezzel az attribútumhalmazzal.[5] Ebben a relációban nincsenek jelölve, hogy mely kapcsolatokban vesz részt az egyedhalmaz, a kapcsolatokat külön relációkban fogjuk kezelni, ahogyan ezt a 4.5.2. alfejezetben tárgyaljuk.

4.24. példa. Vegyük a 4.17. ábra *Filmek*, *Színészek* és *Stúdiók* egyedhalmazait, amelyet itt újból megismétlünk a 4.23. ábrán. A *Filmek* egyedhalmaz attribútumai a *filmcím*, *év*, *hossz*, *műfaj*. A relációra átírt eredmény, a `Filmek` reláció olyan, mint a 2.1. ábra `Filmek` relációja, amellyel a 2.2. alfejezetben kezdtünk foglalkozni.

Most vegyük a 4.23. ábra *Színészek* egyedhalmazát. Két attribútuma van, a *név* és a *cím*. Így a megfelelő `Színészek` reláció sémája `Színészek(név, cím)` lesz és egy lehetséges előfordulását az alábbi formában kapjuk:

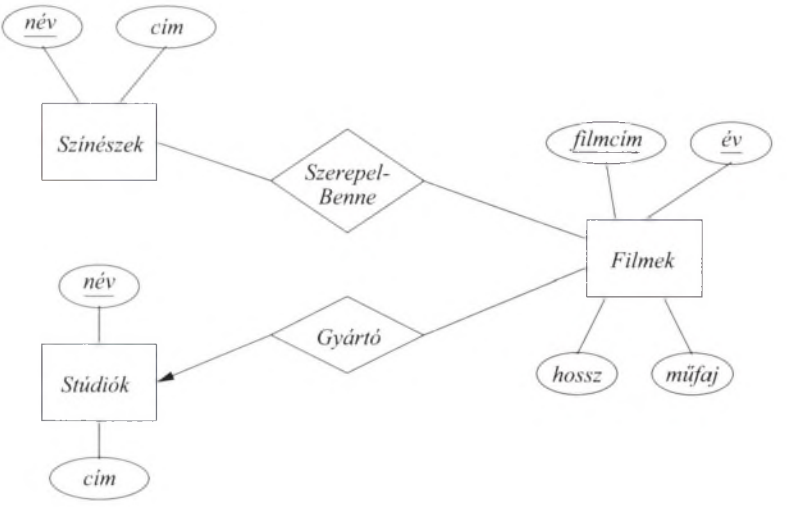
név	*cím*
Carrie Fisher	123 Maple St., Hollywood
Mark Hamill	456 Oak Rd., Brentwood
Harrison Ford	789 Palm Dr., Beverly Hills

□

4.5.2. E/K-kapcsolatok átírása relációkká

Az E/K-modellben a kapcsolatokat szintén relációkkal reprezentáljuk. Egy adott kapcsolathoz rendelt R reláció a következő attribútumokat tartalmazza:

[5] Az egyedeknek megfeleltetjük az attribútumaihoz rendelt értékekből álló sort. Az így keletkező reláció-előfordulás az (erős) egyedhalmaz kulcsa miatt nem tartalmazhat két azonos sort, tehát korrekt. *(A fordító megjegyése.)*



4.23. ábra. A filmadatbázis E/K-diagramja

1. Az R reláció sémájába belevesszük az R kapcsolatban részt vevő minden egyedhalmaz kulcsát képező attribútumot vagy attribútumokat.
2. Ha a kapcsolatnak vannak attribútumai, akkor ezek szintén az R reláció attribútumai lesznek.

Ha valamelyik egyedhalmaz többszörösen benne van a kapcsolatban, különböző szerepben, akkor minden kulcsattribútuma annyiszor kell szerepeljen, ahányféle szerepe van az egyedhalmaznak. Ilyenkor át kell neveznünk ezen attribútumokat, hogy elkerüljük a nevek ismétlődését egy sémán belül. Általánosságban, ha ugyanaz az attribútumnév kétszer vagy többször szerepel magának az R-nek és az R kapcsolatban részt vevő egyedhalmazoknak a kulcsattribútumai között, akkor át kell neveznünk, hogy elkerüljük az ismétlődést.

4.25. példa. Most vegyük a 4.23. ábra *Gyártó* kapcsolatát. Ez a kapcsolat a *Filmek* és a *Stúdiók* egyedhalmazokat kapcsolja össze. Így a `Gyártó` relációsémájában használjuk a *Filmek* kulcsát, azaz a *filmcím* és *év* attribútumokat és a *Stúdiók* kulcsát, amely a *név* attribútum:

```
Gyártó(filmcím, év, stúdióNév)
```

Az egyértelműség kedvéért itt a `stúdióNév` attribútumot választottuk, ami megfelel a *Stúdiók név* attribútumának. A reláció előfordulására egy példa a következő:

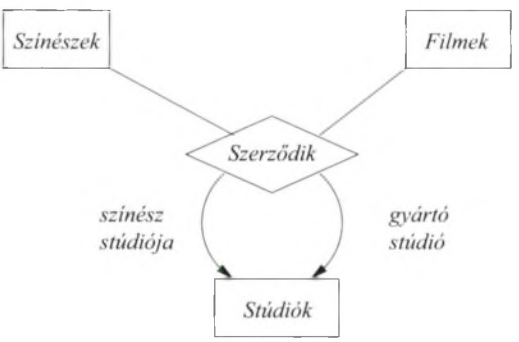
filmcím	*év*	*stúdióNév*
Csillagok háborúja	1977	Fox
Elfújta a szél	1939	MGM
Wayne világa	1992	Paramount

□

filmcím	*év*	*színészNév*
Csillagok háborúja	1977	Carrie Fisher
Csillagok háborúja	1977	Mark Hamill
Csillagok háborúja	1977	Harrison Ford
Elfújta a szél	1939	Vivien Leigh
Wayne világa	1992	Dana Carvey
Wayne világa	1992	Mike Meyers

4.24. ábra. A *SzerepelBenne* kapcsolat relációja

4.26. példa. Hasonlóan írjuk át a 4.23. ábra *SzerepelBenne* kapcsolatát relációvá, amelynek attribútumai `filmcím` és `év` (a *Filmek* kulcsa) és a `színészNév`, amely a *Színészek* egyedhalmaz kulcsa. A 4.24. ábrán a `SzerepelBenne` relációra találunk egy példát. □



4.25. ábra. A *Szerződik* kapcsolat

4.27. példa. A többirányú kapcsolatok szintén könnyen átírhatók relációkká. Tekintsük a 4.6. ábra négyirányú *Szerződik* kapcsolatát, amelyet itt a 4.25. ábrán újból bemutatunk. Ez a kapcsolat tartalmazza a színészt, a filmet és két stúdiót, az első stúdió, amelyikkel a színész szerződött, a második pedig, ahova a színész szerződést kötött a filmre. Ezt a kapcsolatot a `Szerződések` relációval reprezentáljuk, amelynek a sémáját az alábbi négy egyedhalmaz kulcsaiban szereplő attribútumok alkotják:

1. `színészNév`, a színész kulcsa.
2. `filmcím` és `év`, a film két attribútumából álló kulcsa.

3. `színészStúdió` kulcs jelzi az első stúdió nevét; korábban feltettük, hogy a `Stúdiók` egyedhalmaz kulcsa a stúdió neve.

4. `gyártóStúdió` kulcs jelzi annak a stúdiónak a nevét, amelyik elkészíti a filmet a szóban forgó színész közreműködésével.

Így tehát a séma a következőképpen alakul:

`Szerződések(színészNév, filmcím, év, színészStúdió, gyártóStúdió)`

Megjegyezzük, hogy találékonyan kellett megválasztanunk az attribútumneveket a relációsémánkban, hogy egyik attribútumként se használjuk azt, hogy „név", ugyanis akkor nem lett volna nyilvánvaló, hogy az a színészre vagy a stúdióra, és az utóbbi esetben pedig melyik stúdióra vonatkozik. Ha a *Szerződések* egyedhalmazhoz is tartoznának további attribútumok, mint például *fizetés*, akkor ezeket is hozzá kellene adnunk a `Szerződések` relációsémához. □

4.5.3. Relációk kombinációja

A relációk, melyeket az egyedhalmazok és kapcsolatok relációkká alakítása során kapunk, néha nem a legmegfelelőbbek a rendelkezésre álló adatokhoz. Egy gyakori esethez juthatunk, ha egy E egyedhalmazhoz van egy R sok-egy kapcsolat E-ből F-be. Ekkor az E-hez, illetve az R-hez tartozó mindegyik reláció sémája tartalmazni fogja az E kulcsát. Továbbá, az E reláció sémája az E nem kulcs típusú attribútumait is tartalmazza. R relációja pedig magában foglalja R összes attribútumát és az F kulcsattribútumait is. Mivel R sok-egy kapcsolat, ezért E kulcsa egyértelműen meghatározza minden attribútumának az értékét, és így a relációkat egyesíthetjük egyetlen relációba egy olyan séma használatával, amely tartalmazza:

1. E minden attribútumát,

2. F kulcsattribútumait és

3. az R kapcsolathoz tartozó összes attribútumot.

Azon E-beli e egyedek, amelyek F egyetlen egyedéhez sem kapcsolódnak, a hozzájuk tartozó sorban, a 2. és 3. típusú attribútumaikban nullértéket tartalmaznak.

4.28. példa. A 4.25. példában relációvá alakított *Gyártó* egy sok-egy kapcsolat a *Filmek* és *Stúdiók* között az általunk használt filmes példában. A *Filmek* egyedhalmazból kapott relációt már megtárgyaltuk a 4.24. példában. Ezen két relációt egyesíthetjük egyetlen relációba, az összes attribútumuk egy relációsémába foglalásával. Ezt a helyzetet szemlélteti a 4.26. ábrán szereplő reláció. □

filmcím	*év*	*hossz*	*műfaj*	*stúdióNév*
Csillagok háborúja	1977	124	sci-fi	Fox
Elfújta a szél	1939	239	dráma	MGM
Wayne világa	1992	95	vígjáték	Paramount

4.26. ábra. A Filmek reláció kombinációja a Gyártó relációval

Az, hogy ilyen módon egyesítsük-e vagy sem a relációkat, valójában döntés kérdése. Van viszont néhány előnye annak, hogy az E egyedhalmaz kulcsától függő összes attribútum egy helyen fordul elő, még akkor is, ha van néhány sok-egy kapcsolat E-ből különböző egyedhalmazokba. Jóval hatékonyabb például, ha a lekérdezés válaszának attribútumai egy relációban szerepelnek, mintha több relációból származnának. Néhány E/K-terven alapuló tervezőrendszer automatikusan egyesíti az ilyen relációkat.

Másrészről elképzelhető, hogy valaki az E relációt egyesíteni szeretné egy olyan R kapcsolathoz tartozó relációval, ami ugyan tartalmazza E-t, de nincs sok-egy kapcsolat az E-ből valamilyen más egyedhalmazba. Ez azonban kockázatos próbálkozás, mivel gyakran redundanciához vezethet, ahogy azt a következő példa is mutatja.

4.29. példa. Annak megértéséhez, hogy mi romolhat el, tekintsük a 4.26. ábrán lévő reláció egyesítését a *SzerepelBenne* sok-sok kapcsolathoz kapott relációval, amely a 4.24. ábrán szerepelt. Ekkor az egyesített reláció úgy néz ki, mint a 3.2. ábrán látható reláció, melyet itt a 4.27. ábrán megismételtünk. Ahogy a 3.3.1. alfejezetben már tárgyaltuk, ez a reláció anomáliákkal terhelt, melyeket a normalizálás folyamán kell megszüntetnünk. □

filmcím	*év*	*hossz*	*műfaj*	*stúdióNév*	*színészNév*
Csillagok háborúja	1977	124	sci-fi	Fox	Carrie Fisher
Csillagok háborúja	1977	124	sci-fi	Fox	Mark Hamill
Csillagok háborúja	1977	124	sci-fi	Fox	Harrison Ford
Elfújta a szél	1939	231	dráma	MGM	Vivien Leigh
Wayne világa	1992	95	vígjáték	Paramount	Dana Carvey
Wayne világa	1992	95	vígjáték	Paramount	Mike Meyers

4.27. ábra. A Filmek reláció a szereplőkre vonatkozó információval kiegészítve

4.5.4. Gyenge egyedhalmazok kezelése

Ha egy E/K-diagramban gyenge egyedhalmaz van, akkor három dolgot kell eltérő módon tennünk.

1. Magához a W gyenge egyedhalmazhoz tartozó relációnak tartalmaznia kell nem csupán W attribútumait, hanem más egyedhalmazok kulcsattribútumait is, amelyek segítik W kulcsának kialakítását. Ezeket a segítő

egyedhalmazokat könnyen felismerhetjük arról, hogy a W-hez kiteljesítő (dupla keretes rombusszal jelölt) kapcsolattal kötődnek.

2. Bármely olyan kapcsolat relációjában, amelyben a W gyenge egyedhalmaz részt vesz, W kulcsának az összes kulcsattribútumát használjuk, azokat az egyéb egyedhalmazokbeli attribútumokat is beleértve, amelyek hozzájárultak W kulcsához.

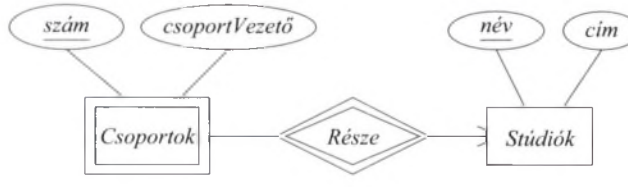
3. A kiteljesítő (dupla keretes) kapcsolatokat a W gyenge egyedhalmaztól valamely más egyedhalmazba, vagyis amelyek hozzájárulnak W kulcsához, egyáltalán nem kell relációkká átírnunk. Ez – mint ahogy a 4.5.3. alfejezetben tárgyaltuk – amiatt van így, mert egy R sok-egy kapcsolat relációjának attribútumai vagy már eleve W relációjának attribútumai, vagy pedig (R saját attribútumai esetén) kombinálhatók W relációsémájával.

Természetesen amikor további attribútumokat veszünk fel egy gyenge egyedhalmaz kulcsába, akkor vigyáznunk kell, hogy ne használjuk ugyanazt a nevet kétszer. Ha szükséges, nevezzük át az összes ilyen attribútumot.

4.30. példa. Tekintsük a 4.20. ábrán szereplő *Csoportok* gyenge egyedhalmazt, amelyet a 4.28. ábrán megismétlünk. Ebből a diagramból három relációt kapunk, amelyeknek a sémái:

```
Stúdiók(név, cím)
Csoportok(szám, stúdióNév, csoportVezető)
Része(szám, stúdióNév, név)
```

Az első relációt, a `Stúdiók`-at természetes módon írtuk át az azonos nevű egyedhalmazból. A második reláció, a `Csoportok` gyenge egyedhalmazból származik. Ennek a relációnak az attribútumai a *Csoportok* kulcsattribútumai, és egy nem kulcs attribútum, a *csoportVezető*. A `Csoportok` relációban a `stúdióNev`-et választottuk attribútumként, amely megfelel a *Stúdiók* egyedhalmaz *név* attribútumának.



4.28. ábra. A *Csoportok*, avagy példa gyenge egyedhalmazra

A harmadik reláció, az `Része`, az azonos nevű kapcsolatból származik. Mint mindig, egy E/K-kapcsolatot a relációs modellben olyan relációval reprezentálunk, amelynek sémája a kapcsolt egyedhalmazok kulcsaiból áll. Ebben az esetben az `Része` attribútumai a `szám` és `stúdióNév`, a *Csoportok* gyenge egyedhalmaz kulcsa, és a `név` attribútum, amely a *Stúdiók* kulcsa. Megjegyezzük, hogy

mivel az *Része* sok-egy kapcsolat, a stúdiót azonosító `stúdióNév` biztos, hogy megegyezik ugyanahhoz a stúdióhoz tartozó `név`-vel.

Például tegyük fel, hogy a hármas számú csoport (#3) a Disney stúdió csoportjainak egyike. Ekkor az *Része* kapcsolathalmaz tartalmazza a következő párt:

(Disney csoport #3, Disney)

Ez a pár az alábbi sort adja az `Része` relációhoz:

(3, Disney, Disney)

Vegyük észre, hogy e sor `stúdióNév` és `név` attribútumai megegyeznek, mint ahogy meg is kell egyezniük. Következésképpen „egybeolvashatjuk" az `Része` reláció `stúdióNév` és `név` attribútumait, így a következő, egyszerűbb sémát kapjuk:

```
Része(szám, név)
```

Tulajdonképpen el is hagyhatjuk az `Része` relációt, mivel attribútumhalmaza így része a `Csoportok` reláció attribútumhalmazának. □

A 4.30. példában megfigyelt jelenség, vagyis hogy a kiteljesítő kapcsolathoz nem szükséges relációt megadni, általában is igaz a gyenge egyedhalmazokra. A gyenge egyedhalmazokra vonatkozóan a következő módosított szabályokat állíthatjuk fel.

- Ha W egy gyenge egyedhalmaz, akkor készítsünk W számára egy olyan relációt, amelynek sémája a következőkből tevődik össze:
  1. W attribútumaiból,
  2. W kiteljesítő kapcsolatainak attribútumaiból,
  3. azon E egyedhalmazok kulcsattribútumaiból, melyekhez W kiteljesítő kapcsolattal kötődik (ezek sok-egy típusú kapcsolatok).

  A névkonfliktusok elkerülése céljából, ha szükséges, nevezzük át az attribútumokat.
- *Ne* készítsünk relációkat W kiteljesítő kapcsolataihoz.

4.5.5. Feladatok

4.5.1. feladat. Alakítsuk át a 4.29. ábra E/K-diagramját relációs adatbázissémává.

Relációk részhalmazsémákkal

A 4.30. példa alapján azt gondolhatnánk, hogy amennyiben egy R relációnak van egy olyan attribútumhalmaza, amely egy másik S reláció attribútumainak részhalmaza, akkor R eltávolítható lesz. Ez viszont nem teljesen igaz. R tartalmazhat S-ben nem szereplő információkat, ugyanis S további attribútumai nem teszik lehetővé számunkra, hogy egy R-beli sorból S-beli sorra következtessünk.

Példaként tekintsük azt, hogy az adóhatóság a potenciális adófizetők neveit és társadalombiztosítási számaikat tartalmazó `Emberek(név, TAJ#)` relációt akarja fenntartani még azokra is, akiknek nincs bevétele, illetve adó-visszaigénylést sem adtak be. Ehhez szükségük lehet még egy `AdóFizetők(név, TAJ#, összeg)` relációra, ami tartalmazza minden olyan személy befizetett adójának mértékét, aki adott be visszaigénylést az adott évben. Az `Emberek` sémája részhalmaza az `AdóFizetők` sémájának, mégis lehetnek olyan értékei, nevezetesen olyan társadalombiztosítási számokkal rendelkező egyének, akik szerepelnek az `Emberek`-ben, de az `AdóFizetők`-ben nem.

Valójában még azonos attribútumhalmazok esetén is eltérő lehet a szemantika, ezért ezeknek a sorait sem fésülhetjük össze. A példa kedvéért vegyük a `Színészek(név, cím)` és `Stúdiók(név, cím)` relációkat. A séma ugyan egyformának tűnik, mégsem rakhatjuk a színészek sorait a stúdiók sorai közé (vagy fordítva).

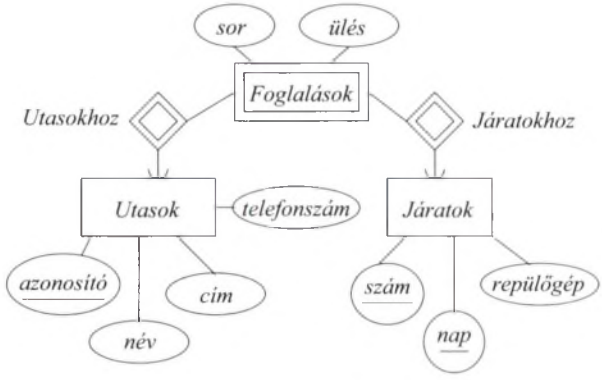
Másrészről, ha a két reláció egy gyenge egyedhalmaz konstrukciójából származik, akkor nem lehetnek a szűkebb attribútumhalmazzal bíró relációnak további értékei. Ennek oka, hogy a kiteljesítő kapcsolatból származó relációsorok egy az egyben megegyeznek a gyenge egyedhalmazból származó relációsorokkal. Ilyenkor rutinszerűen az egyik relációt eltávolítjuk.

! **4.5.2. feladat.** A 4.29. ábrán látható *Foglalások* gyenge egyedhalmazhoz más E/K-diagram is elképzelhető. Vegyük észre, hogy egy foglalást egyértelműen azonosít a járat száma, dátuma, a sor és az ülés száma. Így tehát az utas nem játszik szerepet egy foglalás azonosításában.

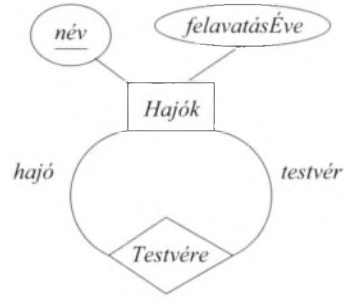
a) Módosítsuk a 4.29. ábrát a fenti nézőpont szerint.

b) Alakítsuk át az *a)* feladat megoldását relációkká. Ugyanazt az eredményt kapjuk-e, mint a 4.5.1. feladatban?

4.5.3. feladat. A 4.30. ábra E/K-diagramja hajókat reprezentál. *Testvérek*nek hívjuk azokat a hajókat, amelyeket ugyanazon terv alapján építettek. Alakítsuk át ezt a diagramot relációs adatbázissémává.



4.29. ábra. A repülőgépes helyfoglalás E/K-diagramja



4.30. ábra. A testvérhajók egy lehetséges E/K-diagramja

4.5.4. feladat. Alakítsuk át a következő E/K-diagramokat relációs adatbázissémákra:

a) 4.22. ábra

b) 4.4.1. feladat válasza

c) 4.4.4 *a)* feladat válasza

d) 4.4.4 *b)* feladat válasza

4.6. Osztályhierarchia átalakítása relációkká

Egyedhalmazok öröklési („az-egy" típusú) kapcsolattal megadott hierarchiája esetén több stratégia közül választhatunk a relációkká alakításhoz. Emlékeztetőül a következőket feltételezzük:

- A hierarchiának van egy gyökéregyedhalmaza.
- Ez az egyedhalmaz rendelkezik kulccsal, amely a hierarchiában reprezentált összes egyed azonosítására szolgál.
- Egy adott egyednek lehetnek *komponens*ei, amelyek a hierarchiában különféle egyedhalmazokhoz tartozhatnak. Ezek az egyedhalmazok a struktúra tetszőleges részfáját alkothatják, melynek a gyökéregyedhalmazt tartalmaznia kell.

Az átalakítási stratégiák elvei az alábbiak:

1. *Az E/K szempontjainak követése.* A hierarchiában lévő összes E egyedhalmazra külön-külön hozzunk létre egy relációt, amely tartalmazza a gyökérben lévő kulcsattribútumokat és E minden attribútumát.
2. *Az egyedek egy egyszerű osztályhoz tartozó objektumkénti kezelése.* Az összes, a gyökeret tartalmazó lehetséges részfára hozzunk létre egy relációt, melynek a sémája magában foglalja a részfában szereplő összes egyedhalmaznak az attribútumait.
3. *Nullértékek használata.* A hierarchiában szereplő összes egyedhalmaz összes attribútumával hozzunk létre egyetlen relációt. Minden egyedet egy sor fog reprezentálni, amely nullértékeket vesz fel azon oszlopokon, ahol az egyed nem rendelkezik attribútumértékkel. (A nullérték az érték hiányát jelzi.)

Lépésenként az összes megközelítést megvizsgáljuk.

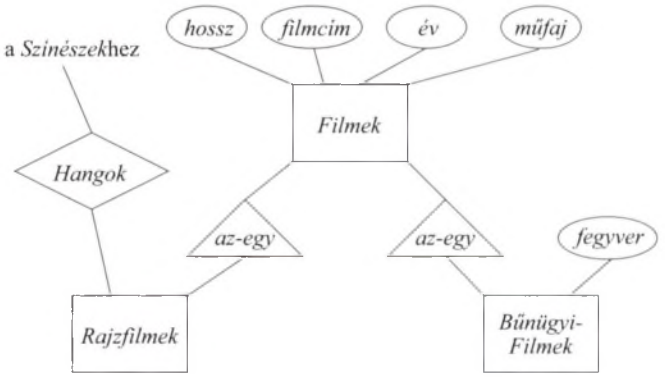
4.6.1. E/K-típusú átalakítás

Az első megközelítésünkben a megszokott módon minden egyedhalmazhoz készítünk egy relációt. Ha az E egyedhalmaz nem a hierarchia gyökércsúcsa, akkor az E-hez tartozó reláció tartalmazza a gyökér kulcsattribútumait is. Ha E emellett szerepel egy kapcsolatban is, akkor a kapcsolatban is ezeket a kulcsattribútumokat fogjuk használni E egyedeinek azonosításához.

Megjegyezzük, hogy annak ellenére, hogy az öröklési (az-egy) kapcsolatot is kapcsolatként kezeltük, mégsem hasonlít a többi kapcsolatra abban az értelemben, hogy egy egyszerű egyed komponenseit kapcsolja össze, nem pedig egyedeket egymással. Az öröklési kapcsolathoz tehát nem készíthető reláció.

4.31. példa. Tekintsük a 4.10. ábrán lévő hierarchiát, amelyet a 4.31. ábrán újra bemutatunk. A relációknak a hierarchiában szereplő négy különböző egyedet kell reprezentálniuk:

1. `Filmek(filmcím, év, hossz, műfaj)`. A 4.24. példában már megállapított reláció, ahol minden film egy sorral van ábrázolva.



4.31. ábra. A filmek hierarchiája

2. `BűnügyiFilmek(filmcím, év, fegyver)`. Az első két attribútum a filmek kulcsa, az utolsó pedig az egyedhalmaz saját attribútuma. A bűnügyi filmekhez mind itt, mind a `Filmek`-ben is tartozik egy-egy sor.

3. `Rajzfilmek(filmcím, év)`. Ez a reláció a rajzfilmek egy halmazát jelenti. A filmek kulcsán kívül nincs más attribútuma, mivel a rajzfilmekhez tartozó kiegészítő információit a *Hangok* kapcsolat tartalmazza. A rajzfilmekhez itt is és a `Filmek`-ben is tartozik egy-egy sor.

Megjegyezzük, hogy a negyedik típusú filmnek – amely rajzfilm és egyben bűnügyi film is – mind a három relációban van sora.

Szükségünk lehet ezeken kívül még egy `Hangok(filmcím, év, színészNév)` relációra, amely a *Színészek* és a *Rajzfilmek* közötti kapcsolatot valósítja meg. Ebben az utolsó attribútum a *Színészek* kulcsa lesz, az első kettő pedig a *Rajzfilmek* kulcsát alkotja majd.

A *Roger nyúl a pácban* című filmhez például mind a négy relációban tartozik sor. Az alapinformációi a `Filmek` relációban, a fegyver a `BűnügyiFilmek` relációban, a hangjukat adó színészek pedig a `Hangok` relációban találhatók.

Vegyük észre, hogy a `Rajzfilmek` reláció sémája a `Hangok` reláció sémájának részhalmaza. A legtöbb esetben egy `Rajzfilmek`-hez hasonló relációt eltávolítanánk, mivel nem szerepel benne olyan információ, amit a `Hangok` ne tartalmazna. Ugyanakkor lehetnek néma rajzfilmek is az adatbázisban, melyekhez nem tartoznak hangok, és így elveszítenénk azt az információt, hogy ezek rajzfilmek. □

4.6.2. Egy objektumorientált megközelítés

Egy másik stratégia lehet az öröklési hierarchia relációvá alakítása során, ha felsoroljuk a hierarchia összes lehetséges részfáját. Ezek mindegyikére létrehozunk egy-egy relációt, amely azon egyedeket reprezentálja, amelyeknek pontosan az

adott részfában vannak komponensei. A reláció sémája az összes, a részfában szereplő egyed komponenseit tartalmazza. Erre objektumorientált megközelítésként hivatkozunk, mivel az a feltevés motiválta, hogy az egyedek egy és csak egy osztályhoz tartozó „objektumok" legyenek.

4.32. példa. Tekintsük a 4.31. ábra hierarchiáját. A gyökeret is beleértve négy lehetséges részfa lehetséges:

1. A *Filmek* önmagában.
2. Csak a *Filmek* és a *Rajzfilmek*.
3. Csak a *Filmek* és a *BűnügyiFilmek*.
4. Mindhárom egyedhalmaz.

Mind a négy „osztályhoz" relációt kell készítenünk. E sémákban most van némi azonosság, mivel csak a bűnügyi filmekhez tartozik saját speciális attribútum:

```
Filmek(filmcím, év, hossz, műfaj)
FilmekRF(filmcím, év, hossz, műfaj)
FilmekBF(filmcím, év, hossz, műfaj, fegyver)
FilmekRFBF(filmcím, év, hossz, műfaj, fegyver)
```

Ha a *Rajzfilmek*nek saját attribútumai lennének, akkor mind a négy reláció különböző attribútumhalmazzal rendelkezne. Mivel jelen esetben nem ez a helyzet, ezért kombinálhatjuk a `Filmek`-et a `FilmekRF`-fel (azaz létrehozhatunk egy relációt a nem bűnügyi filmekre), illetve a `FilmekBF`-et a `FilmekRF`-fel (azaz létrehozhatunk egy relációt minden bűnügyi filmre), habár így elveszítjük azt az információt, hogy melyik film lesz rajzfilm.

Azt is meg kell még fontolnunk, hogyan kezeljük a *Rajzfilmek* és *Színészek* közötti *Hangok* kapcsolatot. Ha a *Hangok* sok-egy kapcsolatban álltak volna a *Rajzfilmek*kel, akkor a `FilmekRF`-hez és a `FilmekRFBF`-hez hozzáadhattunk volna egy *hang* attribútumot, amely ábrázolhatná a *Hangok* kapcsolatot, és ebből adódóan mind a négy reláció különböző lenne. Viszont a *Hangok* sok-sok típusú, ezért önálló relációkat kell létrehoznunk ehhez a kapcsolathoz. Ennek a sémája – mint mindig – egy kulcsattribútumon keresztül kapcsolódik az egyedhalmazokhoz, és így a következő séma alkalmas lesz erre:

```
Hangok(filmcím, év, színészNév)
```

Felmerülhet, hogy szükség van-e két ilyen reláció létrehozására, ahol az egyik a nem bűnügyi témájú rajzfilmekkel, a másik pedig a bűnügyi témájú rajzfilmekkel kapcsolja össze a szinkronhangokat. Ennek viszont jelen esetben semmi haszna sem lenne. □

4.6.3. Nullértékek használata relációk egyesítéséhez

Még egy megközelítés maradt arra, hogy egyedhalmazok hierarchiáját relációs modellben ábrázoljuk. Ha megengedjük a relációkban (az SQL nullértékéhez hasonló) `NULL` érték használatát, akkor az egyedhalmazok teljes hierarchiáját egyetlen egyszerű reláció segítségével ábrázolhatjuk. Ez a reláció a hierarchia összes egyedhalmazához tartozó minden attribútumot tartalmaz. Egy egyedet pedig egy sorral ábrázolunk. Ennek a sornak azon értékei, amelyek nem definiáltak az adott egyedre, `NULL` értéket vesznek fel.

4.33. példa. Ha ezt a megközelítést alkalmazzuk a 4.31. ábra diagramjára, akkor a következő sémájú relációt kapjuk:

```
Filmek(filmcím, év, hossz, műfaj, fegyver)
```

A nem bűnügyi filmek sorai a `fegyver` komponensben tartalmazhatnak `NULL` értéket. Szükségünk lehet még a 4.32. példához hasonlóan egy `Hangok` relációra is, amely a rajzfilmeket kapcsolja össze a szinkronizálást végző színészekkel. □

4.6.4. A megközelítések összehasonlítása

Mindhárom tárgyalt megközelítésnek, amelyekre a továbbiakban „E/K-elvű", „objektumelvű", illetve „nullértékes"-ként hivatkozunk, vannak előnyei és hátrányai is. A most következőkben listaszerűen felsoroljuk a legfontosabb tényezőket:

1. A válaszadás költséges a sok relációt érintő lekérdezésekre, ezért inkább azt részesítjük előnyben, amikor egy lekérdezés eredményéhez minden szükséges attribútum egyetlen relációban szerepel. Mivel a nullértékes megközelítés csak egy, minden attribútumot magában foglaló relációt használ, így ebből a szempontból ez rendelkezik ezzel az előnnyel. A másik két megközelítésnek más típusú lekérdezések esetén van előnye. Például:

   a) A „Mely 2008-as filmek voltak 150 percnél hosszabbak?" típusú kérdést a 4.31. példa E/K-elvű megközelítéséből származó `Filmek` relációjából közvetlenül megválaszolhatjuk. A 4.32. példa objektumelvű megközelítése esetén viszont a `Filmek`, a `FilmekRF`, a `FilmekBF` és a `FilmekRFBF` vizsgálatára is szükségünk lesz, mivel egy hosszú film ezek bármelyikében benne lehet.

   b) Másrészről viszont a „Milyen fegyvereket használtak a 150 percnél hosszabb rajzfilmekben?" típusú kérdés esetén az E/K-elvű megközelítés használata gondot okozhat. Hozzá kell férnünk a `Filmek`-hez a 150 percnél hosszabb filmek meghatározásához, a `Rajzfilmek`-hez annak ellenőrzésére, hogy egy film rajzfilm-e, illetve a `BűnügyiFilmek`-hez a fegyver megkereséséhez. Az objektumelvű megközelítés esetén pedig csak a `FilmekRFBF`-hez kell hozzáférni, ahol minden szükséges információ megtalálható.

2. Nem szeretnénk túl sok relációt használni. Így itt újra a nullértékes módszer jeleskedik, mivel annál csak egy relációra van szükség. A másik két módszer között viszont vannak különbségek, ugyanis míg az E/K-elvű megközelítésnél a hierarchiában szereplő minden egyedhalmaz csak egyetlen relációt használt, addig az objektumelvű megközelítésnél, ha egy gyökér és n darab gyerek csúcs (összesen $n + 1$ darab egyedhalmaz) adott, akkor 2^n különböző egyedosztályt kapunk, és ennyi relációra lenne szükségünk.

3. Minimalizálni szeretnénk a területet és elkerülni az információk ismétlődését. Mivel az objektumelvű megközelítés minden egyedhez egyetlen sort használ, és ez a sor csak az egyeden értelmezett attribútumokat tartalmazza, így ez a megközelítés biztosítja a lehetséges minimális helyfoglalást. A nullértékes megközelítés is hasonló módon egyedenként egy sort tárol, viszont ezek a sorok „hosszúak" abban az értelemben, hogy minden attribútumra van komponensük, akár értelmezettek az adott egyedre, akár nem. Ha sok egyedhalmaz van a hierarchiában, illetve ezen egyedhalmazok attribútumai sokban különböznek egymástól, akkor a nullértékes megközelítés használata esetén nagy, kihasználatlan tárterület lesz lekötve. Az E/K-elvű módszer több sort rendel egy-egy egyedhez, viszont csak a kulcsattribútumokat ismétli. Azaz ez a módszer akár kevesebb vagy több területet is használhat, mint a nullértékes.

4.6.5. Feladatok

4.6.1. feladat. Alakítsuk át a 4.32. ábra E/K-diagramját relációs adatbázissémává, a különböző megközelítésekkel:

a) E/K-elvű,

b) objektumelvű,

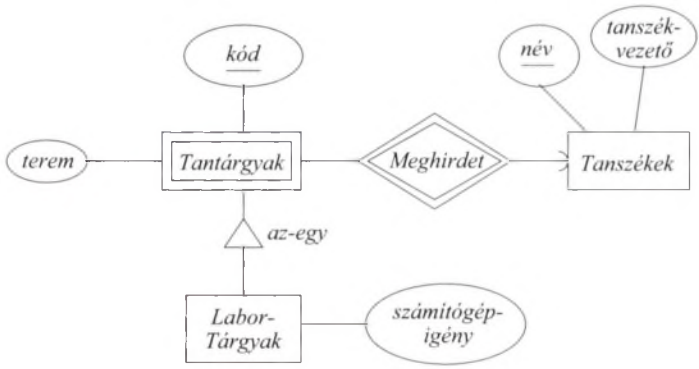
c) nullértékes módszerrel.

! **4.6.2. feladat.** Alakítsuk át a 4.33. ábra E/K-diagramját relációs adatbázissémává, a különböző megközelítésekkel:

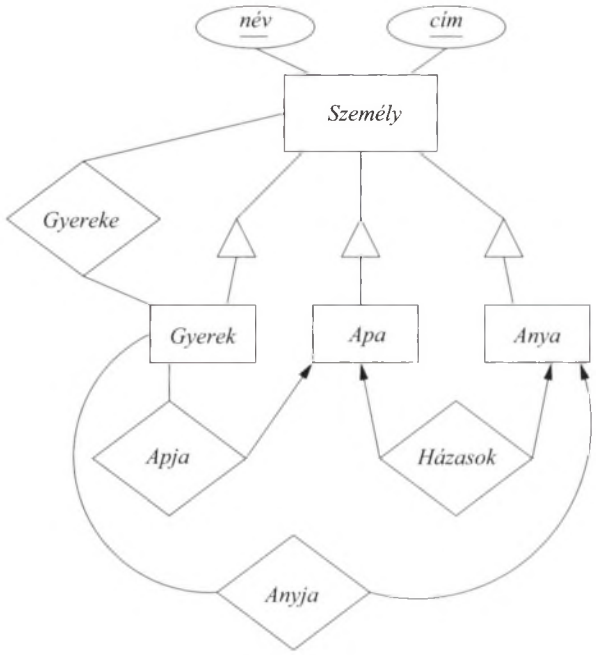
a) E/K-elvű,

b) objektumelvű,

c) nullértékes módszerrel.



4.32. ábra. A 4.6.1. feladathoz tartozó E/K-diagram



4.33. ábra. A 4.6.2. feladathoz tartozó E/K-diagram

4.6.3. feladat. A 4.1.7. feladat megoldásaként kapott E/K-diagramot alakítsuk át relációs adatbázissémává:

a) E/K-elvű,

b) objektumelvű,

c) nullértékes módszerrel.

! **4.6.4. feladat.** Tegyük fel, hogy adott e darab egyedhalmaz öröklési hierarchiája. Minden egyedhalmaznak van a darab attribútuma és ezek közül k darab a gyökérben az összes egyedhalmaz kulcsát alkotja. Adjunk képletet, hogy relációsémákká alakításkor *i)* legalább, illetve legfeljebb hány reláció keletkezik, *ii)* összességében legalább, illetve legfeljebb hány komponensből álló sor(ok)ra képződik egy egyed, ha az alábbi átalakítási módszert használjuk:

a) E/K-elvű,

b) objektumelvű,

c) nullértékes módszerrel.

4.7. Bevezetés az UML-be

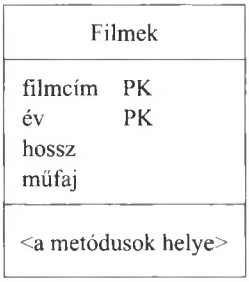
Az UML-t (*Unified Modeling Language – Egységesített Modellező Nyelv*) eredetileg az objektumorientált szoftvertervezési stílushoz igazodó grafikus jelölőkészletnek fejlesztették ki. Később kibővítették olyan módosításokkal, amelyekkel az adatbázistervek leírásának népszerű eszköze lett. Az UML-nek ezt a részét fogjuk tanulmányozni. A többszörös kapcsolatok kivételével az UML lehetőségei sokban hasonlítanak az E/K-modell lehetőségeihez. Az UML is lehetőséget ad az egyedhalmazok – mint valódi osztályok – kezelésére, azok metódusaival és adataival. A 4.34. ábrán összefoglaljuk az E/K- és az UML-modellek közös koncepcióját az általuk használt terminológiai különbségekkel.

UML	*E/K-modell*
Osztály	Egyedhalmaz
Társítás	Bináris kapcsolat
Társításosztály	A kapcsolat attribútumai
Alosztály	Osztályhierarchia
Aggregáció (összesítés)	Sok-egy kapcsolat
Kompozíció (összeállítás)	Sok-egy kapcsolat hivatkozásiépség-megszorítással

4.34. ábra. Az UML- és az E/K-terminológiák összehasonlítása

4.7.1. UML-osztályok

Az UML-ben az osztály az E/K-modell egyedhalmazához hasonló fogalom. A jelölése ugyanakkor meglehetősen eltér attól. A 4.35. ábra a fejezet állandó filmes példájában már látott, *Filmek* E/K-egyedhalmaznak megfelelő UML-osztályt mutatja.



4.35. ábra. A *Filmek* osztály megadása UML-ben

Az osztály doboza három részre van osztva. A felső az osztály nevét tartalmazza. A középsőben vannak az attribútumok, ezek az osztály tartalmazott változóihoz hasonlók. A *Filmek* osztályunkban a *filmcím*, *év*, *hossz* és *műfaj* attribútumokat használjuk.

Az alsó rész a metódusoké. Sem az E/K-modellben, sem a relációs modellben nincsenek metódusok. Ugyanakkor a metódusok használata egy fontos elgondolás, sok modern relációs rendszerben is megjelenik, az ilyen rendszereket „objektumrelációs" rendszereknek nevezik (lásd a 10.3. alfejezetben).

4.34. példa. Gondolhatunk például egy *hosszÓrákban*() tárolt metódus bevezetésére. Az UML-specifikáció a metódusokról csak a bemenő argumentumaik és a visszatérési értékük típusának megadását követeli meg. A metódus visszaadhatja például a *hossz*/60.0 értéket, de a metódus belső szerkezetéről semmit nem kell megadnunk. □

Ebben az alfejezetben az adatbázisterveinkben nem fogunk metódusokat használni. Így a továbbiakban az UML-osztályokat megadó dobozokat csak két részre osztjuk, amelyeket az osztály nevének és attribútumainak megadására használunk.

4.7.2. Az UML-osztályok kulcsai

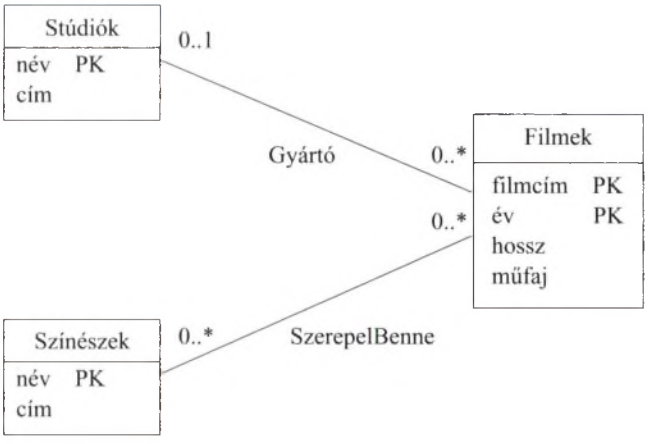
Ahogy az egyedhalmazokhoz, úgy az UML-osztályokhoz is meg kell adnunk egy kulcsot. A kulcsot képező attribútumok neve után a PK (primary key = elsődleges kulcs) betűket kell írnunk. Több attribútum vagy attribútumhalmaz kulcsként való megjelölésére nincs megfelelő mód.

4.35. példa. A 4.35. ábrán a szokásos feltételezésünkkel éltünk, nevezetesen, hogy a *Filmek* kulcsát a *filmcím* és *év* együtt képezik. Figyeljük meg, hogy a PK csak ezen attribútumok sorában fordul elő, másutt nem. □

4.7.3. Társítások

Az UML-ben az osztályok közötti bináris kapcsolatot *társítás*nak nevezzük. A többszörös kapcsolatoknak az UML-ben nincs megfelelője. A többszörös kapcsolatokat inkább bináris kapcsolatokká bontják úgy, ahogy ezt a 4.1.10. alfejezetben már javasoltuk. A társítás megvalósítása pontosan úgy történik, ahogy azt a 4.1.5. alfejezetben a kapcsolathalmazokkal leírtuk. A társítás objektumpárok halmaza, egy-egy objektum az összekapcsolt osztályokból.

Két osztály közötti UML-társítást a kettő közötti vonal megrajzolásával jelezzük, és a vonalnak nevet adunk. A nevet lehetőség szerint a vonal alá írjuk. Példaként a 4.36. ábrán a 4.17. ábra E/K-diagramjának megfelelő UML-diagramot látjuk. Két társítást adtunk meg, nevük: *SzerepelBenne* és *Gyártó*; az első a *Filmek*et a *Színészek*kel, a második a *Filmek*et a *Stúdiók*kal kapcsolja össze.



4.36. ábra. Filmek, színészek és stúdiók megadása UML-ben

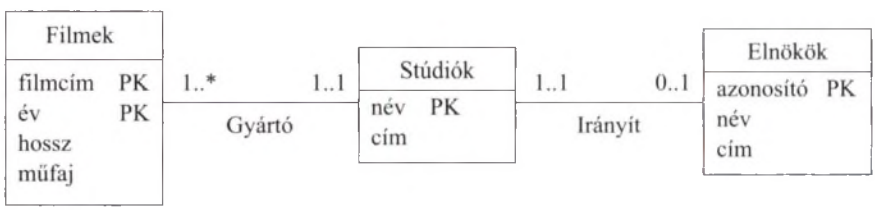
Minden társítás tartalmaz a kapcsolatban részt vevő, az adott osztály objektumai számára vonatkozó megszorítást is. Az ilyen megszorítást a kapcsolat mindkét végén $m..n$ alakú címkékkel jelezzük. A címke jelentése az, hogy a kapcsolat másik végén lévő objektumok a kapcsolat ennél a végénél legalább m és legfeljebb n objektumhoz kapcsolódnak. Továbbá:

- Ha n helyén $*$ áll, azaz: $m..*$, akkor jelentése: „korlátlan". Ez azt jelenti, hogy a kapcsolatban részt vevő objektumok számára nézve nincs felső korlát.

- Ha az *m..n* helyett egyedül egy ∗ áll, akkor ez 0..∗ jelentésű, vagyis nincs megkötés az objektumok számára nézve.

- Ha egy társítást jelképező vonal végénél egyáltalán nincs ilyen címke, akkor úgy tekintjük, mintha 1..1 állna ott, a megszorítás tehát „pontosan egy" jelentésű.

4.36. példa. A 4.36. ábrán mindkét társítás *Filmek*nél lévő végénél 0..∗ címkét látunk. Ez azt jelenti, hogy a színész 0 vagy több filmben szerepel, és a stúdió 0 vagy több filmet gyárt. A *SzerepelBenne* társítás *Színészek* végénél is 0..∗ van, ami azt mondja nekünk, hogy a filmnek tetszőleges számú szereplője lehet. Ugyanakkor a *Gyártó* társítás *Stúdiók* végénél a 0..1 azt jelenti, hogy 0 vagy 1 stúdió vesz részt a társításban. Megjegyezzük, hogy ezt a megszorítást éppen a 4.17. ábra E/K-diagramján a *Stúdiók* felé mutató nyíl fejezi ki. □



4.37. ábra. Hivatkozási épség kifejezése UML-ben

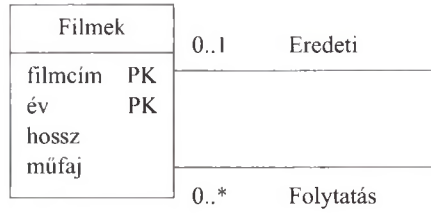
4.37. példa. A 4.37. ábra UML-diagramja a 4.18. ábra E/K-diagramját kívánja tükrözni. Most a 4.36. példától némileg különböző feltevéseket használunk a társításban részt vevő filmek és stúdiók számára nézve. A *Gyártó* társítás *Filmek* végénél az 1..∗ azt jelenti, hogy minden stúdiónak legalább egy filmet kellett gyártania (különben valójában nem tekinthető stúdiónak). Felső korlát nincs arra, hogy egy stúdió hány filmet gyártott.

A *Gyártó* társítás *Stúdió* végénél 1..1 címkét látunk. Ez a címke azt mondja, hogy a filmet egy és csakis egy stúdió készíti. Az nem lehetséges, hogy a filmet egyetlen stúdió sem gyártotta (a 4.36. ábra szerint ez lehetséges volt). Az 1..1 címke pontosan azt jelenti, mint az E/K-diagramon a kerekített nyílhegyvégződés.

Az ábrán egy *Irányít* társítást is látunk a stúdiók és elnökök között. Ennek a *Stúdiók* végénél 1..1 címkét találunk. Ez azt jelenti, hogy az elnök egy és csakis egy stúdió elnöke lehet. Ez ugyanazt a megszorítást jelenti, mint a 4.18. ábrán az *Elnökök*től a *Stúdiók*hoz vezető, kerekített nyílhegyben végződő vonal. Az *Irányít* társítás másik végénél 0..1 címke van. Ez azt fejezi ki, hogy a stúdiónak legfeljebb egy elnöke lehet, de nem kell, hogy minden pillanatban legyen elnöke. Ez a megszorítás pontosan megegyezik a nyílhegyben végződő vonal által kifejezett megszorítással. □

4.7.4. Társítások önmagával

A társítás mindkét vége lehet ugyanaz az osztály, az ilyen társításokat *önmagával való társítás*nak nevezzük. Az önmagával társított egyetlen osztály a társításban kétféle szerepet játszik, ezt kifejezendő a társításnak két nevet adunk, mindkét végénél egyet.



4.38. ábra. A filmsorozatokat kifejező önmagával való társítás

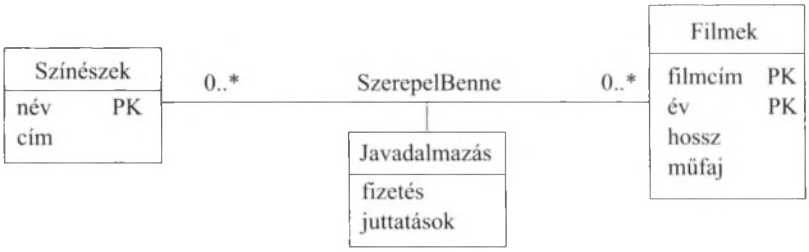
4.38. példa. A 4.38. ábrán a filmek folytatása kapcsolatát mutatja be. Egy társítást látunk, mindkét végénél a *Filmek* osztállyal. A társításnak az *Eredeti* szerepben lévő vége a film eredetijére mutat, a címkéje 0..1. Tehát a folytatásos filmeknek pontosan egy eredeti (első) filmjük van, ugyanakkor vannak nem folytatásos filmek is. A másik szerep, a *Folytatás* 0..* címkéjű. Ennek az az oka, hogy a folytatásos filmek eredetijének tetszőleges számú folytatása lehet. Megjegyezzük, hogy ezzel azt a nézőpontot fejezzük ki, hogy a folytatások mindegyikének ugyanaz az eredetije, nem pedig a folytatási sorban őt megelőző valamelyik. Például a *Rocky II.*–*Rocky V.* filmek mind a *Rocky* folytatásai. A *Rocky IV.*-et nem tekintjük a *Rocky III.* folytatásának. □

4.7.5. Társításokból képzett osztályok

A 4.1.9. alfejezetben[6] az E/K-modellben bemutatotthoz hasonló módon a társításokhoz is rendelhetünk attribútumokat. Az UML-ben társításosztálynak nevezett új osztályt hozunk létre, és ezt a társítás közepéhez kapcsoljuk. A társításosztálynak saját neve lesz, de attribútumai annak a társításnak az attribútumaiként tekintendők, melyhez a társításosztály kapcsolódik.

4.39. példa. Tegyük fel, hogy a *Filmek* és *Színészek* közötti *SzerepelBenne* kapcsolathoz a színész adott filmért való javadalmazására vonatkozó adatait szeretnénk hozzárendelni. Ez az információ nem rendelhető a filmhez (hiszen különböző színészek különböző gázsit kapnak), és nem rendelhető a színészhez sem (mert a színész különböző filmekért más-más gázsit kap). Így ezt az információt magához a társításhoz kell rendelni. Tehát minden színész-film pároshoz saját javadalmazás információ tartozik.

[6] A 4.7. ábrán bemutatott példa nem hozható át közvetlenül, mert az háromágú kapcsolatra vonatkozik.



4.39. ábra. A *SzerepelBenne* társítás *Javadalmazás* társításosztálya

A 4.39. ábrán a *SzerepelBenne* társítás és a *Javadalmazás* társításosztály látható. Ennek az osztálynak két attribútuma a *fizetés* és a *juttatások*. Megjegyezzük, hogy a *Javadalmazás* osztálynak nincs elsődleges kulcsa. Amikor a 4.39. ábrán látható diagramot relációvá alakítjuk, akkor a *Javadalmazás* attribútumait a színész-film párosokhoz előállított sorokhoz adjuk hozzá, ahogy ezt a 4.5.2. alfejezetben leírtuk. □

4.7.6. Osztályhierarchia az UML-ben

Minden UML-osztályhoz alosztályokból felépülő osztályhierarchia tartozhat. Ugyanúgy, ahogy az E/K-osztályhierarchiában, az elsődleges kulcs a hierarchia gyökércsúcsán lévő osztály kulcsa. Az UML-ben a C osztályhoz négy különböző típusú osztályhierarchia tartozhat annak függvényében, hogy az alábbi két kérdésre hogyan válaszolunk:

1. *Teljes vagy részleges.* A C osztály minden objektuma valamely alosztály eleme? Ha igen, akkor az alosztályok rendszere *teljes*, különben *részleges* vagy *nem teljes*.

2. *Diszjunkt vagy átlapolt.* Az alosztályok *diszjunkt*ak (egy objektum nem tartozhat több alosztályhoz)? Ha egy objektum kettő vagy több alosztályhoz is tartozhat, akkor az alosztályokat *átlapolt*nak mondjuk.

Megjegyezzük, hogy ezeket a döntéseket a hierarchia minden pontján, egymástól függetlenül meghozhatjuk.

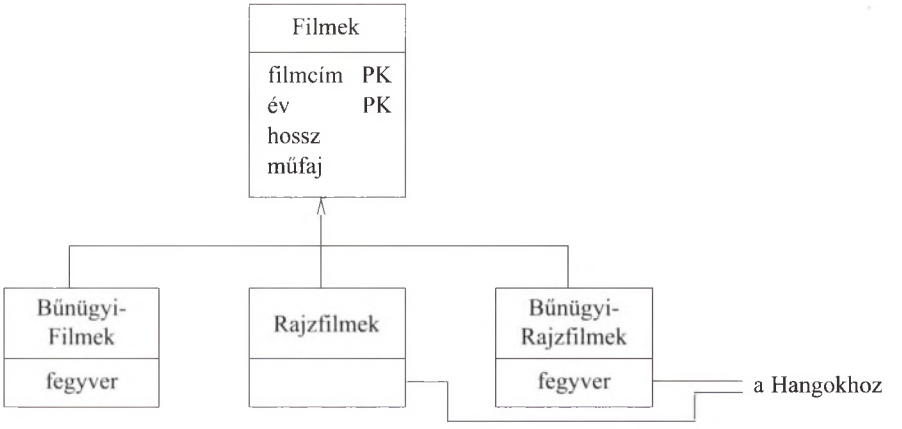
Több érdekes kapcsolat van az UML fent megadott osztályhierarchia fogalma, az objektum szemléletű rendszerekben megszokott osztályhierarchia fogalom és az E/K-modell osztályhierarchia fogalma között.

- Tipikus objektumszemléletű rendszerben az alosztályok diszjunktak. Azaz ugyanaz az objektum nem tartozhat két osztályhoz. A szülőosztályokból természetesen örökölnek tulajdonságokat, ebben az értelemben az objektumok a szülő osztályaikhoz is „tartozóak". Mindazonáltal az objektum nem tartozhat testvérosztályokhoz sem.

- Az E/K-modell automatikusan megengedi az alosztályok átlapolását.

- Mind az E/K-modell, mind az objektumszemléletű rendszerek megengedik alosztályok teljes és részleges rendszereit is. Vagyis nincs arra vonatkozó előírásuk, hogy egy szuperosztály elemeinek az alosztályaikban elő kell-e fordulniuk.

Az alosztályokat az osztályokhoz hasonlóan téglalapokkal reprezentáljuk. Úgy tekintjük, hogy az alosztály örökli a szuperosztály tulajdonságait (attribútumokat és társításokat). Az alosztályhoz tartozó összes további attribútumok az alosztály dobozában láthatók, az alosztálynak lehetnek továbbá saját társítási kapcsolatai más osztályokkal. Az osztály/alosztály viszonyt az UML-diagramban szabad csúcsával a szuperosztályra mutató háromszöggel jelöljük. Az alosztályok vízszintes vonallal kapcsolódnak a (háromszögből keletkező) nyílhoz.



4.40. ábra. A filmek osztálya rajzfilmek és bűnügyi filmek diszjunkt alosztályai

4.40. példa. A 4.40. ábrán a 4.1.11. alfejezet alosztályokat bemutató példájának UML-változata látható. Ugyanakkor különbözik is az E/K-alosztályoktól, mert ott nem volt előírás az alosztályok diszjunktsága, itt diszjunkt alosztályokat kell képeznünk. Ez az osztályhierarchia természetesen részleges, mivel sok olyan film van, mely nem rajzfilm, és nem is bűnügyi film.

Mivel az alosztályok diszjunktak, így harmadik alosztályt is képeznünk kell a *Roger nyúl a pácban* és hasonló filmek számára, amelyek rajzfilmek és bűnügyi filmek is egyszerre. *Megjegyzés:* mind a *BűnügyiFilmek*, mind a *BűnügyiRajzfilmek* osztályoknak van további attribútuma, a *fegyver*, a *Rajzfilmek* és *Bűnügyi-Rajzfilmek* osztályoknak pedig társítási kapcsolatuk van az ábrán nem látható *Hangok* osztállyal. □

4.7.7. Aggregáció és kompozíció

A sok-egy kapcsolatok két speciális jelölése inkább érdekes, mint jelentős. Egy felfogásban ezek az objektumszemléletű programozásra vonatkoznak, ahol nagyon általános, hogy egyik osztály attribútumai között a másik osztályokra hivatkozók is vannak. Más felfogásban ezek a speciális jelölések valódi különbségeket tükröznek arra nézve, hogyan kell a diagramot relációvá átalakítani; a dolog ezen vonatkozását a 4.8.3. alfejezetben tárgyaljuk.

Az *aggregáció* két osztály közötti kapcsolatot jelképező vonal, amely egyik végén üres rombuszban végződik. A rombusz azt jelenti, hogy a kapcsolat ezen végét 0..1 címkével kellene ellátni, tehát az aggregáció egy sok-egy társítás a kapcsolat rombusszal ellentétes végén lévő osztály felől a rombuszos végen lévő osztállyal. Jóllehet az aggregáció egy társítás, mégsem kell nevet adnunk neki, ugyanis a gyakorlatban ezt a nevet semmire nem használjuk a relációs megvalósításban.

A *kompozíció* hasonlóan egy társítási kapcsolat, de a rombuszos végén az 1..1 címkét feltételezi. Tehát a társítás túlsó végén lévő objektumok mindegyike a rombuszos végen lévő oldalon pontosan egy objektumhoz kell hogy kapcsolódjon. A kompozíciót feketével kitöltött rombusszal jelöljük.

4.41. példa. A 4.41. ábrán példát látunk mind aggregációra, mind kompozícióra. Ez egyszerre módosítása és részletezése a 4.37. ábrán bemutatott helyzetnek. Látunk egy társítást a *Filmek*től a *Stúdiók*hoz. Ennek a *Filmek*nél lévő végénél az 1..* azt mondja, hogy a stúdió legalább egy filmet gyártott. A társítás másik végéhez nem kell címkét adnunk, mert az üres rombusz jelzi, hogy ott a címke 0..1 lenne. Tehát egy filmhez vagy tartozik stúdió, vagy nem, de egynél több stúdió nem tartozhat hozzá. Ennek megvalósítása az is, hogy a filmek tartalmaznak a gyártó stúdió objektumra vonatkozó hivatkozást, ami lehet null, ha a filmet nem stúdió készítette.

Az ábra jobb oldalán a *GyártásIrányítók* osztályt és annak *Elnökök* alosztályát látjuk. Az *Elnökök*től a *Stúdiók*hoz vezető kapcsolat egy kompozíció. Jelzi, hogy egy elnök pontosan egy stúdió elnöke. A *Stúdiók* végződésnél a feketén kitöltött rombusz mutatja, hogy ott 1..1 címke áll(na). A kompozíció megvalósítása úgy történik, hogy *Elnökök* objektumban legyen hivatkozás a *Stúdiók* objektumra, és ez a hivatkozás nem lehet null. □

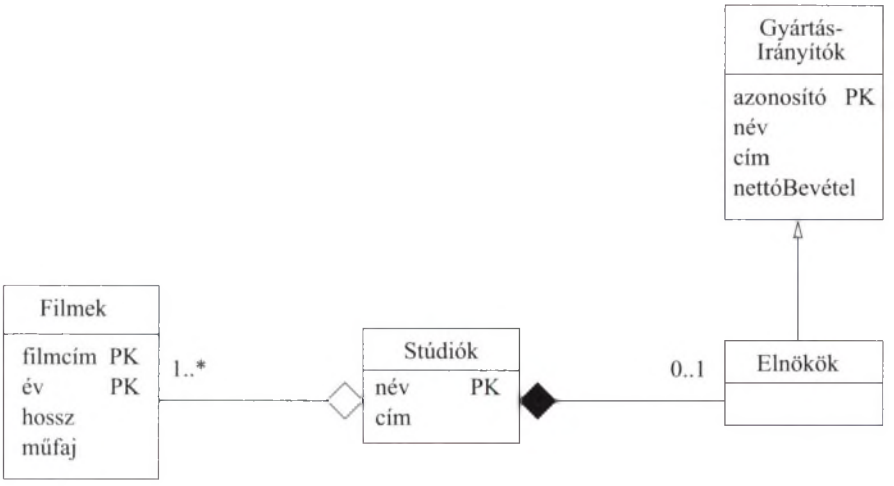
4.7.8. Feladatok

4.7.1. feladat. Rajzoljuk meg a 4.1.1. feladatban megfogalmazott probléma UML-diagramját.

4.7.2. feladat. Módosítsuk a 4.7.1. feladatban elkészített diagramot a 4.1.2. feladat kívánalmainak megfelelően.

4.7.3. feladat. Ismételjük meg a 4.1.3. feladatot UML-t használva.

4.7.4. feladat. Ismételjük meg a 4.1.6. feladatot UML-t használva.



4.41. ábra. Aggregáció a *Filmek*től a *Stúdiók*hoz és kompozíció az *Elnökök*től a *Stúdiók*hoz

4.7.5. feladat. Ismételjük meg a 4.1.7. feladatot UML-t használva. Az alosztályai diszjunktak vagy átfedők; teljesek vagy részlegesek?

4.7.6. feladat. Ismételjük meg a 4.1.9. feladatot UML-t használva.

4.7.7. feladat. Alakítsuk át a 4.30. ábrán látható E/K-diagramot UML-diagrammá.

! **4.7.8. feladat.** Hogyan tudnánk UML-ben megadni a filmek, színészek és stúdiók közötti háromágú kapcsolatot (lásd 4.4. ábra)?

! **4.7.9. feladat.** Ismételjük meg a 4.2.5. feladatot UML-t használva.

4.7.10. feladat. Amikor a társításokat *m..n* formájú címkékkel látjuk el, akkor általában azt tapasztaljuk, hogy m és n a 0, 1 vagy $*$ egyike. Adjunk példát olyan társításra, amelyben m és n más értékkel is megjelenik.

4.8. UML-diagram átírása relációs modellé

Az E/K-diagramok relációkká való átalakításakor tett meggondolások nagy része az UML-diagramok átalakítására is alkalmazható. Ezért röviden áttekintjük a fontosabb technikákat, és csak az olyan pontokon állunk meg, ahol a két modellezési módszer különbözik.

4.8.1. UML-diagram átírása relációs modellé – alapok

Áttekintést adunk a 4.5. alfejezetben tett meggondolások közül az itt is hasznosíthatókról:

- *Osztályok átalakítása relációkká.* Minden osztályhoz létrehozunk egy vele azonos nevű relációt, a reláció attribútumai pedig az osztály attribútumai lesznek.
- *Társítások átalakítása relációkká.* Minden társításhoz létrehozunk egy vele azonos nevű relációt. A társítással összekapcsolt két osztály kulcsattribútumai a létrehozott reláció attribútumai lesznek. Ha az attribútumok neve egymással ütközne, akkor megfelelően nevezzük át őket. Ha a társításhoz társítási osztály is tartozik, akkor annak attribútumait is illesszük be társításhoz létrehozott relációba.

4.42. példa. Tekintsük a 4.36. ábra UML-diagramját. Az ábrán látható három osztályhoz hozzuk létre a következő relációkat:

```
Filmek(filmcím, év, hossz, műfaj)
Színészek(név, cím)
Stúdiók(név, cím)
```

A két társításhoz az alábbi relációkat hozzuk létre:

```
SzerepelBenne(filmCím, filmÉv, színészNév)
Gyártó(filmCím, filmÉv, stúdióNév)
```

Megjegyzés: az attribútumok nevét nagy szabadsággal választhatjuk meg, most ezzel a lehetőséggel azért nem kívánunk élni, hogy az átalakítás jól átlátható legyen.

Másik példaként tekintsük a 4.39. ábrán lévő UML-diagramot, ezen társítási osztály látható. A *Filmek* és *Színészek* osztályokból konstruált táblák a fentiekkel megegyeznek, a társításhoz pedig a következő relációt rendeljük:

```
SzerepelBenne(filmCím, filmÉv, színészNév, fizetés, juttatások)
```

Vagyis vettük a társított osztályok kulcsattribútumait és a *Javadalmazás* társítási osztály két attribútumát. Megjegyzés: a *Javadalmazás* osztályhoz önmagában nem hozunk létre relációt. □

4.8.2. Az UML-osztályhierarchia átírása relációs modellé

A 4.6. alfejezetben megadott három lehetőség az UML-osztályhierarchiákra is alkalmazható. A lehetőségeket felidézve ezek az „E/K-elvű" (minden alosztályhoz létrehozunk relációt, amelybe csak a kulcsattribútumok és az alosztály saját attribútumai kerülnek), az „objektumelvű" (minden egyed, összes attribútumával, egyetlen, az alosztályhoz konstruált relációban fordul elő) és a „nullértékes"

(egyetlen reláció az osztályhierarchia összes osztályai számára). Az UML-ben, ha rendelkezünk azokkal az információkkal, hogy az alosztályok diszjunktak vagy átlapoltak, illetve hogy az osztályhierarchia teljes vagy részleges, akkor ennek birtokában egyik vagy másik módszert találhatjuk alkalmasabbnak. Néhány meggondolás:

1. Ha az osztályhierarchia minden szinten diszjunkt, akkor az „objektumelvű" reprezentálás javallott. A relációk képzésekor nem kell az összes lehetséges alosztályrészfát figyelembe vennünk, mivel tudjuk, hogy minden objektum csak egyetlen osztályhoz és a hierarchia szerinti őseihez tartozik. Így nem szükséges exponenciálisan növekvő számú relációt létrehoznunk.

2. Ha az osztályhierarchia teljes és minden szinten diszjunkt, akkor a feladat egészen egyszerű. Ha az „objektumelvű" megközelítést használjuk, akkor csak a hierarchia leveleinél található osztályokhoz kell relációkat konstruálni.

3. Ha az osztályhierarchia nagy és néhány, vagy az összes szinteken átlapolt, akkor az az „E/K-elvű" megközelítést indukálja. Annyi reláció létrehozását igényli, hogy az adatbázisséma ormótlan lesz.

4.8.3. Aggregáció és kompozíció átírása relációs modellé

Az aggregációk és a kompozíciók valójában a sok-egy kapcsolatok típusai. Így ezeknek a relációs adatbázissémában való reprezentálásának egyik megközelítése az olyan átalakítás, amilyeneket a 4.8.1. alfejezetben, az összes kapcsolatokra vonatkozóan megadtunk. Mivel az UML-diagramban ezeket az elemeket nem kellett elnevezni, a hozzájuk konstruált relációknak nevet is kell konstruálnunk.

Ugyanakkor rejtett feltevésünk az, hogy az aggregációk és kompozíciók ezen megvalósítása nem kívánatos. Emlékeztetünk a 4.5.3. alfejezetre, ha van egy E egyedhalmazunk és R egy sok-egy kapcsolat E-ről egy másik F egyedhalmazhoz, akkor lehetőségünk – egyesek szerint kötelességünk – van arra, hogy az E és R relációkat egyesítsük. Ezzel egy relációt hozunk létre E-ből és R-ből, E összes attribútumával és F kulcsattribútumaival.

A javaslatunk az, hogy az aggregációk és kompozíciók esetében ezt a gyakorlatot kövessük. Ne hozzunk létre önálló relációkat az aggregációk és kompozíciók számára. Inkább járjunk el úgy, hogy a kapcsolat nem rombusz végéhez létrehozott relációt bővítsük ki a rombuszvéghez tartozó osztály kulcsattribútumaival. Aggregáció esetében megengedett, kompozíció esetében nem megengedett, hogy ezen attribútumok értéke null legyen.

4.43. példa. Tekintsük a 4.41. ábra UML-diagramját. Mivel itt osztályhierarchia van, el kell döntenünk, hogy a *GyártásIrányítók*at és az *Elnökök*et hogyan alakítjuk át a relációs modellbe. Használjuk az „E/K-elvű" megközelítést, így az `Elnökök` relációnak csak a *GyártásIrányítók*ból származó `azonosító` attribútuma lesz.

A *Filmek*től a *Stúdiók*hoz vezető aggregációt a *Stúdiók név* kulcsattribútumának a `Filmek` táblába való beillesztésével tudjuk reprezentálni. Az *Elnök*öktől a *Stúdiók*hoz vezető kompozíciót pedig a *Stúdiók* kulcsának az `Elnökök` relációba való beillesztésével reprezentáljuk. Nem hoztunk létre önálló relációt sem az aggregáció, sem a kompozíció reprezentálásához. Ebből az UML-diagramból összesen az alábbi relációkat konstruáltuk meg:

```
GyártásIrányítók(azonosító, név, cím, nettóBevétel)
Elnökök(azonosító, stúdióNév)
Filmek(filmcím, év, hossz, műfaj, stúdióNév)
Stúdiók(név, cím)
```

Ahogy korábban is, az attribútumok elnevezésénél némi szabadsággal éltünk azért, hogy az értelmezés világos legyen. □

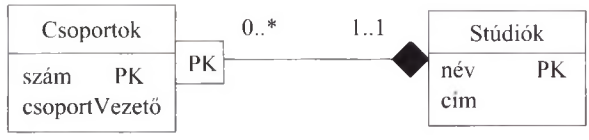
4.8.4. A gyenge egyedhalmazok UML-megfelelője

Nem foglalkoztunk még az E/K-modellben dupla keretezéssel jelölt gyenge egyedhalmazok UML-jelölésével. Most megmutatjuk, hogy nem is kell. Ennek az az oka, hogy az E/K-modelltől eltérően az UML-ben az objektum szemléletű rendszerek hagyományai szerint minden objektumnak van *objektumazonosító*ja. Tehát akkor is meg tudunk különböztetni két objektumot, ha attribútumértékeik és más tulajdonságaik is egymással mind megegyezők. Az objektumazonosító tipikusan olyan, mint az objektumra mutató hivatkozás vagy pointer.

Az UML-ben tekinthetjük úgy, hogy az osztályhoz tartozó objektumoknak is hasonló objektumazonosítójuk van. Így, ha az osztályhoz tartozó attribútumok nem szolgáltatnak egyedi azonosítót az osztály objektumai számára, akkor létrehozhatunk egy új attribútumot az osztályhoz konstruált relációban, mely a reláció kulcsa lesz és az objektum objektumazonosítóját fogja reprezentálni.

Mindazonáltal, ahogy a gyenge egyedhalmazokhoz E/K-modellben kiteljesítő kapcsolatot használtunk, ennek megfelelőjeként az UML-ben a kompozíciót használhatjuk. Ez a kompozíció a „gyenge" osztálytól (azon osztálytól, amely attribútumai nem képeznek kulcsot) a kiteljesítő osztályhoz vezető kapcsolat. Ha több „kiteljesítő" osztály van, akkor több kompozíciót használhatunk. A *kiteljesítő* kompozíciókhoz speciális jelölést használunk: a *gyenge* osztályhoz egy kis dobozt ragasztunk, benne PK felirattal, és ez szolgál a kiteljesítő kompozíció végződéseként. A PK-val azt jelezzük, hogy a gyenge osztály kulcsát a kapcsolat másik végén lévő *kiteljesítő* osztálykulcs attribútumai és a gyenge osztályban PK-val megjelölt attribútumok együttesen képezik. Mint ahogy a gyenge egyedhalmazok esetében, úgy itt is több kiteljesítő kompozíció és kiteljesítő osztály lehetséges, és ezek a kiteljesítő osztályok maguk is lehetnek gyengék, ebben az esetben a most leírt szabály rekurzívan alkalmazandó.

4.44. példa. A 4.42. ábra a 4.20. példában szereplő *Csoportok* gyenge egyedhalmazzal analóg. A *Csoportok*tól „PK" dobozban végződő kompozíciós kapcsolat vezet a *Stúdiók*hoz, jelezvén, hogy ez a kompozíció szolgáltatja a *Csoportok* kulcsának egy részét. □



4.42. ábra. A *Csoportok* gyenge egyedhalmaz-támogatása kompozícióval és a *Stúdiók* osztállyal

Az olyan gyenge struktúrákat, amilyeneket a 4.42. ábra is mutat, ugyanúgy alakítjuk át relációvá, ahogy azt a 4.5.4. alfejezetben tettük. Értelemszerűen van egy reláció a *Stúdiók* számára. Természetesen nem hozunk létre relációt a kompozíció részére. A *Csoportok*hoz létrehozott reláció pedig nemcsak a saját attribútumait tartalmazza, hanem a kompozíció másik végén lévő *Stúdiók* osztály kulcsát is.

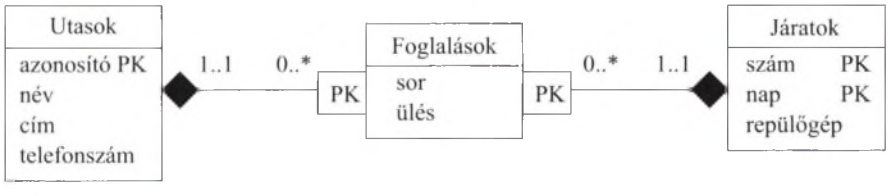
4.45. példa. A 4.44. példához tartozó relációk:

```
Stúdiók(név, cím)
Csoportok(szám, csoportVezető, stúdióNév)
```

Amint korábban is, a *Stúdiók név* attribútumát a `Csoportok` relációban – a jól érthetőség kedvéért – átneveztük. □

4.8.5. Feladatok

4.8.1. feladat. Alakítsa át a 4.43. ábra UML-diagramját relációkká.



4.43. ábra. A 4.29. ábra E/K-diagramjának megfelelő UML-diagram

4.8.2. feladat. Alakítsa át a következő UML-diagramokat relációkká:

a) 4.37. ábra;

b) 4.40. ábra;

c) a 4.7.1. feladat megoldásaként létrehozott diagram;

d) a 4.7.3. feladat megoldásaként létrehozott diagram;

e) a 4.7.4. feladat megoldásaként létrehozott diagram;

f) a 4.7.6. feladat megoldásaként létrehozott diagram.

! **4.8.3. feladat.** Hány relációt fogunk létrehozni az „objektumelvű" megközelítést használva egy háromszintű osztályhierarchiához, ha az első és második szinteken minden osztálynak három-három alosztálya van, és a hierarchiáról tudjuk, hogy:

a) minden szinten diszjunkt és teljes;

b) diszjunkt, de nem minden szinten teljes;

c) nem diszjunkt és nem is teljes.

4.9. Bevezetés az ODL-be

Az ODL (*Object Definition Language – Objektum Definíciós Nyelv*) egy szöveg alapú nyelv, melynek segítségével adatbázisok struktúráját specifikálhatjuk objektumorientált terminológiával. Az UML-hez hasonlóan az ODL központi fogalma szintén az osztály. Az ODL-osztályoknak – ugyanúgy, mint az UML-osztályoknak is – van nevük, attribútumaik és metódusaik. A kapcsolatok – amelyek analógak az UML társításaival – az ODL-ben nem önálló fogalmak, hanem tulajdonságok új családjaként az osztályokba beágyazottak.

4.9.1. Osztálydeklarációk

Az ODL-ben az osztálydeklaráció egyszerű formája:

```
class <név> {
    <jellemzők listája>
};
```

Tehát a `class` kulcsszót az osztály neve követi, majd zárójelben a jellemzők listája. Jellemző lehet attribútum, kapcsolat és metódus.

4.9.2. Attribútumok az ODL-ben

A jellemzők legegyszerűbb fajtája az *attribútum*. Az ODL nem követeli meg, hogy az attribútum típusa csak elemi (például egész vagy karaktersorozat) lehet. Az ODL-nek a 4.9.6. alfejezetben leírt típusrendszere van, mely lehetővé teszi, hogy strukturált típusokat és kollekciótípusokat (például halmazok) hozzunk létre. Például a `cím` lehet strukturált típus, az utca, város és irányítószám mezőkkel; a `telefonszám`, amely karaktersorozatok halmaza lehet; és még sokkal összetettebb típusok is előfordulhatnak. Az attribútumot az osztálydeklarációban az `attribute` kulcsszóval, a típusával és a neve megadásával határozzuk meg.

```
1) class Film {
2)   attribute string filmcím;
3)   attribute integer év;
4)   attribute integer hossz;
5)   attribute enum Műfajok
           {dráma, vígjáték, sci-fi, ifjúsági} műfaj;
   };
```

4.44. ábra. A `Film` osztály deklarációja ODL-ben

Miért adunk nevet a felsorolásnak és a struktúrának?

A 4.44. ábrán deklarált felsorolás neve `Műfajok`, úgy tűnik, mintha nem volna rá (az elnevezésre) szükség. Mindazonáltal nevet adva neki másutt is, többek között más osztályok deklarációjában is hivatkozhatunk rá. A hivatkozáskor a `Film::Műfajok` *bővített nevet* használjuk, mivel a felsorolástípust eredetileg a `Film` osztályban deklaráltuk.

4.46. példa. A 4.44. ábrán látható a filmek osztályának deklarációja. Ez a deklaráció nem teljes, de később meg fogjuk adni a teljes deklarációt is. Az első sor deklarálja a `Film` osztályt. A következő négy sor deklarálja a `Film` objektum négy attribútumát.

A 2., 3. és 4. sorok deklarálják a `filmcím`, `év` és `hossz` attribútumokat. Ezek közül az első karakterlánc típusú, a másik kettő egész szám típusú. Az 5. sorban a `műfaj` attribútumot felsorolás típusúnak deklarálja. A felsorolás (szimbolikus konstansok listája) neve `Műfajok`, a `műfaj` attribútum négy lehetséges értéke pedig `dráma`, `vígjáték`, `sci-fi` és `ifjúsági`. A felsorolásnak nevet kell adni, ez a név használható arra, hogy bárhol erre a típusra hivatkozhassunk. □

4.47. példa. A 4.46. példában az összes típus egyszerű típus volt. Most adunk egy példát, amelyben szerepel összetett típus is. A `Színész` osztályt a következő módon definiálhatjuk:

```
1) class Színész {
2)   attribute string név;
3)   attribute Struct Cím
           {string város, string utca} lakcím;
   };
```

A 2. sor specifikálja a színész nevét, a 3. sor pedig a lakcímét. Ez utóbbi attribútum típusa egy *rekordstruktúra*. Ennek a típusnak a neve `Cím` és két mezője van: a `város` és az `utca`. Mindkét mező típusa karakterlánc. Általában az

ODL-ben a `Struct` kulcsszóval definiálhatunk rekordstruktúrát, és kapcsos zárójelek között soroljuk fel a mezőit névvel és típussal. A felsoroláshoz hasonlóan a struktúratípusnak is nevet kell adni, amellyel hivatkozhatunk rá. □

4.9.3. Kapcsolatok az ODL-ben

Az ODL-ben a kapcsolatot az osztály deklarációjában adjuk meg, a `relationship` kulcsszóval, a kapcsolat típusának és nevének megadásával. A kapcsolat típusa azt írja le, hogy az osztály egyedi objektumai részt vesznek a kapcsolatban. A típus jellemzően egy másik osztály (ha a kapcsolat sok-egy) vagy kollekciótípus (ha a kapcsolat egy-sok vagy sok-sok). A komplex típusokat egy példával mutatjuk be, a teljes típusrendszert majd a 4.9.6. alfejezetben tárgyaljuk.

4.48. példa. Tegyük fel, hogy szeretnénk a 4.46. példában lévő `Film` osztályba felvenni egy jellemzőt, ami színészek egy halmaza. Pontosabban, minden `Film` objektumot össze akarunk kapcsolni a filmben szereplő `Színész` objektumok halmazával. A `Film` és a `Színész` osztályok ezen kapcsolata reprezentálásának legjobb módja egy *kapcsolat* deklarálása lesz a `Film` és `Színész` osztályok között. Ezt a kapcsolatot a következő módon adhatjuk meg a `Film` osztály deklarációjában:

```
relationship Set<Színész> szereplők;
```

Ez a kapcsolat azt jelenti, hogy a `Film` osztály minden objektumában van egy hivatkozás a `Színész` osztály objektumainak egy halmazára. Ennek a halmaznak a neve `szereplők`. □

4.9.4. Inverz kapcsolatok

Ahogy szeretnénk hozzákapcsolni a színészeket egy adott filmhez, ugyanígy szeretnénk tudni, hogy egy adott színész mely filmekben játszott. Ezt az információt a `Színész` osztályban a következő sor tartalmazza:

```
relationship Set<Film> szerepelBenne;
```

Ezt a sort kell hozzáadni a 4.47. példához. Azonban ez a sor és a hasonló deklaráció a `Film` osztályban nem veszi tekintetbe a színészek és filmek közötti kapcsolat egy nagyon fontos aspektusát. Elvárjuk ugyanis, hogy ha egy S színész benne van az F filmhez tartozó szereplők halmazában, akkor F benne legyen az S színészhez tartozó `szerepelBenne` halmazban. Feltüntethetjük ezt az összefüggést a szereplők és a `szerepelBenne` kapcsolatok között úgy, hogy mindkét deklarációban használjuk az `inverse` kulcsszót és a másik kapcsolat nevét. Ha a másik kapcsolat egy másik osztályban van (rendszerint ez így van), akkor az osztály nevével együtt kell hivatkozni a kapcsolatra úgy, hogy az osztály nevét dupla kettőspont (`::`) követi, és utána jön a kapcsolat neve.

4.49. példa. Azért, hogy a `Színész` osztály `szerepelBenne` kapcsolata inverze legyen a `Film` osztály szereplők kapcsolatának, a 4.45. ábrának megfelelően módosítanunk kell ezen osztályok deklarációját. (Az ábrán a `Stúdió` osztály deklarációja is látható, ezt később fogjuk használni.) A 6. sorban a `Film` osztály szereplők kapcsolatát definiáljuk, és megadjuk, hogy ezen kapcsolat inverze a `Színész::szerepelBenne` kapcsolat. Mivel a `szerepelBenne` kapcsolat egy másik osztályban definiált kapcsolat, ezért a hivatkozott kapcsolat neve előtt két kettősponttal elválasztva meg kell adnunk az osztály nevét is (`Színész`).

Hasonló a 11. sorban a `szerepelBenne` kapcsolat deklarációja. Inverze a `Film` osztály szereplők kapcsolata, ennek megadása kötelező, mert az inverz kapcsolatok mindig összetartozó párok. □

```
1)  class Film {
2)    attribute string filmcím;
3)    attribute integer év;
4)    attribute integer hossz;
5)    attribute enum Műfajok
            {dráma, vígjáték, sci-fi, ifjúsági} műfaj;
6)    relationship Set<Színész> szereplők
            inverse Színész::szerepelBenne
7)    relationship Stúdió gyártó
            inverse Stúdió::gyárt;
    };

8)  class Színész {
9)    attribute string név;
10)   attribute Struct Cím
            {string város, string utca} lakcím;
11)   relationship Set<Film> SzerepelBenne
            inverse Film::szereplők;
    };

12) class Stúdió {
13)   attribute string név;
14)   attribute string cím;
15)   relationship Set<Film> gyárt
            inverse Film::gyártó;
    };
```

4.45. ábra. Néhány ODL-osztály és kapcsolataik

Általános szabályként kimondhatjuk, hogy egy C osztály R kapcsolata esetén, ha a C osztály egy x objektumával a D osztály $y_1, y_2, \ldots, y_n$ objektumai kapcsolódnak, akkor minden y_i objektum inverz R kapcsolatban van az x-szel (esetleg más objektumokkal is).

4.9.5. Kapcsolattípusok

Az E/K-modell bináris kapcsolattípusához hasonlóan az ODL inverz kapcsolatpárosai is osztályozhatók, lehetnek sok-sok, sok-egy, egy-sok és egy-egy kapcsolatok. A kapcsolat deklarációja mutatja meg, hogy a kapcsolat a fentiek közül hová tartozik.

1. Ha a C és D osztályok között sok-sok kapcsolat van, akkor a C osztályban a kapcsolat típusa `Set<`D`>` és a D osztályban a kapcsolat típusa `Set<`C`>`[7].

2. Ha a C és D osztályok között sok-egy kapcsolat van, akkor a C osztályban a kapcsolat típusa D, ugyanakkor a D osztályban a kapcsolat típusa `Set<`C`>`.

3. Ha a C és D osztályok között egy-sok kapcsolat van, akkor a C és D fenti (2. pontbeli) szerepe megcserélődik.

4. Ha a kapcsolat egy-egy kapcsolat, akkor a C osztályban a kapcsolat típusa D, és a D osztályban a kapcsolat típusa C.

Megjegyezzük, hogy amint az E/K-modellben, úgy itt is megengedjük, hogy a sok-egy és az egy-egy kapcsolatokba beleértjük az olyan eseteket is, amikor az „egy" egyes objektumokra „semmi". Például a C és D osztályok közötti sok-egy kapcsolatban egyes C osztálybeli objektumok esetében nincs a D osztályban (hiányzik vagy „nullértékű") kapcsolódó objektum. Természetesen, ha a D objektumaihoz C-beli objektumok halmaza társul, ekkor is előfordulhat, hogy egyes D objektumok esetében ez a halmaz üres.

4.50. példa. A 4.45. ábrán megadtuk a `Film`, a `Színész` és a `Stúdió` osztályokat. Az első két osztályt már bevezettük a 4.46. és 4.47. példákban. Már foglalkoztunk a `szereplők` és a `szerepelBenne` párok közötti kapcsolattal is. Mivel a kapcsolat típusa mindkét oldalon `Set`, így látjuk, hogy a kapcsolatpár a `Színész` és `Film` osztályok közötti sok-sok típusú kapcsolatot reprezentál.

A stúdióobjektumoknak `név` és `cím` attribútumaik vannak, amelyek a 13. és a 14. sorban találhatók. A stúdiókhoz ugyanolyan típusú címet használunk, mint amilyet a `Színész` osztályban a színészek címeként megadtunk.

A 7. sorban látható a `gyártó` kapcsolat a filmek és a stúdiók között. Ennek az inverz kapcsolata a 15. sorban látható `gyárt` kapcsolat. Mivel a gyártó típusa `Stúdió`, és a `gyárt` típusa `Set<Film>`, ebből látjuk, hogy ezen inverz kapcsolatpáros sok-egy típusú a `Film`-től a `Stúdió` felé. □

[7] A `Set` kicserélhető bármilyen más „kollekciótípusra", lehet `list` (lista), vagy `bag` (multihalmaz), a 4.9.6. alfejezetnek megfelelően. A kapcsolatok szempontjából a kollekciótípusok mind halmaznak tekinthetők.

4.9.6. Típusok az ODL-ben

Az ODL egy típusrendszert ajánl fel az adatbázis-tervezőnek, amely hasonló ahhoz, ami a C-ben vagy más hagyományos programozási nyelvben található. Egy típusrendszer alaptípusokból épül fel úgy, hogy adottak az alaptípusok, és bizonyos rekurzív szabályokkal összetett típusok építhetők az egyszerűbb típusokból. ODL-ben az alaptípusok a következők:

1. *Atomi típusok:* integer, float, character, string, boolean és enum. Ez utóbbi a *felsorolás* típus, szimbolikus nevek – mint például a 4.45. ábra 5. sorában a `dráma` – listája.

2. *Osztálynevek:* olyanok, mint a `Film` vagy a `Színész` osztályok, amelyek lényegében struktúrákat reprezentáló típusok. Ezeknek a struktúráknak a komponensei az osztály attribútumainak és kapcsolatainak felelnek meg.

A fenti alaptípusokból a következő *típuskonstruáló* utasításokkal összetett struktúrák képezhetők:

1. *Halmaz.* Ha T egy típus, akkor `Set<T>` jelöli azt a típust, amelynek az értékei T típusú elemek egy véges halmaza. Például ezt használjuk a 4.45. ábra 6., 11. és 15. sorában.

2. *Multihalmaz.* Ha T egy típus, akkor `Bag<T>` jelöli azt a típust, amelynek az értéke T típusú elemek multihalmaza.

3. *Lista.* Ha T egy típus, akkor `List<T>` jelöli azt a típust, amelynek az értéke T típusú elemek véges listája, a lista természetesen lehet üres is.

4. *Tömb.* Ha T egy típus és i egy egész szám, akkor `Array<T,i>` jelöli azt a típust, amelynek értéke egy i elemű tömb, ahol az elemek típusa T. Például `Array<char,10>` egy 10 karakter hosszúságú láncot jelent.

5. *Szótár.* Ha T és S típusok, akkor `Dictionary<T,S>` párok véges halmazából álló típust jelöl. Minden pár egy T *kulcs típusú értékből* és egy S *kiterjedéstípusú értékből* áll. A szótárban nem fordulhat elő két, azonos kulcsú pár.

6. *Struktúra.* Ha $T_1, T_2, \ldots, T_n$ típusok és $F_1, F_2, \ldots, F_n$ mezőnevek, akkor

   `Struct N {`T_1 F_1`,` T_2 F_2`,...,` T_n F_n`}`

   jelöli azt az N nevű típust, amely egy n mezőből álló struktúra. Az i mező neve F_i és típusa T_i. Például a 4.45. ábra 10. sorában használtuk a `Cím` struktúrát két mezővel. Mindkét mező típusa `string` és a mezők nevei rendre `város` és `utca`.

Halmazok, multihalmazok és listák

A halmazok, multihalmazok és listák közötti különbség megértéséhez gondoljunk vissza arra, hogy a halmaz rendezetlen elemekből áll és minden elem csak egyszer fordul elő. A multihalmaz megengedi, hogy egy elem többször is előforduljon benne, de az elemek továbbra is rendezetlenek. A listában viszont az elemek rendezettek és a lista is megengedi, hogy egy elem többször is előforduljon benne. Így $\{1, 2, 1\}$ és a $\{2, 1, 1\}$ ugyanaz a multihalmaz, de az $(1, 2, 1)$ és $(2, 1, 1)$ nem azonos listák.

Az első öt típust – halmaz, multihalmaz, lista, tömb és szótár – *kollekciótípusok*nak nevezzük. Különböző szabályok vannak arra, hogy mely típusok használhatók attribútumoknál és mely típusok használhatók kapcsolatok esetében.

- Egy kapcsolat típusa vagy egy osztálytípus, vagy egy kollekció típusú konstruktor alkalmazva egy osztálytípusra.
- Egy attribútum típusa atomi típusból vagy típusokból kiindulva épül fel[8]. Ezután tetszőlegesen sokszor kedvünk szerint alkalmazhatjuk akár a struktúra típusú, akár a kollekció típusú konstruktorokat.

4.51. példa. Néhány lehetséges attribútumtípus:

1. `integer`;
2. `Struct N {string mező1, integer mező2}`;
3. `List<real>`;
4. `Array<Struct N {string mező1, integer mező2}, 10>`.

Az első példa egy atomi típus, a második atomi típusok struktúrája, a harmadik atomi típus kollekciója, a negyedik pedig atomi típusok struktúrájából felépülő tömb.

Tegyük fel, hogy a `Film` és a `Színész` osztálynevek alaptípusok. Ekkor például `Film` vagy `Bag<Színész>` típusú kapcsolatokat konstruálhatunk. Azonban a következő típusok nem szerepelhetnek kapcsolat típusaként:

1. `Struct N {Film mező1, Színész mező2}`. Egy kapcsolat típusa nem lehet struktúra.
2. `Set<integer>`. Egy kapcsolat típusa nem alapulhat atomi típuson.

[8] Osztálytípusok is használhatók, ezek „egyirányú" kapcsolathoz hasonló attribútumokat állítanak elő. Az ilyen attribútumokkal itt nem foglalkozunk.

3. `Set<Array<Színész, 10 >>`. Egy kapcsolat típusa nem tartalmazhat két kollekciótípust.

□

4.9.7. Alosztályok az ODL-ben

Lehetőségünk van arra, hogy egy C osztályt egy másik, D osztály alosztályaként deklaráljunk. Ehhez C deklarációjában a nevét követően meg kell adnunk az `extends` kulcsszót és a D osztály nevét. Így a C osztály örökölni fogja a D osztály összes tulajdonságait, és lehetnek további saját tulajdonságai is.

4.52. példa. A 4.10. példában a rajzfilmeket a filmek alosztályaként tekintettük, és a filmekre jellemző tulajdonságokon felül még „hangok" tulajdonságként hozzájuk rendeltük a szinkronszínészeik halmazát. A `Film` osztály `Rajzfilm` alosztályának ODL-deklarációja:

```
class Rajzfilm extends Film {
   relationship Set<Színész> hangok;
};
```

A következő példában a `Film` osztály másik alosztályát, a `BűnügyiFilm` osztályt definiáljuk a `fegyver` kiegészítő attribútummal:

```
class BűnügyiFilm extends Film {
    attribute string fegyver;
};
```

□

Olykor – a *Roger nyúl a pácban*-hoz hasonló esetekben – olyan osztályra van szükségünk, amely egyszerre kettő vagy több másik osztály alosztálya. Az ODL-ben az `extends` kulcsszót több, egymástól kettősponttal elválasztott osztály felsorolása követheti[9]. Így a negyedik osztályt a következőképpen deklarálhatjuk:

```
class BűnügyiRajzFilm extends BűnügyiFilm : Rajzfilm;
```

Megjegyezzük, hogy többszörös öröklésnél van arra esély, hogy az osztály két azonos nevű tulajdonságot örököljön. E konfliktus feloldási módja implementációfüggő.

[9] Technikailag a második és az azt követő neveknek inkább „interface"-eknek kell lenniük osztályok helyett. Az ODL-ben az *interface* nagyjából úgy tekinthető, mint olyan osztálydefiníció, melyhez nem tartozik objektumhalmaz.

4.9.8. Kulcsok deklarálása az ODL-ben

Az ODL-ben az osztályhoz a kulcs vagy kulcsok deklarálása nem előírás. Ennek az az oka, hogy az ODL – objektumszemléletű lévén – úgy tekinti, hogy minden objektumnak van azonosítója, ahogy ezt a 4.8.4. alfejezetben az UML-lel kapcsolatban már tárgyaltuk.

ODL-ben egy vagy több attribútumot deklarálhatunk kulcsnak egy osztály számára. Egy kulcsot a `key` vagy a `keys` kulcsszó (mindegy, hogy melyik) után megadott attribútummal, illetve attribútumokkal deklarálhatunk. Ha több attribútum alkotja a kulcsot, akkor az attribútumok listáját zárójelbe kell tenni. A kulcs deklarációja közvetlenül az osztály nevének megadása után, a zárójelben következik.

4.53. példa. Deklaráljuk a `Film` osztályt kulcsával, amely a `filmcím` és `év` attribútumok halmaza. A deklaráció a következőképpen fog kezdődni:

```
class Film (key (filmcím, év)) {
```

Használhatnánk a `keys` kulcsszót is, még akkor is, ha csak egy kulcsot deklarálunk. □

Lehetséges, hogy több attribútumhalmaz is kulcs. Ha így van, akkor a `key(s)` kulcsszó után a kulcsokat vesszővel elválasztva kell megadni. Ha egy kulcs több attribútumból áll, akkor zárójelek között kell megadni az attribútumok listáját, így egyértelmű, hogy egy több attribútumot tartalmazó kulcsról van-e szó vagy pedig több, egyattribútumos kulcsról.

Az ODL-szabvány megengedi, hogy ne csak attribútumok, hanem más tulajdonságok is előforduljanak a kulcsokban. Nem alapvető probléma, ha metódust vagy kapcsolatot kulcsként vagy kulcs részeként deklarálunk, mivel a kulcsok csak tájékoztató utasítások az adatbázisrendszer számára, csak lehetőségek, nem pedig követelmények. Például, ha metódust kulcsként deklarálunk, akkor azt jelenti, hogy az osztály különböző objektumaira alkalmazva a metódust, garantált, hogy különböző visszatérési értékeket fog adni.

Ha sok-egy kapcsolatot használunk a kulcsdeklarációban, akkor az E/K-modell gyenge egyedhalmazaihoz hasonló hatást érünk el. Deklarálhatjuk, hogy az O_1 objektum, amelyre a kapcsolat „sok" oldalán az O_2 objektum hivatkozik, az O_2 objektum – esetleg további tulajdonságokkal együtt – a kulcsához tartozik, tehát különböző O_2 objektumokra egyedi. Ugyanakkor emlékezzünk arra, hogy nem követelmény az, hogy az osztálynak kulcsa legyen; nem vagyunk kötelezve arra, ahogy a gyenge egyedhalmazok esetében volt, hogy olyan osztályokhoz, melyek saját attribútumaiból nem képezhető kulcs, valamilyen speciális módszerrel keressünk kulcsot.

4.54. példa. Tekintsük a 4.20. ábra *Csoportok* gyenge egyedhalmazát. Emlékezzünk arra, hogy – mivel több stúdiónak is lehet azonosan számozott csoportja – feltételeztük, hogy a csoportokat a számuk és az a stúdió azonosítja, amelyhez a csoport tartozik. A `Csoport` osztályt a 4.46. ábrán látható módon deklarálhatjuk. Megjegyezzük, hogy módosítanunk kell a `Stúdió` deklarációját is, bele

kell illesztenünk a `csoportja` kapcsolatot, a `Csoport`-beli `része` kapcsolat inverzeként; ezt a módosítást most nem tesszük meg.

```
Class Csoport (key (szám,része)) {
    attribute integer szám;
    attribute string csoportVezető;
    relationship Stúdió része
        inverse Stúdió::csoportja;
};
```

4.46. ábra. A csoportok ODL-deklarációja

Ahogy ez a kulcsdeklaráció állítja, nem fordulhat elő, hogy ugyanazon stúdióhoz két azonos számú csoport tartozzon. Megjegyezzük, hogy ez a kikötés hasonlít a 4.20. ábrán látható E/K-diagram által mutatotthoz, mely szerint a csoport száma és a vele kapcsolatban levő stúdió neve (ami a stúdió kulcsa) egyértelműen meghatározza a csoportot, mint egyedet. □

4.9.9. Feladatok

4.9.1. feladat. A 4.1.1. feladatban egy bank adatbázisának informális leírása szerepelt. Adjuk meg ezt a tervet ODL-ben, benne a megfelelő kulcsokat is.

4.9.2. feladat. Módosítsuk a 4.9.1. feladat tervét a 4.1.2. feladatban leírtaknak megfelelően. Írjuk le a változásokat úgy, hogy nem készítünk teljesen új tervet.

4.9.3. feladat. Adjuk meg a 4.1.3. feladatban szereplő csapatok-játékosok-szurkolók adatbázis tervét ODL-ben, a megfelelő kulcsokkal együtt. Az eredeti feladatban nehézséget jelentett a csapatok színeinek kezelése, ez a probléma az ODL-ben miért nem jelenik meg?

! **4.9.4. feladat.** Tegyük fel, hogy családfát szeretnénk összeállítani. Csak egy osztályunk van, a `Személy`. Egy személyről a következő információkat kívánjuk tárolni: neve (az egyetlen attribútum), anyja, apja és gyerekei (ezek a kapcsolatok). Adjunk meg egy ODL-tervet a `Személy` osztályra. Ügyeljünk a kapcsolatok inverzére; az `anya`, `apa` és `gyerekek` kapcsolatok a `Személy` osztály önmagával való kapcsolatai. Az `anya` kapcsolat a `gyerekek` kapcsolat inverze? Miért vagy miért nem? Írjuk le a kapcsolatokat és inverzeiket párok halmazaként.

! **4.9.5. feladat.** A 4.9.4. feladat tervét egészítsük ki a `végzettség` attribútummal. Ennek az attribútumnak az értéke a megszerzett végzettségek gyűjteménye. Minden végzettséghez tartozik egy név (a megszerzett fokozat neve), egy iskola (ahol megszerezte) és egy dátum (amikor megszerezte az adott személy). A végzettséghez tartozó kollekciótípus lehet halmaz, multihalmaz, lista és tömb. Írjuk le a következményeit bármelyik választásának! Milyen információkat nyerünk vagy vesztünk a különböző kollekciótípusok választásával? Az elvesztett információ fontos-e a gyakorlatban?

4.9.6. feladat. A 4.4.4. feladatban két olyan példát láttunk, melyben a gyenge egyedhalmazok fontosak voltak. Adjuk meg ezeket az adatbázisokat ODL-ben, a megfelelő kulcsokkal együtt.

4.9.7. feladat. Adjuk meg a 4.1.9. feladatban leírt egyetemi adatbázis ODL-tervét.

!! **4.9.8. feladat.** Milyen körülmények között lehet egy kapcsolat önmaga inverze? *Útmutatás:* Gondoljunk a kapcsolatokra, mint párok halmazára, ahogy a 4.9.4. alfejezetben láttuk.

4.10. ODL-sémák átírása relációsémákká

Az ODL-t eredetileg is úgy tervezték, hogy – az SQL `CREATE TABLE` utasításához hasonlóan – az objektumorientált adatbázisok leíró nyelve legyen. Valójában van néhány kísérlet ilyen rendszer megvalósítására. Ugyanakkor az ODL úgy is tekinthető, mint egy szöveg alapú magas szintű tervező nyelv, melyből történetesen levezethető a relációs adatbázisséma. Ebben a fejezetben áttekintjük, hogy az ODL-tervek hogyan alakíthatók át relációtervekké.

A folyamat sokban hasonlít a 4.5. alfejezetben bemutatotthoz, amikor az E/K-diagramok relációs adatbázissémába való átkonvertálásával foglalkoztunk, és a 4.8. alfejezetben tárgyalt UML-konverzióhoz. Az osztályokból relációkat, a kapcsolatokhoz pedig a kapcsolatban lévő osztályok kulcsattribútumaiból álló relációkat képezünk. Az ODL kapcsán néhány új probléma jelenik meg, köztük:

1. Az egyedhalmazoknak kell hogy legyen kulcsuk, ODL-osztályok esetében ez nem garantált.

2. Az E/K-modellben, az UML-ben és a relációs modellben az attribútumok atomisága előírás volt, az ODL attribútumaira ez az előírás nem vonatkozik.

4.10.1. Az ODL-osztályoktól a relációkig

Kiindulási pontként tegyük fel, hogy az a célunk, hogy minden osztályhoz legyen egy reláció, és az adott relációban minden tulajdonsághoz legyen egy attribútum. Látni fogjuk, hogy több szempontból is módosítanunk kell ezt a megközelítést, de ebben a pillanatban vegyük a lehető legegyszerűbb esetet, amikor az osztályok valóban átírhatók relációkká, a tulajdonságok pedig attribútumokká. A következő megkötéseket alkalmazzuk:

1. Az osztályok összes tulajdonságai attribútumok (nem kapcsolatok vagy metódusok).

2. Az attribútumok atomi típusúak (nem struktúrák vagy halmazok).

Ebben az esetben az ODL-osztály nagyban hasonlít az egyedhalmazhoz és az UML-osztályhoz. Ugyan az ODL-osztályoknak lehet hogy nincs kulcsa, de van objektumazonosítója. Így az objektum azonosítása céljából létrehozhatunk egy mesterséges attribútumot, és ez szolgálhat a reláció kulcsaként; ezt a megközelítést a 4.8.4. alfejezetben, az UML-lel kapcsolatban már tárgyaltuk.

4.55. példa. A 4.47. ábra a gyártásirányítók ODL-leírása. Nem adtunk meg kulcsot, és nem feltételezhetjük azt sem, hogy a `név` egyedileg meghatározná a gyártásirányítót (eltérően a színészektől, mert ők egyedi neveket választanak).

```
Class GyártásIrányító {
    attribute string név;
    attribute string cím;
    attribute integer nettóBevétel;
};
```

4.47. ábra. A `GyártásIrányító` osztály

Létrehozunk az osztály nevével megegyező nevű relációt. A relációnak négy attribútuma lesz: az osztály minden attribútumához egy és egy az objektumazonosítóhoz:

```
GyártásIrányító(azonosító, név, cím, nettóBevétel)
```

Az objektumazonosítót reprezentáló `azonosító` attribútumot használjuk kulcsként. □

4.10.2. Összetett attribútumok

Sajnos, még akkor is, ha az osztály tulajdonságai mind attribútumok, előfordulhatnak nehézségek az osztálynak a relációra történő átírása közben. Ennek az az oka, hogy az ODL-ben az attribútumok összetett típusúak is lehetnek, mint például struktúrák, halmazok, multihalmazok vagy listák. Másrészt pedig a relációs modell egyik alapelve, hogy a relációattribútumok atomi típusúak, mint például számok és karaktersorozatok. Így meg kell találnunk, hogyan reprezentálhatjuk az összetett attribútumokat relációkkal.

Azokat a rekordszerkezeteket a legegyszerűbb kezelni, amelyeknek a mezői maguk már atomiak. Egyszerűen csak bővítjük a struktúra definícióját úgy, hogy a rekordszerkezet mindegyik mezőjéhez elkészítünk a relációban egy attribútumot.

4.56. példa. Tekintsük a `Színész` osztály 4.48. ábrán látható deklarációját, ebben tulajdonságként csak attribútumok szerepelnek. A `név` attribútum atomi, a `lakcím` attribútum két, a `város` és `utca` mezőkből álló rekordszerkezet. Így ezt az osztályt a következő relációval reprezentáljuk:

```
Színész(név, város, utca)
```

```
class Színész (key név) {
    attribute string név;
    attribute Struct Cím
        {string város, string utca} lakcím;
};
```

4.48. ábra. Egy összetett attribútummal rendelkező osztály

A `név`, a kulcs, a `város` és `utca` attribútumok pedig a `lakcím` struktúrát reprezentálják. □

4.10.3. Halmazértékű attribútumok reprezentálása

A rekordszerkezet persze nem a legbonyolultabb attribútumféle, amit az ODL osztálydefiníciókban használhatunk. A 4.9.6. alfejezetben bemutatott `Set`, `Bag`, `List`, `Array` és `Dictionary` típuskonstruktorokkal is felépíthetünk értékeket. Mindegyik sajátos problémát jelent a relációs modellé transzformálás során. Részletesen csak a legáltalánosabbal, a `Set` (halmaz) konstruktorral foglalkozunk.

Az egyik megközelítés, ahogy reprezentálhatjuk egy A attribútumhoz tartozó halmaz értékeit úgy, hogy a halmaz minden értékéhez készítünk egy külön sort ezzel az A értékkel, és a sor többi attribútumához az objektum megfelelő értékeit vesszük fel. Ez a megközelítés működik, jóllehet normalizálatlan relációt hoz létre, ahogy azt a következő példában is láthatjuk.

```
class Színész (key név) {
   attribute string név;
   attribute Set<
          Struct Cím {string város, string utca}
               > lakcím;
   attribute Date születésiDátum;
};
```

4.49. ábra. Színészek lakcímhalmazzal és születési dátummal

4.57. példa. Tegyük fel, hogy a `Színész` osztályt úgy definiáltuk, hogy minden színészhez egy lakcímhalmazt és egy nem kulcs, atomi `születésiDátum`-ot tudunk bejegyezni, amint azt a 4.49. ábra mutatja. Így a `születésiDátum` a `Színész` reláció egy attribútuma lesz, tehát a reláció sémája:

```
Színész(név, város, utca, születésiDátum)
```

Szerencsétlen módon ez a reláció egy egész sor olyan anomáliát mutat, melyekről a 3.3.1. alfejezetben volt szó. Tegyük fel azt, hogy Carrie Fishernek van

egy lakása és tengerparti háza is, így őt a Színész reláció két sorával reprezentáljuk. Ha Harrison Fordnak a címhalmaza üres, akkor ő elő sem fog fordulni a Színész relációban. A Színész reláció egy tipikus részletét a 4.50. ábrán látjuk.

név	*város*	*utca*	*születésiDátum*
Carrie Fisher	Hollywood	123 Maple St.	9/9/99
Carrie Fisher	Malibu	5 Locust Ln.	9/9/99
Mark Hamill	Brentwood	456 Oak Rd.	8/8/88

4.50. ábra. Születési dátumok hozzávétele

A Színész osztály kulcsa a név, mivel most a több cím miatt egy színészt a relációban több sor reprezentál, így a név *nem* lehet a Színész reláció kulcsa. Most a reláció kulcsa a {név, város, utca}. Így a

$$\texttt{név} \rightarrow \texttt{születésiDátum}$$

funkcionális függés sérti a BCNF-előírásokat, a

$$\texttt{név} \twoheadrightarrow \texttt{város utca}$$

többértékű függőség pedig sérti a 4NF előírásait. □

Többféle lehetőségünk van arra, hogyan kezeljük az osztálydeklarációban más attribútumokkal (halmazértékűek vagy nem azok) együtt előforduló halmazértékű attribútumokat. Egyik megközelítés az, hogy az összes halmazértékű attribútumot elkülönítjük, ahogy az osztályok objektumai közötti sok-sok típusú kapcsolatok esetében is eljártunk.

Másik megközelítés szerint először egyszerűen elhelyezzük a reláció sémájában az összes (halmazértékűek vagy nem olyan) attribútumot. Azután alkalmazzuk a 3.3. és 3.6. alfejezetek normalizációs technikáit, melyek eredményeként megszüntethetjük a BCNF és a 4NF normálformák előírásait sértő állapotokat. Megjegyezzük, hogy a halmazértékű attribútumok egyszerű attribútumokkal való együttes előfordulása a BCNF-előírások megsértéséhez vezet, ahogy a 4.57. példában láthattuk. Ugyanazon osztálydeklarációban előforduló két halmazértékű attribútum pedig a 4NF előírásainak megsértését eredményezi, ha nem egyértékű attribútumok.

4.10.4. Egyéb típuskonstruktorok reprezentálása

A rekordszerkezeteken és halmazokon kívül az ODL-osztálydefiníció használhat Bag, List, Array vagy Dictionary szerkezeteket az összetett értékek konstrukciójához. A multihalmaz (bag) reprezentációjához, amikor egyetlen objektum n-szer szerepelhet a multihalmazban, nem vehetjük be egyszerűen ugyanazt a

sort n-szer a relációba,[10] hanem a relációsémát bővítjük egy újabb attribútummal, amellyel azt reprezentáljuk, hányszor fordulnak elő az egyes elemek a multihalmazban. Például tegyük fel, hogy a 4.49. ábrában a lakcím halmaz helyett multihalmazként szerepelne. Legyen például Carrie Fisher Hollywood, 123 Maple St. lakcíme kétszer, Malibu, 5 Locust Ln. lakcíme pedig háromszor. Ezt az alábbi reláció reprezentálja:

név	*város*	*utca*	*szám*
Carrie Fisher	Hollywood	123 Maple St.	2
Carrie Fisher	Malibu	5 Locust Ln.	3

A lakcímek listája egy új attribútummal, a `pozíció`-val reprezentálható, amely megmutatja, hogy az adott lakcím hányadikként szerepel a listában. Például Carrie Fisher lakcímeit listaként felsorolva, a hollywoodi lakcímet először véve az alábbi relációval írható le:

név	*város*	*utca*	*pozíció*
Carrie Fisher	Hollywood	123 Maple St.	1
Carrie Fisher	Malibu	5 Locust Ln.	2

Ha a lakcímek fix hosszúságú tömbök, akkor ezt a tömb minden pozíciójához hozzárendelt attribútumokkal tudjuk reprezentálni. Például, ha a `lakcím` két város-utca szerkezetű tömb, akkor a `Színész` objektumot a következő relációval reprezentáljuk:

név	*város1*	*utca1*	*város2*	*utca2*
Carrie Fisher	Hollywood	123 Maple St.	Malibu	5 Locust Ln.

Végül, a szótárat (`Dictionary`) halmazként reprezentáljuk a párok mindkét tagjával, a kulcs típusú és a hozzárendelt érték típusú komponensekkel. Például tegyük fel, hogy a színészek címe helyett az összes színészre vonatkozóan egy olyan szótárat kívánunk tárolni, mely a lakásaikra jelzálogot bejegyzőket tartalmazza. A szótár kulcsa a cím lesz, a hozzárendelt érték pedig a bank neve. A Carrie Fisher objektum elképzelhető értékei a szótárattribútumokkal kiegészítve:

név	*város*	*utca*	*jelzálog-tulajdonos*
Carrie Fisher	Hollywood	123 Maple St.	Bank of Burbank
Carrie Fisher	Malibu	5 Locust Ln.	Torrance Trust

[10] Precizebben, a 2.2. alfejezetben leírt absztrakt relációs modell nem engedi meg, hogy egy relációba teljesen megegyező sorok kerüljenek. Ugyanakkor az SQL-alapú adatbázis-kezelők ezzel szemben mégis megengedik a sorok ismétlődését, és ezáltal az SQL-ben használt relációk valójában nem is halmazok, hanem multihalmazok. Minderről az 5.1. és 6.4. alfejezetekben olvashatunk bővebben. Ha a sorok száma lényeges szempont, akkor azt ajánljuk, hogy az itt megadott sémát használjuk még akkor is, ha az adatbázis-kezelőnk meg is engedné az azonos sorokat.

Az ODL attribútumtípusai egynél több típuskonstruktorral is képezhetők. Ha a típus a szótáron kívül bármelyik kollekciótípus, amelyet struktúrára alkalmazunk (azaz struktúrák halmazaként), akkor a 4.10.3., illetve a 4.10.4. alfejezetekben ismertetett technikákat használjuk úgy, mint az atomi értékekből épülő struktúráknál, majd kicseréljük az elemi értékeket reprezentáló egyszerű attribútumokat a struktúrát reprezentáló attribútumok sorozatával. Ezt a stratégiát alkalmaztuk fentebb is abban a példában, amelyben a cím struktúra volt. A struktúrákra alkalmazott szótár esete hasonló, gyakorlásként meghagyjuk az olvasónak.

Számos ok van arra, hogy korlátozzuk a struktúra és kollekció típuskonstruktorok egymás utáni alkalmazásával létrehozható attribútumtípusok bonyolultságát. A 4.1.1. alfejezetben említettük, hogy az E/K-modell néhány változatában megengedik az attribútumtípusok ilyen általánosítását, az E/K-modell használatakor mi mégis az atomi attribútumtípusokra korlátoztuk magunkat. Azt javasoljuk, hogy ha ODL-tervezést kívánunk használni, akkor – a relációs adatbázissémába való egyszerűbb átalakíthatóság miatt – hasonlóan korlátoznunk kell magunkat. A feladatok között bemutatunk néhány olyan esetet, amelyek összetett attribútumtípusok használatához vezetnek.

4.10.5. ODL-kapcsolatok reprezentálása

Általában egy ODL-osztálydefiníció tartalmazza a többi ODL-osztállyal való kapcsolatokat is. Ahogy az E/K-modellben is, minden kapcsolathoz létrehozunk egy új, a két kapcsolatban lévő osztály kulcsaiból álló relációt. Az ODL-kapcsolat az inverzével párban kell hogy előforduljon, de a kapcsolat és inverz párja reprezentálásához elegendő egy relációt létrehoznunk.

Ha a kapcsolat sok-egy típusú, akkor lehetőségünk van arra, hogy a kapcsolat „sok" oldalához megkonstruált relációt kombináljuk a kapcsolattal. Ha így járunk el, akkor a két relációban, amelyekből egyet képeztünk, lesz közös kulcs, ahogy a 4.5.3. alfejezetben ezzel foglalkoztunk. Következésképpen nem okozza a BCNF sérülését, helyénvaló, általában követendő eljárás.

4.10.6. Feladatok

4.10.1. feladat. Alakítsuk át az alábbi feladatok ODL-terveit relációs adatbázissémákká.

a) 4.9.1. feladat;

b) 4.9.2. feladat (mind a négy változatára, amit az adott feladatban megadtunk);

c) 4.9.3. feladat;

d) 4.9.4. feladat;

e) 4.9.5. feladat.

! **4.10.2. feladat.** Tegyük fel, hogy egy szótárnak mind a kulcs-, mind az értékkészlet komponense atomi típusokból felépülő struktúra. Mutassuk meg, hogyan reprezentálhatunk relációval ilyen típusú attribútumot tartalmazó osztályt.

4.10.3. feladat. Említettük, hogy ha az attribútum típusa struktúrák kollekciójánál sokkal összetettebb, akkor nehézzé válik a relációba való konvertálása. Szükségessé válhat közbülső állapotok és a nekik megfelelő relációk megkonstruálása. A következő kérdéssorozatban egyre összetettebb típusokat vizsgálunk, és azt, hogyan reprezentálhatók relációkkal.

a) A *kártya* a `rang` (2, 3, ..., 10, Jung, Dáma, Király, Ász) és a `szín` (Treff, Káró, Kőr, Pikk) mezőkből álló struktúrával reprezentálható. Adjuk meg a `Kártya` típusú struktúra alkalmas definícióját. Ennek a definíciónak függetlennek kell lennie bármely osztály definíciójától, de mindegyik számára elérhetőnek kell lennie.

b) A *kéz* kártyák egy halmaza. A kártyák száma változhat. Adja meg a `Kéz` osztály deklarációját, az osztályhoz tartozó objektumok a kezek. Tehát ezen osztálynak kell legyen egy `aKéz` attribútuma, amelynek típusa kéz.

! *c)* A *b)* pontban létrehozott `Kéz` osztályt konvertáljuk relációsémává.

d) A *póker-kéz* öt kártyából álló halmaz. Erre vonatkozóan ismételjük meg a *b)* és *c)* feladatokat.

! *e)* A *leosztás* párok halmaza. Minden pár a játékos nevéből és a kézből (kezében lévő kártyák) áll. Deklaráljuk a `Leosztás` osztályt, amelynek objektumai a leosztások. Tehát ezen osztálynak kell legyen egy `aLeosztás` attribútuma, melynek típusa leosztás.

f) Ismételjük meg az *e)* feladatot, de a kézre vonatkozó olyan megkötéssel, hogy az pontosan öt kártyából kell hogy álljon.

g) Ismételjük meg az *e)* feladatot a leosztáshoz szótárat használva. Feltételezhetjük, hogy a játékosok neve egyedi.

!! *h)* Konvertáljuk az *e)* feladat osztálydeklarációját relációs adatbázissémába.

! *i)* Tegyük fel, hogy a leosztást – játékosok hozzárendelése nélkül – kártyák halmazaiból képzett halmazként definiáljuk. Javasolható, hogy a leosztást a `Leosztás(Lazonosító, kártya)` relációsémával reprezentáljuk, ami azt jelenti, hogy a kártya az `Lazonosító` értékével azonosított leosztás valamelyik kezének eleme volt. Mi a hiba, ha van egyáltalán hiba ebben a reprezentálásban? Hogyan oldható meg a probléma?

4.10.4. feladat. Tegyük fel, hogy a C osztály definíciója az alábbi:

```
class C (key a) {
    attribute string a;
    attribute T b;
}
```

ahol T valamilyen típus. Adjuk meg a C-ből levezetett relációk sémáját, és jelezzük a kulcsot képező attribútumaikat, ha T a következő:

a) `Set<Struct S {string f, string g}>`

! *b)* `Bag<Struct S {string f, string g}>`

! *c)* `List<Struct S {string f, string }>`

! *d)* `Dictionary<Struct K {string f, string g}, Struct R {string i, string j}>`

4.11. Összefoglalás

- *Egyed-kapcsolat modell:* Az E/K-modellben egyedhalmazokat adunk meg, köztük lévő kapcsolatokkal, valamint attribútumokat ezen egyedhalmazokhoz és kapcsolatokhoz. Az egyedhalmazok elemeit egyedeknek nevezzük.

- *Egyed-kapcsolat diagramok:* Téglalapokat, rombuszokat és oválisokat használunk az egyedhalmazok, a kapcsolatok és attribútumaik ábrázolásához.

- *Kapcsolatok típusa:* Bináris kapcsolat lehet sok-sok, sok-egy vagy egy-egy típusú. Egy-egy kapcsolatban bármely egyedhalmaz egyede legfeljebb egy egyeddel lehet kapcsolatban a másik egyedhalmazból. Sok-egy kapcsolatban a „sok" oldalán álló egyedhalmaz minden egyede a másik oldalnak legfeljebb egy egyedével állhat kapcsolatban. A sok-sok típusú kapcsolatokra nincs a fentiekhez hasonló megszorítás.

- *A jó terv:* Az adatbázis-tervezés megköveteli, hogy a valóságot pontosan reprezentáljuk, megfelelő elemeket válasszunk (például kapcsolatokat, attribútumokat), és elkerüljük a redundanciát, azaz ugyanazt a dolgot ne vegyük kétszer, vagy valamit ne indirekt vagy bonyolult módon használjunk.

- *Alosztályok:* Az E/K-modellben ún. öröklési („az-egy" típusú) kapcsolattal adhatjuk meg azt, hogy egy egyedhalmaz speciális esete egy másiknak. Egyedhalmazokat e kapcsolatokkal hierarchikus szerkezetbe sorolhatunk, ahol a közvetlen leszármazott mindig speciális esete a szülőnek. Az egyedek komponensei tetszőleges részfát alkothatnak, melynek a gyökér mindig eleme.

- *Gyenge egyedhalmazok:* Ezek egyedei csak kapcsolódó egyedhalmazok attribútumainak segítségével azonosíthatók. A gyenge egyedhalmazok megkülönböztetésére a dupla keret speciális jelölést használjuk.

- *Egyedhalmazok átírása relációkká:* Az egyedhalmazhoz megfeleltetett relációban az egyedhalmaz minden attribútumához tartozik egy attribútum. Kivéve, ha az E gyenge egyedhalmaz, amelyhez tartozó relációban olyan attribútumoknak is benne kell lenniük, amelyek más egyedhalmaz kulcsattribútumai és az E egyedek azonosítását segítik.

- *Kapcsolatok átírása relációkká:* Egy E/K-kapcsolat átírásával keletkezett reláció tartalmazza a kapcsolatban részt vevő minden egyes egyedhalmaz kulcsattribútumait. Ha egy kapcsolat valamely gyenge egyedhalmaz támogató kapcsolata, akkor nem szükséges hozzá külön relációt létrehozni.

- *Öröklési hierarchiák átírása relációkká:* Az egyik lehetőség az, hogy az egyedeket a hierarchia különböző egyedhalmazai szerint komponensekre bontjuk, és minden egyedhalmazhoz létrehozunk egy-egy relációt az összes szükséges attribútummal. A második lehetőség az, hogy az egyedhalmazok összes lehetséges részhalmazához (részfájához) létrehozunk egy-egy relációt, így minden egyedhez egy sor tartozik abban a relációban, amely pontosan azon egyedhalmazoknak felel meg, amelyhez az egyed hozzátartozik. A harmadik lehetőség az, hogy csupán egyetlen relációt hozunk létre, melyben nullértékeket szerepeltetünk azon attribútumokon, amelyek egy adott egyedet reprezentáló sorban nem értelmezettek.

- *UML (Unified Modeling Language – Egységesített Modellező Nyelv):* Az UML-ben osztályokat és az osztályok közötti társításokat írunk le. Az osztályok az E/K-modell egyedhalmazainak megfelelői, a társítások pedig az E/K bináris kapcsolataihoz hasonlók. A sok-egy kapcsolatok speciális fajtái az aggregációk és a kompozíciók; ezek a relációvá való átalakításkor adnak segítséget.

- *UML-osztályhierarchiák:* Az UML-ben az osztályoknak lehetnek alosztályai, ezek öröklik a szuperosztály tulajdonságait. Az osztály alosztályi rendszere lehet teljes vagy részleges, diszjunkt vagy átfedő.

- *Az UML-diagramok relációkká konvertálása:* A módszer hasonló ahhoz, amit az E/K-modellekhez használtunk. Az osztályokból relációkat képezünk, a társításokból a társított osztályok kulcsaiból álló relációt hozunk létre. Az aggregációk és a kompozíciók a kapcsolat „sok" végéhez létrehozott relációkba épülnek bele.

- *ODL (Object Definition Language – Objektum Definíciós Nyelv):* Adatbázisok sémájának objektumorientált stílusú formális leírására alkalmas nyelv. Osztályokat definiál, amelyek háromféle tulajdonsággal, nevezetesen attribútumokkal, metódusokkal és kapcsolatokkal rendelkezhetnek.

- *ODL-kapcsolatok:* ODL-ben a kapcsolatoknak binárisaknak kell lenniük. A kapcsolatot nevük reprezentálja, amelyet a két kapcsolatban levő osztályban egymás inverzeiként kell definiálni. A kapcsolat lehet sok-sok, sok-egy, vagy egy-egy típusú, attól függően, hogy a kapcsolatban lévő párok típusa egyszerű objektumként vagy objektumok halmazaként van deklarálva.

- *Az ODL típusrendszere:* Az ODL lehetővé teszi típusok konstruálását. Az osztálynevekkel és atomi típusokkal (mint például integer típus) kezdve a következő típuskonstruktorokat használhatjuk: struktúraképző, halmaz-, multihalmaz-, lista-, tömb- és szótárkonstruktorok.

- *Kulcsok az ODL-ben:* a kulcsok használata az ODL-ben nem kötelező. Megengedett, hogy egy vagy több kulcsot deklaráljunk, de minthogy az objektumoknak van objektumazonosítójuk, amely nem tartozik a (deklarációban felsorolt) tulajdonságaik közé, az ODL-t megvalósító rendszer különbséget tud tenni két objektum között akkor is, ha minden tulajdonságértékük azonos.

- *Az ODL-osztályok relációkká konvertálása:* A módszer megegyezik az E/K és az UML esetében használt módszerrel, kivéve amikor az osztálynak komplex típusú attribútuma is van. Ekkor előfordul, hogy az eredményreláció normalizálatlan lesz, és emiatt szét kell bontanunk (dekomponálni kell). Szükségessé válhat új attribútum létrehozása is az objektum azonosítójának reprezentálása céljából, ez fog kulcsként is szolgálni.

- *Az ODL-kapcsolatok relációkká konvertálása:* A módszer megegyezik az E/K-kapcsolatok esetében használt módszerrel, azzal a különbséggel, hogy először az ODL-kapcsolatokat az inverzeikkel összepárosítjuk, és e kapcsolatpár számára egyetlen relációt hozunk létre.

4.12. Irodalomjegyzék

Az E/K-modellt leíró eredeti cikk [5]. Az E/K-tervezéssel foglalkozó két könyv [2] és [7].

Az ODL leírását a [4] kézikönyv tartalmazza. Az objektumszemléletű adatbázisrendszerek történetéről többet az [1], [3], [2], [6] anyagokban találhatunk.

[1] F. Bancilhon, C. Delobel, P. Kanellakis, *Building an Object-Oriented Database System*, Morgan-Kaufmann, San Francisco, 1992.

[2] Carlo Batini, S. Ceri, S. B. Navathe, Carol Batini, *Conceptual Database Design: an Entity/Relationship Approach*, Addison-Wesley, Boston MA, 1991.

[3] R. G. G. Cattell, *Object Data Management*, Addison-Wesley, Reading, MA, 1994.

[4] R. G. G. Cattell (ed.), *The Object Database Standard: ODMG–99*, Morgan-Kaufmann, San Francisco, 1999.

[5] P. P. Chen, „The entity-relationship model: toward a unified view of data," *ACM Trans. on Database Systems* **1**:1, pp. 9–36, 1976.

[6] W. Kim (ed.), *Modern Database Systems: The Object Model, Interoperability, and Beyond*, ACM Press, New York, 1994.

[7] B. Thalheim, „Fundamentals of Entity-Relationship Modeling," Springer-Verlag, Berlin, 2000.

II. rész

Relációs adatbázisok programozása

5. fejezet

Algebrai és logikai lekérdező nyelvek

A jelen fejezet során a relációs adatbázisok modellezése helyett a programozásra fektetjük a hangsúlyt. A tárgyalást két absztrakt programozási nyelv leírásával kezdjük: egy algebraival, illetve egy logikán alapulóval. Az algebrai programozási nyelvet (vagy relációs algebrát) már a 2.4. alfejezetben bemutattuk, amely során láthattuk, milyen műveletek szerepelnek a relációs modellben. Most azonban többről van szó, mint az ott bemutatott algebráról, ezért a 2.4. alfejezetben tárgyalt halmaz alapú algebrát kiterjesztjük multihalmazokra. Ez a megfogalmazás már jobban fogja tükrözni a relációs modell gyakorlatban használt megvalósítási módjait. A kiterjesztés miatt a korábbiakon felül lesz még néhány újabb művelet is, mint például a reláció oszlopainak összegzése (például átlag).

A fejezetet egy másik, logikán alapuló lekérdező nyelvnek a formalizmusával zárjuk le. Ezt *Datalog*nak fogjuk nevezni. Ebben a nyelvben már megfogalmazhatunk a kívánt eredménynek megfelelő lekérdezéseket is anélkül, hogy az eredményt kiszámító olyan algoritmust adnánk, amely a relációs algebrai megközelítésnél elhanyagolhatatlan volt.

5.1. Relációs műveletek multihalmazokon

Ebben az alfejezetben a relációkat inkább multihalmaznak tekintjük, mint halmaznak. Éppen ezért ugyanaz a sor többször is megjelenhet egy adott relációban. Néhány relációs műveletet azonban át kell fogalmaznunk, amennyiben a relációkra multihalmazokként tekintünk. Első lépésben tekintsük az alábbi egyszerű példát, amelyben a reláció egy valódi multihalmaz, azaz nem egy halmaz.

5.1. példa. Az 5.1. ábrán látható reláció tulajdonképpen sorokból álló multihalmaz, amelyben az $(1, 2)$ sor háromszor és a $(3, 4)$ sor egyszer szerepel. Ha az 5.1. ábra relációját halmaznak tekintenénk, akkor ki kellene küszöbölni az $(1, 2)$ sor két megjelenését. A multihalmazként kezelt relációban megengedjük

ugyan, hogy ugyanazon sor többször szerepeljen, viszont a sorok sorrendje itt sem számít éppúgy, mint a halmazként kezelt relációk esetén. □

A	B
1	2
3	4
1	2
1	2

5.1. ábra. Egy multihalmaz

5.1.1. Mire jók a multihalmazok?

A forgalomban lévő ABKR-ek a relációkat multihalmazként valósítják meg (nem pedig halmazként), mint ahogy azt már korábban is említettük. A relációk hatékony megvalósítása szempontjából a relációk multihalmazokként való kezelése több módon is gyorsíthatja a relációs műveleteket. Például:

1. Ha két multihalmazként értelmezett reláció unióját vesszük, akkor az egyik reláció sorait egyszerűen lemásoljuk, és ehhez a másolathoz hozzáadjuk a másik reláció összes sorát. Ebben az esetben nem kell megszüntetnünk azon ismétlődő sorokat sem, amelyek mind a két relációban szerepelnek.

2. Ha a reláció vetítését halmazként fogjuk fel, akkor minden vetített sort össze kell hasonlítanunk az összes többi vetített sorral azért, hogy meggyőződjünk arról, hogy a vetítésben minden sor csak egyszer szerepel. Ezzel szemben, hogyha az eredményt multihalmaznak tekintjük, akkor minden sort levetítünk, és egyszerűen hozzáadjuk ezeket az eredményhez. Így egyetlen összehasonlítást sem kell végeznünk a vetítésből származó sorok között.

A	B	C
1	2	5
3	4	6
1	2	7
1	2	8

5.2. ábra. Az 5.2. példában szereplő multihalmaz

5.2. példa. Az 5.1. ábrán látható multihalmaz, akár az 5.2. ábrán látható reláció A és B attribútumokra történő vetítésének eredménye is lehetne, feltéve, ha megengednénk, hogy az $(1, 2)$ sor többször is szerepeljen az eredményben, azaz az eredményt multihalmazként kezelnénk. Ha a hagyományos relációs algebrai vetítési operátort használnánk, és ily módon kiküszöbölnénk az azonos sorokat az eredményből, akkor a következő relációt kapnánk:

A	B
1	2
3	4

Figyeljük meg, annak ellenére, hogy a multihalmazként kezelt eredmény nagyobb méretű, mégis gyorsabban ki lehet számolni, hiszen nem kell összehasonlítsuk az összes $(1, 2)$, illetve $(3, 4)$ sort a megelőző lépésben megkapott sorokkal. □

Egy másik motiváció a relációk multihalmazként történő kezelésére: előfordulhat olyan helyzet, mikor a kívánt választ csak úgy kaphatjuk meg, ha legalábbis átmenetileg multihalmazos megközelítést használunk. A következő példa ezt mutatja be.

5.3. példa. Ha például az 5.2. ábrán lévő halmazként kezelt reláció A oszlopára vonatkozó átlagot akarjuk venni, akkor nem használhatjuk a halmaz modell A-ra vonatkozó vetítését, mivel ez esetben az átlagra 2-t kapunk. Ugyanis az 5.2. ábrán az A-nak csak két különböző értéke van (3 és 1), ezeknek az átlaga pedig 2. Ha viszont az A oszlopra mint a $\{1, 3, 1, 1\}$ multihalmazra tekintünk, akkor a valódi átlagot kapjuk, amelynek értéke – az 5.2. ábra 4 sorának felhasználásával számolva – 1,5 lesz. □

5.1.2. Multihalmazok egyesítése, metszete, különbsége

Mind a három műveletnek új értelmezése lesz multihalmazok esetén. Tegyük fel, hogy R és S multihalmazok, és legyen t az R n-szer, illetve az S m-szer előforduló sora. Megengedjük azt is, hogy akár n, akár m, akár mindkettő 0 legyen. Ekkor:

- Az $R \cup S$ multihalmazon értelmezett egyesítésben a t sor $n + m$-szer fog előfordulni.
- Az $R \cap S$ multihalmazon értelmezett metszetben a t sor $\min(n, m)$-szer fog előfordulni.
- Az $R - S$ multihalmazon értelmezett különbségben a t sor $\max(0, n - m)$-szer fog előfordulni. Azaz ha a t többször fordul elő R-ben, mint S-ben, akkor t az $R - S$-ben annyiszor fordul elő, mint ahányszor előfordult R-ben mínusz ahányszor előfordul S-ben. Amennyiben viszont t legalább annyiszor előfordul S-ben, mint R-ben, akkor t nem lesz benne $R - S$ különbségben. Informálisan tehát t minden egyes előfordulása S-ben „semlegesíti" t egy előfordulását R-ben.

5.4. példa. Legyen R az 5.1. ábrán látható reláció, amely valójában egy multihalmaz, amelyben az $(1,2)$ sor kétszer és a $(3,4)$ sor egyszer szerepel. Legyen S a következő multihalmaz:

A	B
1	2
3	4
3	4
5	6

Ebben az esetben az $R \cup S$ egyesítés eredménye az a multihalmaz, amelyben az $(1,2)$ négyszer (háromszor r miatt és egyszer S miatt), a $(3,4)$ háromszor és az $(5,6)$ egyszer szerepel.

Az $R \cap S$ metszet eredménye a következő multihalmaz:

A	B
1	2
3	4

Mind az $(1,2)$, mind a $(3,4)$ sor csak egyszer szerepel az $R \cap S$ eredményben: Az $(1,2)$ háromszor szerepel R-ben és egyszer S-ben és $\min(3,1) = 1$; hasonló módon a $(3,4)$ sorra $\min(1,2) = 1$. Az $(5,6)$ sor egyszer szerepel S-ben és nem szerepel R-ben, $\min(0,1) = 0$, tehát az $R \cap S$ eredményben ez a sor nem szerepel.

Az $R - S$ különbség eredménye a következő multihalmaz:

A	B
1	2
1	2

Az $(1,2)$ háromszor szerepel R-ben és egyszer S-ben, $\max(0, 3-1) = 2$, ezért az $R - S$ különbségben ez a sor kétszer jelenik meg. A $(3,4)$ sor egyszer szerepel R-ben és kétszer szerepel S-ben, $\max(0, 1-2) = 0$, tehát ez a sor nem szerepel az $R - S$ különbségben. Mivel R-nek nincs más sora, ezért az $R - S$ eredmény sem tartalmazhat más sort.

Az $S - R$ különbség a következő multihalmaz:

A	B
3	4
5	6

A $(3,4)$ sor egyszer jelenik meg, hiszen ki kell vonni az S-beli előfordulások számából az R-beli előfordulások számát. Az $(5,6)$ sor ugyanilyen okból szerepel egyszer az eredményben. Ebben az esetben a multihalmazként kezelt eredmény történetesen halmaz. □

Multihalmaz-műveletek halmazokon

Képzeljük el, hogy van két halmazunk, R és S. Bármely halmaz tekinthető multihalmaznak, olyan multihalmaznak, amelynek történetesen mindegyik sora különböző. Tegyük fel, hogy az $R \cap S$ metszetet akarjuk kiszámolni, de R-et és S-et multihalmaznak tekintjük, és az ennek megfelelő szabályt alkalmazzuk. Ebben az esetben ugyanazt az eredményt kapjuk, mintha az R-et és S-et halmaznak tekintettük volna. Ha R-et és S-et multihalmaznak tekintjük, akkor egy t sor annyiszor szerepel majd az $R \cap S$ eredményben, amennyi a két multihalmazban található előfordulások minimuma. Mivel az R és S halmazok, ezért a t előfordulási száma 0 vagy 1. Azaz mindegy, hogy halmaz- vagy multihalmazszabállyal számoljuk ki a metszetet, a t sor legfeljebb egyszer szerepelhet az $R \cap S$ eredményben, és pontosan egyszer akkor fog megjelenni, ha mind az R-ben, mind az S-ben szerepel a t sor. Hasonló módon, ha az $R - S$ vagy az $S - R$ kiszámításához multihalmaz-különbséget használunk, akkor ugyanazt az eredményt kapjuk, mintha halmazkülönbséget használtunk volna.

Az egyesítés azonban különböző eredményt ad attól függően, hogy az R-et és az S-et halmaznak vagy multihalmaznak tekintjük. Ha multihalmazszabályt alkalmazunk az $R \cup S$ kiszámításához, akkor előfordulhat, hogy az eredmény nem halmaz lesz annak ellenére, hogy mind az R, mind az S halmaz. Azaz, ha a t sor az R-ben és az S-ben is szerepel, akkor multihalmazszabályt alkalmazva a t sor kétszer jelenik meg az $R \cup S$ eredményben, ha viszont halmazszabályt alkalmazunk, akkor a t sor csak egyszer jelenik meg az $R \cup S$ eredményben.

5.1.3. Multihalmazok vetítése

A multihalmazok vetítéséről már volt szó. Amint az 5.2. példában láthattuk, a vetítés során mindegyik sort külön kezeljük. Ha az R reláció az 5.2. ábrán látható multihalmaz, akkor $\pi_{A,B}(R)$ eredménye az 5.1. ábrán látható multihalmaz.

Ha vetítés során különböző sorokból egy vagy több attribútum elhagyásával egyforma sorokat kapunk, akkor ezeket az egyforma sorokat nem kell kiküszöbölni a multihalmaz vetítésének eredményéből. Ha az 5.2. ábra R relációjának $(1,2,5)$, $(1,2,7)$ és $(1,2,8)$ sorait levetítjük az A és B attribútumokra, akkor mindhárom sor ugyanazt az $(1,2)$ sort eredményezi. Ez azt jelenti, hogy a multihalmaz-eredményben az $(1,2)$ sor háromszor szerepel, míg a hagyományos vetítéssel csak egyszer szerepelne.

Algebrai azonosságok multihalmazok esetén

Az algebrai azonosság két relációs algebrai kifejezés közötti ekvivalencia. A kifejezések argumentumai relációk. Az ekvivalencia szerint mindegy, hogy milyen relációkat helyettesítünk be, a két kifejezés ugyanazt az eredményt adja. Egy jól ismert példa az egyesítésre vonatkozó azonosság: $R \cup S = S \cup R$. Ez az azonosság fennáll, ha az R és S relációkat halmazként kezeljük, de akkor is fennáll, ha a relációkat multihalmazként kezeljük. Létezik azonban számos olyan azonosság, amely fennáll abban az esetben, ha a relációs algebrai műveleteket a hagyományos halmazelméleti módon értelmezzük, viszont nem érvényesek, ha a relációkat multihalmazként kezeljük. Ilyen azonosság például a különbség és az egyesítés disztributív volta: $(R \cup S) - T = (R - T) \cup (S - T)$. Ez az azonosság érvényes halmazokra, viszont nem érvényes multihalmazokra. Hogy lássuk, miért is nem igaz multihalmazok esetén, tegyük fel, hogy a t sor egyszer szerepel mindhárom relációban, R-ben, S-ben és T-ben. Ebben az esetben a bal oldali kifejezés eredményében egyszer szerepel a t sor, a jobb oldali kifejezés eredményében viszont nem szerepel a t sor. Ha halmazszabályt alkalmazunk, akkor egyik oldal eredményében sem szerepel a t sor. Az 5.1.4. és 5.1.5. feladatokban megvizsgáljuk további algebrai azonosságok érvényességét multihalmazok esetén.

5.1.4. Multihalmazokon értelmezett kiválasztás

A multihalmazon végzett kiválasztás azt jelenti, hogy a kiválasztás feltételét minden egyes sorra alkalmazzuk. A multihalmazoknál megszokott módon itt sem küszöböljük ki az azonos sorokat.

5.5. példa. Legyen R a következő multihalmaz:

A	B	C
1	2	5
3	4	6
1	2	7
1	2	7

a $\sigma_{C \geq 6}(R)$ multihalmazos kiválasztás eredménye:

A	B	C
3	4	6
1	2	7
1	2	7

Azaz, az első sor kivételével mindegyik sor teljesíti a feltételt. A két utolsó sor R-ben is megegyezik, így mindkettő szerepel az eredményben is. □

5.1.5. Multihalmazok szorzata

A multihalmazok Descartes-szorzatára vonatkozó szabály pontosan az, amit vártunk: az első reláció minden egyes sorát össze kell párosítanunk a második reláció mindegyik sorával függetlenül attól, hogy az adott sor hányszor szerepel az adott relációban. Következésképpen, ha az r sor m-szer szerepel az R relációban és az s sor n-szer szerepel az S relációban, akkor az rs sor mn-szer szerepel az $R \times S$ szorzat eredményében.

5.6. példa. Legyen az R és S reláció az 5.3. ábrán látható két reláció. Ekkor az $R \times S$ szorzat hat sorból áll, amint az az 5.3 (c) ábrán is látható. Megjegyezzük, hogy a hagyományos szorzatnál bevezetett, attribútumnevekre vonatkozó jelölés multihalmazokra is kitűnően alkalmazható. Ekképpen az R és S relációkban egyaránt megtalálható B attribútum kétszer jelenik meg a szorzatban, és előtagként a megfelelő reláció neve szolgál. □

A	B
1	2
1	2

(a) Az R reláció

B	C
2	3
4	5
4	5

(b) Az S reláció

A	$R.B$	$S.B$	C
1	2	2	3
1	2	2	3
1	2	4	5
1	2	4	5
1	2	4	5
1	2	4	5

(c) Az $R \times S$ szorzat

5.3. ábra. Multihalmazok szorzatának kiszámítása

5.1.6. Multihalmazok összekapcsolása

A multihalmazok összekapcsolása szintén nem szolgál különösebb meglepetéssel. Az első reláció minden egyes sorát összehasonlítjuk a második reláció valamennyi sorával, és eldöntjük, hogy az így kapott sorpárok sikeresen összekapcsolhatók-e vagy sem. Ha az összekapcsolás elvégezhető, akkor az eredményből nem kell kiküszöbölni a megegyező sorokat.

5.7. példa. Az 5.3. ábrán látható R és S reláció természetes összekapcsolásának eredménye, $R \bowtie S$:

A	B	C
1	2	3
1	2	3

Az R reláció $(1,2)$ sora összekapcsolható az S $(2,3)$ sorával. Mivel az R relációban az $(1,2)$ sor kétszer szerepel és az S relációban a $(2,3)$ sor egyszer szerepel, ezért az eredményben az $(1,2,3)$ kétszer jelenik meg. Az R és S egyéb sorai nem kapcsolhatók össze.

Ugyanezeken a relációkon végezzünk most egy théta-összekapcsolást:

$$R \bowtie_{R.B<S.B} S$$

Az eredményül kapott multihalmaz:

A	$R.B$	$S.B$	C
1	2	4	5
1	2	4	5
1	2	4	5
1	2	4	5

Az összekapcsolást a következő módon végezzük: az R $(1,2)$ sora és az S $(4,5)$ sora megfelel a feltételnek. Mivel mindkét sor kétszer szerepel a megfelelő relációkban, ezért az összekapcsolt sor $2 \times 2 = 4$ alkalommal jelenik meg az eredményben. Ezenkívül még összekapcsolhatnánk az R $(1,2)$ sorát az S $(2,3)$ sorával, de ez a sorpár nem teljesíti a feltételt, ily módon az eredményben sem jelenik meg. □

5.1.7. Feladatok

5.1.1. feladat. Tekintsük a 2.21 (a) ábrán szereplő PC relációt. Számítsuk ki a $\pi_{\text{sebesség}}(\text{PC})$ kifejezést. Mi lesz az értéke, ha a relációkat halmazokként kezeljük? És ha multihalmazokként? Mennyi lesz az értékek átlaga ebben a projekcióban, ha a relációkat halmazokként kezeljük? Mennyi, ha multihalmazokként?

5.1.2. feladat. Végezzük el az 5.1.1. feladatot a $\pi_{\text{merevlemez}}(\text{PC})$ kifejezéssel.

5.1.3. feladat. Tekintsük a 2.4.3. feladatban szereplő „Csatahajó" relációt.

a) Tekintsük a $\pi_{\texttt{kaliber}}(\texttt{Hajóosztályok})$ egyoszlopos relációt adó kifejezést, amely az egyes osztályok kaliberét adja meg. Mi lesz ennek az értéke a 2.4.3. feladat adataival, ha a relációkat halmazoknak tekintjük? Mi, ha multihalmazoknak?

! *b)* Készítsünk egy relációs algebrai kifejezést, amely az egyes hajók kaliberét adja (nem az egyes osztályokét). A kifejezéshez használjunk multihalmazokat, azaz annyiszor szerepeljen a b mint kaliber, ahány b kaliberű hajó van.

! **5.1.4. feladat.** Bizonyos algebrai azonosságok, amelyek érvényesek a halmazokként kezelt relációkra, érvényesek a multihalmazokként kezelt relációkra is. Mutassuk meg, miért maradnak érvényben a következő azonosságok multihalmazokra is:

a) Az egyesítés asszociatív tulajdonsága: $(R \cup S) \cup T = R \cup (S \cup T)$.

b) A metszet asszociatív tulajdonsága: $(R \cap S) \cap T = R \cap (S \cap T)$.

c) A természetes összekapcsolás asszociatív tulajdonsága:

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

d) Az egyesítés kommutatív tulajdonsága: $R \cup S = S \cup R$.

e) A metszet kommutatív tulajdonsága: $R \cap S = S \cap R$.

f) A természetes összekapcsolás kommutatív tulajdonsága:

$$R \bowtie S = S \bowtie R$$

g) $\pi_L(R \cup S) = \pi_L(R) \cup \pi_L(S)$, ahol L az attribútumok tetszőleges listája

h) Az egyesítés és a metszet disztributív tulajdonsága:

$$R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$$

i) $\sigma_{C \text{ AND } D}(R) = \sigma_C(R) \cap \sigma_D(R)$, ahol C és D tetszőleges feltételek az R soraira.

!! **5.1.5. feladat.** A következő azonosságok érvényesek, ha a relációkat halmazokként kezeljük, de nem érvényesek, ha multihalmazokként. Mutassuk meg, hogy halmazokra miért érvényesek, és adjunk ellenpéldát multihalmazokra.

a) $(R \cap S) - T = R \cap (S - T)$.

b) A metszet disztributív tulajdonsága az egyesítés felett:

$$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$$

c) $\sigma_{C \text{ OR } D}(R) = \sigma_C(R) \cup \sigma_D(R)$, ahol C és D tetszőleges feltételek az R soraira.

5.2. Kiterjesztett műveletek a relációs algebrában

A klasszikus relációs algebrát a 2.4. alfejezetben már bemutattuk, illetve az olyan módosításokat is megtettük az 5.1. alfejezetben, amelyek elengedhetetlenek voltak ahhoz, hogy a korábbi halmazos szemlélettel szemben, most a relációkra mint sorok multihalmazára tekinthessünk. Ennek a két résznek az elképzelései képezik a legtöbb korszerű lekérdező nyelv alapjait. Habár az olyan nyelvek, mint az SQL, számos olyan, ezektől eltérő műveletet is tartalmaznak, amelyek alkalmazások írásánál nagyon fontosak lehetnek. Ezért a relációs műveletek egy teljesebb kezelési megközelítésében szerepelni kell még néhány más műveletnek is, amelyeket ezen alfejezet során be is fogunk vezetni. A kiegészítések a következők:

1. *Az ismétlődések megszüntetésének művelete: δ.* Ez a művelet a multihalmazt halmazzá alakítja a sorok másolatainak megszüntetésével.

2. *Az összesítő műveletek.* Ilyen például az összegzés, illetve az átlag, amelyek ugyan nem a relációs algebra műveletei, de csoportosító műveletekkel használhatók (ezt később részletezzük). Az összesítő műveletek egy reláció attribútumaira (oszlopaira) vannak értelmezve. Például egy oszlopra vonatkozó összegzés értéke azt a számot reprezentálja, amelyet az oszlopban szereplő értékek összeadása által kapunk meg.

3. *Csoportosítás.* A sorok csoportosítása egy reláció sorainak „csoportokba" történő besorolása a reláció egy vagy több attribútumának értékétől függően. Ezek után már az egyes csoportok oszlopaira összesítési művelet is végezhető, amely lehetővé teszi számunkra néhány olyan lekérdezés megfogalmazását is, amelyeket a klasszikus relációs algebrával nem tudtunk kifejezni. A γ *csoportosítási művelet* egy olyan művelet, amely a csoportosítás és az összesítés hatását kombinálja.

4. *Kiterjesztett vetítés művelet.* Ez kiterjeszti a π művelet hatását azzal, hogy a néhány oszlopra történő vetítés mellett azt is lehetővé teszi, hogy az érintett oszlopok valamilyen összesítési relációja alapján új oszlopok kiszámítását is elvégezhessük.

5. *Rendezési művelet.* A τ egy relációt a sorainak egy vagy több attribútumtól függő rendezett listájává alakít. Megfontoltan kell bánni ezzel az operátorral, hiszen a relációs algebra néhány művelete nincs értelmezve listára. Ezzel szemben a vetítés és a kiválasztás elvégezhető listákra is, sőt az eredményben a lista elemeinek sorrendje is megkövetelhető.

6. *Külső összekapcsolás.* Az összekapcsolás egyik fajtája, amely a lógó sorokat is megőrzi. A lógó sorok `NULL` értékekkel „egészülnek ki" a külső összekapcsolás eredményében, tehát a lógó sorok is szerepelnek a végeredményben.

5.2.1. Ismétlődések megszüntetése

Néhány esetben szükségünk lehet olyan műveletre, amely a multihalmazból halmazt állít elő. A $\delta(R)$ kifejezést használjuk arra, hogy olyan halmazt kapjunk, amely R relációnak csak a különböző sorait tartalmazza.

5.8. példa. Tekintsük az 5.1. ábra R relációját:

A	B
1	2
3	4
1	2
1	2

Ekkor $\delta(R)$ a következő:

A	B
1	2
3	4

Figyeljük meg, hogy az $(1, 2)$ sor, amely $\delta(R)$-ben csak egyszer fordul elő, R-ben háromszor is szerepelt. □

5.2.2. Összesítési műveletek

Több olyan művelet is létezik, amely alkalmazható számok vagy karakterláncok halmazaira vagy multihalmazaira. Ezek a műveletek összegzik vagy összesítik a reláció egy oszlopában szereplő értékeket. Ezen tulajdonságuk miatt ezeket összefoglalóan *összesítési műveletek*nek nevezzük. A legáltalánosabb műveletek ezek közül az alábbiak:

1. `SUM`, az oszlop értékeinek összegét határozza meg.
2. `AVG`, az oszlop értékeinek átlagát határozza meg.
3. `MIN`, illetve `MAX` az oszlop értékeinek minimumát, illetve maximumát határozza meg. Amennyiben karakterláncokat tartalmazó oszlopra alkalmazzuk, akkor a lexikografikusan (ábécé szerinti) legelső, illetve legutolsó értéket határozzák meg.
4. `COUNT`, az oszlopban található (nem feltétlenül különböző) elemek számát határozza meg. Más szavakkal a `COUNT` egy reláció bármely attribútumára alkalmazva megadja a reláció sorainak a számát, beleértve az ismétlődéseket is.

5.9. példa. Tekintsük az alábbi relációt:

A	*B*
1	2
3	4
1	2
1	2

Tekintsünk néhány példát az adott reláció attribútumain történő összesítésekre:

1. `SUM(B)` $= 2 + 4 + 2 + 2 = 10$.
2. `AVG(A)` $= (1 + 3 + 1 + 1)/4 = 1,5$.
3. `MIN(A)` $= 1$.
4. `MAX(B)` $= 4$.
5. `COUNT(A)` $= 4$.

□

5.2.3. Csoportosítás

Néha nem egyszerűen egy oszlop összesítésére van szükségünk, hanem a reláció sorait kell csoportosítanunk egy vagy több oszlop értékei szerint, és így csoportonként összesíthetünk. Például szeretnénk meghatározni mindegyik stúdió által előállított filmpercek összegét stúdiónként, azaz egy, az alábbihoz hasonló relációt:

stúdióNév	*hossz*
Disney	12345
MGM	54321
...	...

Induljunk ki a 2.2.8. alfejezetben szereplő adatbázissémából:

`Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)`

Ezért csoportosítanunk kell a `Filmek` reláció sorait a `stúdióNév` szerint, majd csoportonként ki kell számolni a `hossz` összegét. Azaz tegyük fel, hogy a `Filmek` sorai az 5.4. ábrán szereplő alakban vannak, és csoportonként alkalmazzuk a `SUM(hossz)` összesítést.

	stúdióNév	
	Disney	
	Disney	
	Disney	
	MGM	
	MGM	
	○	
	○	
	○	

5.4. ábra. Egy reláció az elképzelt csoportfelosztásról

5.2.4. A csoportosítási művelet

Most már bevezethetünk egy olyan műveletet, ami lehetővé teszi egy reláció csoportokra osztását, illetve néhány oszlopra vonatkozó összesítést. Ha van csoportosítás, akkor az összesítés az egyes csoportokon belül értendő.

A γ alsó indexeként szereplő L elemeknek egy listája, ahol egy elem az alábbiak közül bármelyik lehet:

a) Az R reláció egy attribútuma, ahol γ R-re van alkalmazva. Azaz egyike R olyan attribútumainak, amelyre a csoportosítást végeztük. Ezt az elemet *csoportosítási attribútum*nak nevezzük.

b) A reláció valamelyik attribútumára vonatkozó összesítési művelet. Ha az összesítés eredményére névvel szeretnénk hivatkozni, akkor egy nyilat és egy új nevet kell utána írnunk. A kiindulásul szolgáló attribútumot *összesítési attribútum*nak nevezzük.

Az $\gamma_L(R)$ kifejezés eredményrelációjának felépítése a következő:

1. Osszuk R sorait *csoportok*ba. Egy csoport azokat a sorokat tartalmazza, amelyeknek az L listán szereplő csoportosítási attribútumokhoz tartozó értékei megegyeznek. Ha nincs csoportosítási attribútum, akkor az egész R reláció egy csoportot képez.

2. Minden csoporthoz hozzunk létre olyan sort, amelyik tartalmazza:

   i) A szóban forgó csoport csoportosítási attribútumait.

   ii) Az L lista összesítési attribútumaira vonatkozó összesítéseket. (Az adott csoport összes sorára.)

5.10. példa. Tegyük fel, hogy adott az alábbi reláció:

```
SzerepelBenne(filmCím, filmÉv, színészNév)
```

A δ speciális esete γ-nak

Technikailag a δ művelet redundáns. Ha $R(A_1, A_2, \ldots, A_n)$ egy reláció, akkor $\delta(R)$ megegyezik a $\gamma_{A_1,A_2,\ldots,A_n}(R)$ kifejezéssel. Azaz az ismétlések megszüntetéséhez csoportosítást végzünk az attribútumokra, de nem összegezzük azokat. Ekkor minden csoportnak egy sor felel meg, amelyek R-ben egynél többször is előfordulhattak. Mivel a γ eredménye valójában csak egy sort tartalmaz a csoportokra, ezért a „csoportosításunk" eredményeként az ismétlődések is megszűnnek. A δ viszont egy elterjedt és fontos művelet, ezért célszerű továbbra is külön vizsgálnunk a műveletek algebrai szabályokkal és algoritmusokkal történő megvalósítása során.

Azt is láthatjuk, hogy γ a halmazon alkalmazott vetítésművelet kiterjesztése. Azaz $\gamma_{A_1,A_2,\ldots,A_n}(R)$ eredménye ugyanaz, mint a $\pi_{A_1,A_2,\ldots,A_n}(R)$ kifejezésé, ha R halmaz. Ha viszont R multihalmaz, akkor a γ megszünteti az ismétlődéseket, míg a π nem.

Meg szeretnénk találni minden olyan színészt, aki már szerepelt legalább három filmben, illetve a hozzá tartozó első film dátumát is. Az első lépésben csoportosítanunk kell a `színészNév` mezővel, mint csoportosítási attribútummal. Mindegyik csoportra ki kell számolnunk a `MIN(filmÉv)` összesítést. Emellett viszont ki kell számítanunk a `COUNT(filmCím)` összesítést is minden csoportra ahhoz, hogy eldönthessük, mely csoport elégíti ki a legalább három filmben való szereplés feltételeit.

Elsőként írjuk fel a csoportosító kifejezést:

$$\gamma_{\texttt{színészNév, MIN(filmÉv)}\rightarrow\texttt{elsőFilmÉv, COUNT(filmCím)}\rightarrow\texttt{filmekSzáma}}(\texttt{SzerepelBenne})$$

A kifejezés első két oszlopa kelleni fog a végeredményben is. A harmadik oszlop egy segédattribútum (amelyet `filmekSzámá`-nak neveztünk) annak meghatározására, hogy az adott színész szerepelt-e már legalább három filmben. Azaz a lekérdezésnek megfelelő algebrai kifejezést egy `filmekSzáma >= 3` feltétellel történő kiválasztással és az első két oszlopra történő vetítéssel kell kiegészítenünk. A lekérdezés kifejezésfáját az 5.5. ábrán tekinthetjük meg. □

5.2.5. A vetítés művelet kiterjesztése

Tekintsük át újra a 2.4.5. alfejezetben bevezetett $\pi_L(R)$ vetítési műveletet. A klasszikus relációs algebrában L az R reláció néhány attribútumának listájaként értelmezhető. Most pedig terjesszük ki a vetítés műveletét úgy, hogy lehetővé tegye számítások elvégzését is a sorok választott komponenseivel. A *kiterjesztett vetítés* – amelynek jelölése szintén $\pi_L(R)$ – vetítési listájában a következő típusú elemek szerepelhetnek:

5.5. ábra. Az 5.10. példa lekérdezésének algebrai kifejezésfája

1. Az R reláció egy attribútuma.
2. Egy $x \rightarrow y$ kifejezés, ahol x és y attribútumneveket jelölnek. Az L lista $x \rightarrow y$ eleme lekérdezi az R x attribútumát és y-ra *nevezi át* az oszlop nevét, azaz az eredmény sémájában ezen attribútum neve y lesz.
3. Egy $E \rightarrow z$ kifejezés, ahol E az R reláció attribútumaira vonatkozó (konstansokat, aritmetikai műveleteket, illetve karakterlánc-műveleteket tartalmazó) kifejezés, z pedig az E kifejezés alapján számolt, az eredményekhez tartozó új attribútumnak a nevét jelöli. Példaként tekintsük az $a + b \rightarrow x$ kifejezést a lista elemének, ekkor ez az a és b attribútumok összegét jelöli, amelynek a neve x lesz. A $c \,||\, d \rightarrow e$ elem jelentése pedig az, hogy c és d karakterláncként értelmezett attribútumokat összefűzzük, majd az eredményül kapott oszlopot e-nek nevezzük el.

A vetítés kiszámítása R összes sorának figyelembevételével történik. Az L lista kiértékelése a sor azon komponenseinek behelyettesítésével történik, amelyek szerepeltek L attribútumai között. A behelyettesítéskor az L-ben szereplő műveleteket is elvégezzük. Az eredmény egy olyan reláció lesz, amelynek a sémáját az L listában szereplő nevek (illetve átnevezések) alkotják. R minden egyes sora egy sort eredményez a végeredményben. Ezért az R relációban többször előforduló sorok az eredményben is többször fordulnak elő. A végeredményben viszont akkor is lehetnek ismétlődések, ha R-ben eredetileg nem voltak.

5.11. példa. Legyen az R reláció:

A	B	C
0	1	2
0	1	2
3	4	5

Ekkor $\pi_{A,B+C\to X}(R)$ értéke a következő:

A	X
0	3
0	3
3	9

Az eredmény sémájának két attribútuma van. Az első az A, ami R attribútuma volt ugyanezzel a névvel. A második az X, ami R második és harmadik attribútumának összegét reprezentálja.

Másik példaként tekinthetjük $\pi_{B-A\to X,C-B\to Y}(R)$ értékét, azaz:

X	Y
1	1
1	1
1	1

Figyeljük meg, hogy ennek a vetítéslistának a kiszámításánál a $(0,1,2)$ és a $(3,4,5)$ sorokat ugyanazon $(1,1)$ sorba képezzük le. Ezért ez a sor jelenik meg háromszor az eredményben. □

5.2.6. Rendezési művelet

Jó néhány esetben elképzelhető, hogy egy reláció sorait egy vagy több attribútuma alapján szeretnénk rendezni. Lekérdezéseink során gyakran megköveteljük az eredményreláció rendezettségét. Tegyük fel például, hogy Sean Connery filmjeit szeretnénk lekérdezni, és elvárjuk, hogy a kapott lista filmcím szerint rendezett legyen azért, hogy egy konkrét filmet gyorsabban megtalálhassunk. A lekérdezések optimalizálásának vizsgálatánál majd láthatjuk, hogyan válik gyakran hatékonyabbá az ABKR-ben egy lekérdezés végrehajtása, ha a relációt először rendezzük.

A $\tau_L(R)$ kifejezés, ahol R egy relációt, L pedig az R reláció attribútumait tartalmazó listát jelenti, magát az R relációt határozza meg, csak már az L lista alapján meghatározott rendezett alakban. Ha az L lista tartalma $A_1, A_2, \ldots, A_n$, akkor R sorait először A_1 értékei szerint fogja rendezni. A megmaradt csomókat A_2 értéke szerint tovább bontja, majd az olyan soroknál, amelyek mind A_1, mind A_2 szerint is azonos rendezettségűek, az A_3 értéke szerint rendezi és így tovább. Azokat a csomókat, amelyek A_n értéke szerinti rendezés után is megmaradtak, tetszőleges sorrendben helyezzük el.

5.12. példa. Ha tekintjük az $R(A,B,C)$ sémájú R relációt, akkor $\tau_{C,B}(R)$ az R sorait C érték szerint rendezi, majd az ugyanazon C értékkel rendelkező sorokat B szerint továbbrendezi. Ha mind a C, mind a B érték megegyezik, akkor tetszőleges lesz a sorrend. □

Amennyiben a τ által már rendezett eredményre egy olyan másik műveletet is elvégzünk, mint az összekapcsolás, akkor a korábbi rendezettség az esetek nagy többségében jelentését veszti, és ezután az eredményre is inkább multihalmazként érdemes tekinteni, nem pedig listaként. Ezzel szemben a multihalmazokon értelmezett vetítések megőrzik a rendezettséget. Sőt a listára alkalmazott kiválasztás eredményében, amely a kiválasztás feltételének nem megfelelő sorokat eldobja, a megmaradó sorok még mindig az eredeti rendezés szerinti sorrendjükben szerepelhetnek.

5.2.7. Külső összekapcsolások

Az összekapcsolás műveletének egyik tulajdonsága, hogy lehetnek lógó sorok, amelyek nem kapcsolhatóak össze más sorokkal, azaz nem lesz egyezés ezen sorok és a másik reláció sorai között a közös attribútumokon. A lógó sorokhoz nem tartozik sor az összekapcsolás eredményében, így az összekapcsolás nem biztos, hogy az eredeti reláció adatait hiánytalanul tükrözi. Az ilyen esetekben az összekapcsolás egy változatát, nevezetesen a „külső összekapcsolást" ajánlott alternatívaként használni. A külső összekapcsolás megtalálható a különféle, forgalomban lévő rendszerekben.

Először is tekintsük a „természetes" esetet, amelyben az összekapcsolás a benne szereplő két reláció közös attribútumaiban lévő értékeknek az egyenlőségén alapult. A $R \overset{\circ}{\bowtie} S$ alakú *külső összekapcsolás* először elvégzi az $R \bowtie S$ természetes összekapcsolást, majd ehhez hozzáadja az R és az S lógó sorait is. Az előbbi módon hozzáadott sort minden olyan attribútumában ki kell egészíteni egy speciális *null* szimbólummal ($\perp$), amellyel a sor ugyan nem rendelkezett, de az összekapcsolás eredményében már szerepel. Megjegyezzük, hogy $\perp$ szimbólum a `NULL` (a 2.3.4. alfejezetben szereplő) értéknek felel meg SQL-ben.

5.13. példa. Vegyük az 5.6 (a) ábra, illetve az 5.6 (b) ábra U és V relációját. Az U $(1, 2, 3)$ sora így V-nek mind a $(2, 3, 10)$, mind a $(2, 3, 11)$ sorával összekapcsolható. Így ez a három sor nem lógó. Viszont a fennmaradó V-beli és U-beli sorok lógók: a $(4, 5, 6)$, illetve a $(7, 8, 9)$ az U-ból és a $(6, 7, 12)$ a V-ből. Azaz, ezen három sor egyikének sincs olyan sorpárja a másik relációban, amellyel a B és C komponenseken is megegyezne. Ezért az 5.6 (c) ábrán szereplő $U \overset{\circ}{\bowtie} V$ eredményében a lógó sorok ki vannak egészítve egy $\perp$ szimbólummal ott, ahol nem rendelkeznek értékkel: a D attribútumnál az U sorainál és az A attribútumnál a V sorai esetén. □

Az alap (természetes) külső összekapcsolás elvének létezik több variánsa is. A $R \overset{\circ}{\bowtie}_L S$ *bal oldali külső összekapcsolás* annyiban különbözik a külső összekapcsolástól, hogy csak a bal oldalon szereplő R argumentum lógó sorainak $\perp$ szimbólummal kiegészített változatát adjuk hozzá az eredményhez. A $R \overset{\circ}{\bowtie}_R S$ *jobb oldali külső összekapcsolás* annyiban különbözik a külső összekapcsolástól, hogy csak a jobb oldalon szereplő S argumentum lógó sorainak $\perp$ szimbólummal kiegészített változatát adjuk hozzá az eredményhez.

A	B	C
1	2	3
4	5	6
7	8	9

(a) Az U reláció

B	C	D
2	3	10
2	3	11
6	7	12

(b) A V reláció

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	$\perp$
7	8	9	$\perp$
$\perp$	6	7	12

(c) Az $U \overset{\circ}{\bowtie} V$ eredményreláció

5.6. ábra. Relációk külső összekapcsolása

5.14. példa. Ha vesszük az 5.6. ábra U és V relációját, akkor $U \overset{\circ}{\bowtie}_L V$:

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	$\perp$
7	8	9	$\perp$

Az $U \overset{\circ}{\bowtie}_R V$ pedig:

A	B	C	D
1	2	3	10
1	2	3	11
$\perp$	6	7	12

□

A három természetes összekapcsolási műveleten túl a théta-összekapcsolás még hátramaradt, amely során először egy théta-összekapcsolást végzünk, majd az eredményéhez hozzáadjuk azokat a $\perp$ szimbólummal kiegészített sorokat is, amelyeket a théta-összekapcsolás feltételének ellenőrzése során nem tudunk a másik reláció egyetlen sorával sem társítani. A $\overset{\circ}{\bowtie}_C$ kifejezést használjuk a C feltétellel rendelkező théta-összekapcsolás jelölésére. Ez a művelet is módosítható L vagy R segítségével bal vagy jobb oldali külső összekapcsolássá.

5.15. példa. Legyenek U és V az 5.6. ábra relációi, és tekintsük a következőt:

$$U \overset{\circ}{\bowtie}_{A>V.C} V$$

Az U $(4,5,6)$ és $(7,8,9)$ sorai, illetve a V $(2,3,10)$ és $(2,3,11)$ sorai is kielégítik a feltételt. Ezért ezen négy sor egyike sem lesz nem társítható. Ezzel szemben a maradék két sor lógó lesz: az U $(1,2,3)$ sora és a V $(6,7,12)$ sora. Ezért kiegészítve fognak szerepelni az 5.7. ábrán szereplő eredményben. □

A	$U.B$	$U.C$	$V.B$	$V.C$	D
4	5	6	2	3	10
4	5	6	2	3	11
7	8	9	2	3	10
7	8	9	2	3	11
1	2	3	$\perp$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	6	7	12

5.7. ábra. A théta-összekapcsolás eredménye

5.2.8. Feladatok

5.2.1. feladat. Tekintsük az alábbi két relációt:

$R(A,B)$: $\{(0,1),\ (2,3),\ (0,1),\ (2,4),\ (3,4)\}$

$S(B,C)$: $\{(0,1),\ (2,4),\ (2,5),\ (3,4),\ (0,2),\ (3,4)\}$

Számítsuk ki a következőket: *a)* $\pi_{A+B,A^2,B^2}(R)$; *b)* $\pi_{B+1,C-1}(S)$; *c)* $\tau_{B,A}(R)$; *d)* $\tau_{B,C}(S)$; *e)* $\delta(R)$; *f)* $\delta(S)$; *g)* $\gamma_{A,\,\mathrm{SUM}(B)}(R)$; *h)* $\gamma_{B,\mathrm{AVG}(C)}(S)$; ! *i)* $\gamma_A(R)$; ! *j)* $\gamma_{A,\mathrm{MAX}(C)}(R \bowtie S)$; *k)* $R \overset{\circ}{\bowtie}_L S$; *l)* $R \overset{\circ}{\bowtie}_R S$; *m)* $R \overset{\circ}{\bowtie} S$; *n)* $R \overset{\circ}{\bowtie}_{R.B<S.B} S$.

! **5.2.2. feladat.** Egy f egyértékű műveletet *idempotens*nek nevezünk, ha bármely R relációra teljesül, hogy $f\big(f(R)\big) = f(R)$, azaz f többszöri alkalmazása ugyanazt az eredményt adja, mintha egyszer alkalmaztuk volna. A következő operátorok közül melyik lesz idempotens? Válaszához adjon számolási példát vagy indokolja meg.

a) δ; *b)* π_L; *c)* σ_C; *d)* γ_L; *e)* τ.

! **5.2.3. feladat.** Az egyik dolog, amit az eredeti 2.4.5. alfejezetben definiált vetítési művelettel szemben elérhetünk a kiterjesztett vetítési művelet segítségével, hogy megduplázhatunk oszlopokat. Például, ha $R(A, B)$ egy reláció, akkor $\pi_{A,A}(R)$ létrehoz egy (a, a) sort minden R-beli (a, b) sorra. Elvégezhető-e ez a művelet a klasszikus relációs algebra 2.4. alfejezetben leírt műveleteinek a segítségével? Indokolja válaszát.

5.3. Logika a relációkhoz

Az algebrán alapuló absztrakt lekérdező nyelvek alternatívájaként logikai forma is használható a lekérdezések leírására. A logikai lekérdezések nyelve a *Datalog („database logic")*, amely *if-then* szabályokat tartalmaz. Minden szabály azt az elvet fejezi ki, hogy az egyes relációkban a sorok bizonyos kombinációjából következtethetünk arra, hogy valamilyen másik sornak szerepelnie kell valamilyen másik relációban vagy egy lekérdezés eredményében.

5.3.1. Predikátumok és atomok

Datalogban a relációkat *predikátum*okkal reprezentáljuk. Minden predikátum rögzített számú argumentummal rendelkezik és egy predikátumot, amelyet az argumentumai követnek, *atom*nak nevezünk. Az atomok szintaxisa ugyanolyan, mint egy függvényhívásé a hagyományos programozási nyelvekben; például $P(x_1, x_2, \ldots, x_n)$ egy atom, amely tartalmazza a P predikátumot az $x_1, x_2, \ldots, x_n$ argumentumokkal.

Egy predikátum tulajdonképpen egy függvénynek a neve, ami logikai értékkel tér vissza. Ha R egy reláció valamilyen rögzített sorrendben lévő n argumentummal, akkor R-et használhatjuk, mint erre a relációra hivatkozó predikátumnevet. Az $R(a_1, a_2, \ldots, a_n)$ atom értéke `IGAZ` (`TRUE`), ha $(a_1, a_2, \ldots, a_n)$ az R egy sora; egyébként az értéke `HAMIS` (`FALSE`).

5.16. példa. Legyen R egy reláció:

A	B
1	2
3	4

Ekkor $R(1, 2)$ és $R(3, 4)$ is igaz. Viszont x és y bármilyen más értékére $R(x, y)$ hamis. □

Egy predikátumnak lehetnek változói, valamint konstansai argumentumként. Ha egy atomnak egy vagy több argumentuma változó, akkor az egy logikai visszatérési értékű függvény, amely ezen változókra vett értékekre `IGAZ` vagy `HAMIS` értékkel tér vissza.

5.17. példa. Ha R az 5.16. példában használt predikátum, akkor $R(x, y)$ egy függvény, amely azt fejezi ki bármely x-re és y-ra, hogy (x, y) az R reláció

egy sora-e vagy sem. Az R jelen előfordulására a 5.16. példában $R(x, y)$ `IGAZ` értékkel tér vissza, ha

1. $x = 1$ és $y = 2$, illetve
2. $x = 3$ és $y = 4$;

és `HAMIS`-sal egyébként. Egy másik példa az $R(1, z)$ atom, amely `IGAZ`-zal tér vissza, ha $z = 2$, és `HAMIS`-sal egyébként. □

5.3.2. Aritmetikai atomok

A Datalogban létezik egy másik fajta fontos atom, az *aritmetikai atom*. Ez a fajta atom tulajdonképpen két aritmetikai kifejezés összehasonlítása, mint például $x < y$ vagy $x + 1 \geq y + 4 \times z$. A kiemelés kedvéért az 5.3.1. alfejezetben bevezetett atomokat *relációs atomok*nak nevezzük (mindkettőt „atomoknak" nevezzük).

Figyeljük meg, hogy mind az aritmetikai és mind a relációs atomok argumentumaikban kaphatnak értéket, ezek az atomban előforduló változók értékei, és az atomok logikai visszatérési értékkel rendelkeznek. Valójában az aritmetikai összehasonlítások, mint a $<$ vagy a $\geq$, tulajdonképpen olyan relációk, amelyek tartalmazzák az összes igaz párt. Azaz a „$<$" relációt úgy képzelhetjük el, mint amely tartalmaz minden olyan sort – mint például az $(1, 2)$ vagy a $(-1,5,\ 65, 4)$ –, amelynek az első komponense kisebb a másodiknál. Ne feledjük, hogy az adatbázis-relációk mindig végesek, és általában időről időre változnak. Ezzel ellentétben az olyan aritmetikai összehasonlításból származó relációk, mint például a $<$, végtelenek és tartalmuk is változatlan.

5.3.3. Datalog-szabályok és lekérdezések

A Datalogban az 5.2. alfejezetben bemutatott klasszikus relációs algebrához hasonló műveletek leírása *szabályok*kal történik, amelyek a következőkből épülnek fel:

1. Egy *fej*nek nevezett relációs atom;
2. Ezt követi a $\leftarrow$ szimbólum, amelyet gyakran „if"-nek („ha") olvasunk ki;
3. Ez után következik a *törzs*, amely egy vagy több *részcél*nak nevezett atomból áll. A részcélok lehetnek relációs vagy aritmetikai atomok. A részcélokat `AND` („és") szócskával kapcsoljuk össze, és a részcélok elé, ha szükséges, kitehetjük a tagadást jelentő `NOT` logikai műveletet is.

5.18. példa. A következő Datalog-szabály:

```
HosszúFilm(fc,é) ← Filmek(fc,é,h,m,s,p) AND h ≥ 100
```

a „hosszú" filmek halmazát határozza meg (a legalább 100 percnél hosszabbakat). A fenti szabályban a standard `Filmek` relációnkra hivatkozunk, amelynek sémája a következő:

`Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)`

A szabály feje a *HosszúFilm(fc, é)* atom. A szabály törzse két részcélból áll:

1. Az első részcél a *Filmek* nevű predikátumból, mégpedig a `Filmek` reláció hat attribútumának megfelelő hat argumentumból áll. Az argumentumok különböző nevű változók: fc a `filmcím` komponens, $é$ az `év` komponens, h a `hossz` komponens stb. „Ezt a részcélt tekinthetjük úgy is, mint a `Filmek` reláció aktuális előfordulásának *(fc, é, h, sz, s, p)* sorát." Pontosabban fogalmazva, a *Filmek(fc, é, h, sz, s, p)* akkor igaz, ha a hat darab változó értéke megegyezik a `Filmek` reláció egy sorának hat komponensével.

2. A második részcél, $h \geq 100$, akkor igaz, ha a `Filmek` reláció megfelelő sorában a `hossz` komponens értéke legalább 100.

Az egész szabályt pedig a következőképpen tekinthetjük: akkor igaz a *HosszúFilm(fc, é)*, ha létezik a `Filmek` relációnak egy olyan sora, amelyre teljesülnek a következők:

a) A `filmcím` és `év` attribútumoknak megfelelő két első komponens értéke fc és $é$.

b) A `hossz` attribútumnak megfelelő harmadik komponens – azaz a h – értéke legalább 100.

c) A negyedik, ötödik és hatodik komponensek értéke tetszőleges.

Figyeljük meg, hogy ez a szabály ekvivalens a következő relációs algebrai megfeleltetésnek:

$$\texttt{HosszúFilm} := \pi_{\texttt{filmcím,év}}(\sigma_{\texttt{hossz}\geq 100}(\texttt{Filmek}))$$

a fenti megfeleltetés jobb oldala egy relációs algebrai kifejezés. □

Datalogban egy *lekérdezés* egy vagy több szabály együttese. Ha a szabály fej részében csak egy reláció van, akkor ezen reláció kiértékelése lesz a lekérdezés eredménye. Ekképpen az 5.18. példában megfogalmazott lekérdezés eredménye a `HosszúFilm` reláció. Ha egynél több relációt tartalmaznak a szabályfejek, akkor ezen relációk egyike lesz a lekérdezés eredménye, míg a maradék az eredmény megfogalmazásában segít. Meg kell jelölnünk, hogy melyik reláció adja meg a lekérdezésre a választ. Ezt megtehetjük például úgy, hogy ennek a relációnak speciális nevet adunk, például azt, hogy `Válasz`.

Név nélküli (anonymus) változók

A Datalog-szabályokban gyakran találhatók olyan változók, amelyek csak egyszer szerepelnek az adott szabályban. Éppen ezért teljesen mindegy, hogy ezeknek a változóknak mi a nevük. A változó neve akkor fontos, ha a változó többször is megjelenik a szabályban, hiszen ekkor a változó második, illetve további előfordulásakor is tudnunk kell, hogy ugyanarról a változóról van-e szó. Ezért megegyezés szerint azokat a változókat, amelyek csak egyszer jelennek meg, jelölhetjük egy aláhúzás jellel. Tehát egy atomban használhatunk _ jelet argumentumként. Az _ többszöri előfordulása különböző változókra utal, soha nem utalhat ugyanarra a változóra. Például, az 5.18. példában használt szabály felírható a következő módon:

```
HosszúFilm(fc,é) ← Filmek(fc,é,h,_,_,_) AND h ≥ 100
```

Az sz, s, p változók csak egyszer szerepelnek a szabályban, ezért ezek helyettesíthetők az aláhúzás jellel. A többi változót nem helyettesíthetjük, hiszen mindegyik kétszer szerepel a szabályban.

5.3.4. A Datalog-szabályok jelentése

Az 5.18. példa útmutatásul szolgált a Datalog-szabályok jelentésére vonatkozóan. Ha pontosabb képet akarunk kapni, képzeljük el, hogy a szabályban szereplő változók felveszik az összes lehetséges értéket. Amikor a változók értékei igazzá teszik az összes részcélt, akkor megkapjuk, hogy mi a fej értéke az aktuális változókra, és hozzáadjuk a kapott sort a fejben szereplő predikátumnak megfelelő relációhoz.

Képzeljük el például, hogy az 5.18. példának mind a hat változója felveszi az összes lehetséges értéket. Az összes részcél csak akkor lehet igaz, ha a $(fc, é, h, sz, s, p)$ értékek a megadott sorrendben a `Filmek` reláció egy sorát képezik. Sőt, mivel a $h \geq 100$ részcélnak is teljesülni kell, ezért a keresett sorban a `hossz` komponensnek megfelelő h változó értéke legalább 100 kell legyen. Ha ilyen értékkombinációt találunk, akkor a $(fc, é)$ sort betesszük a fejnek megfelelő `HosszúFilm` relációba.

Ha biztosítani szeretnénk, hogy egy szabály kiértékelésekor kapott reláció véges legyen, illetve ha értelmezni szeretnénk az aritmetikai és a *negált részcélok*at is, akkor bizonyos megszorításokat kell tennünk a változókra nézve. (Negált részcélnak nevezzük azt a részcélt, amely előtt `NOT` szerepel.) Az ilyen megszorítást *biztonságossági feltétel*nek nevezzük, és a következőképpen fogalmazhatjuk meg:

- A szabályban szereplő valamennyi változónak szerepelnie kell valamely nem negált, relációs részcélban is.

Ez azt jelenti, hogy a fejben, a negált relációs részcélokban és az aritmetikai részcélokban szereplő valamennyi változónak szerepelnie kell valamely nem negált relációs részcélban is.

5.19. példa. Tekintsük az 5.18. példában megfogalmazott szabályt:

```
HosszúFilm(fc,é) ← Filmek(fc,é,h,_,_,_) AND h ≥ 100
```

Az első részcél egy nem negált relációs részcél, és ráadásul tartalmazza a szabályban szereplő összes változót. A fejben szereplő két változó $(fc, é)$ szerepel az első részcél törzsében is. Ehhez hasonlóan, az aritmetikai részcélban szereplő h változó pedig megjelenik az első részcélban is. □

5.20. példa. Az alábbi szabály három dologban is megsérti a biztonságossági feltételt:

```
P(x,y) ← Q(x,z) AND NOT R(w,x,z) AND x<y
```

1. Az y változó szerepel a fejben, de nem jelenik meg egyetlen nem negált relációs részcélban sem. Figyeljük meg, hogy az y ugyan megjelenik az $x < y$ aritmetikai részcélban, de ettől a lehetséges y értékek halmaza nem lesz véges. Ha találunk a w, x és z változókhoz olyan a, b és c értékeket, amelyekre az első két részcél teljesül, a fejhez rendelt P relációhoz még mindig végtelen sok olyan (b, d) sor lesz hozzáadható, amelyre igaz, hogy $d > b$.

2. A w változó megjelenik egy negált relációs részcélban, viszont nem jelenik meg egyetlen nem negált relációs részcélban sem.

3. Az y változó megjelenik egy aritmetikai részcélban, viszont nem jelenik meg egyetlen nem negált relációs részcélban sem.

Látható tehát, hogy ez a szabály nem biztonságos, éppen ezért Datalogban nem használható. □

A szabályok jelentését más módon is meghatározhatjuk. Ahelyett hogy vennénk a változók összes lehetséges értékét, a nem negált relációs részcélok relációinak soraiból álló halmazt vegyük csak. A sorok olyan megfeleltetését, amely valamennyi nem negált részcélra nézve ugyanazt az értéket feleltei meg az ugyanolyan nevű változóknak, *konzisztens* megfeleltetésnek nevezzük. Figyeljük meg, hogy a biztonságosság miatt konzisztens megfeleltetéskor minden egyes változóhoz egy érték tartozik.

Minden egyes konzisztens megfeleltetéskor kiértékeljük a negált relációs részcélokat és az aritmetikai részcélokat, hogy megbizonyosodhassunk arról, hogy a változókhoz hozzárendelt értékek kielégítik-e ezeket a részcélokat. Ne feledjük, hogy egy negált részcél akkor igaz, ha a részcél atomja hamis. Ha mindegyik részcél igaz, akkor vesszük a fejben szereplő változók értékeiből álló sort, és ezt a sort hozzáadjuk a fej predikátumához rendelt relációhoz.

5.21. példa. Tekintsük a következő Datalog-szabályt:

```
P(x,y) ← Q(x,z) AND R(z,y) AND NOT Q(x,y)
```

Legyen a Q relációnak két sora, az $(1,2)$ és az $(1,3)$. Az R reláció sorai legyenek $(2,3)$ és $(3,1)$. Két nem negált relációs részcélunk van, a $Q(x,z)$ és az $R(z,y)$. Ezekhez a részcélokhoz vennünk kell a Q és R relációk sorainak összes lehetséges megfeleltetési kombinációját. Az 5.8. ábrán látható táblázat tartalmazza mind a négy lehetséges kombinációt.

	A `Q(x,z)` sorai	Az `R(z,y)` sorai	Konzisztens a megfeleltetés?	Igaz a `NOT Q(x,y)` részcél?	A fej eredménye
1.	$(1,2)$	$(2,3)$	Igen	Nem	—
2.	$(1,2)$	$(3,1)$	Nem; $z=2,3$	Irreleváns	—
3.	$(1,3)$	$(2,3)$	Nem; $z=3,2$	Irreleváns	—
4.	$(1,3)$	$(3,1)$	Igen	Igen	$P(1,1)$

5.8. ábra. A sorok összes lehetséges megfeleltetése a $Q(x,z)$ és $R(z,y)$ részcélokhoz

Az 5.8. ábrán látható második és harmadik megfeleltetés nem konzisztens, mert mindegyik két különböző értéket feleltet meg a z változónak. Ezért ezzel a két megfeleltetéssel nem kell tovább foglalkoznunk.

Az első megfeleltetésben a $Q(x,z)$ részcélhoz az $(1,2)$ sort feleltettük meg, az $R(z,y)$ részcélhoz pedig a $(2,3)$ sort. Ez a megfeleltetés konzisztens, hiszen akármelyik változót tekintjük, a megfeleltetés során egyetlen értéket kapott. Az x változó az 1, az y változó a 3 és a z változó a 2 értéket kapta. Ekképpen hozzákezdhetünk a nem negált, relációs részcéloktól különböző részcélok vizsgálatához. Csak egyetlen ilyen részcélunk van, a `NOT Q(x,y)`. A változók értékét behelyettesítve a `NOT Q(1,3)` kifejezést kapjuk, ami hamis, hiszen az $(1,3)$ sor benne van a Q relációban. Tehát ez a részcél hamis, és így nem keletkezett olyan sor, amelyre a fej igaz lenne (1.).

Az utolsó megfeleltetés (4.) szintén konzisztens, az x, y, z változók rendre az 1, 1, 3 értékeket kapták. A `NOT Q(x,y)` részcél ezúttal `NOT Q(1,1)` kifejezéssé alakul. Mivel az $(1,1)$ nem sora a Q relációnak, tehát ez a részcél ezúttal igaz. Ezután a fejben szereplő változókba behelyettesítjük a megfelelő értékeket, és a $P(1,1)$ kifejezést kapjuk. Az $(1,1)$ sort hozzáadjuk a P predikátumhoz rendelt relációhoz. Ily módon a sorok összes lehetséges megfeleltetését megvizsgáltuk, és a P relációba csak ez az egy sor került be. □

5.3.5. Extenzionális és intenzionális predikátumok

Ajánlatos különbséget tennünk az alábbi kétfajta predikátum között:

- *Extenzionális* predikátumok, amelyekhez rendelt relációk megtalálhatók az adatbázisban;

- *Intenzionális* predikátumok, amelyekhez rendelt relációkat egy vagy több Datalog-szabály kiértékelése alapján kapjuk meg.

Ez a különbség ugyanolyan, mint a relációs algebrai kifejezések operandusai és a relációs algebrai kifejezéssel számított relációk közötti eltérés. A relációs algebrai kifejezések operandusai „extenzionálisak", mivel a „hozzájuk tartozó" relációk aktuális előfordulásaival (*extenzió*val) vannak definiálva. A relációs algebrai kifejezéssel számított relációk pedig – legyenek akár néhány részkifejezéshez tartozó végleges vagy köztes eredmények – „intenzionálisak", hiszen tulajdonképpen a program írója hozta létre őket.

Datalog-szabályok esetén az extenzionális, illetve intenzionális predikátumokhoz rendelt relációkat extenzionális, illetve intenzionális relációknak nevezzük. Az intenzionális predikátumokra, illetve relációkra történő hivatkozás esetén használatos az *IDB* rövidítés (a rövidítés az „intensional database" angol kifejezésből ered). Az *EDB* rövidítést az extenzionális predikátumok, illetve relációk esetén használjuk (a rövidítés az „extensional database" angol kifejezésből ered).

Így az 5.18. példában a `Filmek` reláció egy EDB-reláció és a *Filmek* predikátum szintén EDB-predikátum. A `HosszúFilm` reláció és a *HosszúFilm* predikátum pedig egyaránt intenzionális.

Egy EDB-predikátum soha nem jelenhet meg egy szabály fejében sem, csak a törzsében. Az IDB-predikátumok szerepelhetnek a szabály fejében, a törzsében vagy mindkettőben egyaránt. Az is megszokott, hogy több szabály fejében is szerepeljen ugyanaz a predikátum. Az 5.24. példában két reláció egyesítését számoljuk ki ily módon.

Több intenzionális predikátum használatával az EDB-relációkból egyre bonyolultabb kifejezéseket építhetünk fel. Ez a folyamat hasonló a relációs algebrai kifejezések operátorokkal történő felépítéséhez.

5.3.6. Multihalmazokra vonatkozó Datalog-szabályok

A Datalog magában foglal egy halmazlogikát. Viszont amíg nincs negált relációs részcél, addig a Datalog-szabályok kiértékelésének elve elfogadható multihalmazokra nézve is, ha a relációk halmazok. Fogalmi szinten könnyebben használható a Datalog-szabályok kiértékelésének második megközelítése (amelyet az 5.3.4. alfejezetnél mutattunk be), amikor a relációk multihalmazok. Emlékeztetőként, ezen technika magában foglalta az összes nem negált relációs részcél vizsgálatát, illetve a reláció összes sorának a megfelelő részcél predikátumába történő behelyettesítését. Ha a részcélokból kiválasztott sorok minden változóra konzisztens értéket adnak és az aritmetikai részcélok is teljesülnek[1], akkor megnézhetjük, hogy ezen helyettesítési értékekre a fej mit ad. Az eredménysorokat hozzáadjuk a fejhez tartozó relációhoz.

[1] Megjegyezve, hogy a szabályban nem szerepelhet negált, relációs részcél. Ugyanis a multihalmazos modellben nincs világosan definiálva a negált, relációs részcélt tartalmazó Datalog-szabályok jelentése.

Mivel most multihalmazokkal foglalkozunk, most nem kell a fejben szereplő duplikátumokat megszüntetnünk. Sőt, mivel az összes részcélra a sorok minden lehetséges kombinációját figyelembe vesszük, ezért egy sor n-szeres előfordulását egy részcélhoz tartozó relációban n-szer tekintettük a részcél sorának a sorok összes kombinációjával együtt a többi részcélra nézve.

5.22. példa. Tekintsük az alábbi szabályt:

```
H(x,z) ← R(x,y) AND S(y,z)
```

ahol $R(A, B)$ reláció sorai:

A	B
1	2
1	2

és $S(B, C)$ sorai:

B	C
2	3
4	5
4	5

Az egyetlen eset, amikor konzisztens sor helyettesítését kapjuk a részcélokra (egy helyettesítés, amelyben y értéke minden részcélra ugyanaz), amikor az első részcél helyettesítése az R-ből származó $(1, 2)$ sor, a második részcél helyettesítése pedig az S-ből származó $(2, 3)$ sor. Mivel $(1, 2)$ kétszer fordul elő R-ben, $(2, 3)$ pedig egyszer S-ben, így a soroknak két helyettesítését kapjuk, melyek az $x = 1$, $y = 2$, $z = 3$ változóhelyettesítést szolgáltatják. A fejhez tartozó (x, z) sor minden behelyettesítésre $(1, 3)$. Az $(1, 3)$ sor tehát kétszer fordul elő a fejhez tartozó H relációban, és más sora nem is lesz. Azaz az alábbi reláció lesz a szabályból kapott fejhez tartozó reláció:

1	3
1	3

Általánosan úgy lehet megfogalmazni, hogy ha az $(1, 2)$ sor n-szer fordul elő R-ben, a $(2, 3)$ sor pedig m-szer fordul elő S-ben, akkor az $(1, 3)$ sor nm-szer fordul elő a H-ban. □

Ha egy relációt több szabály ír le, akkor az eredmény a szabályok által előállított összes sorból képzett multihalmazok uniója.

5.23. példa. Tekintsük a H relációt, amelyet két szabállyal definiálunk:

```
H(x,y) ← S(x,y) AND x>1
H(x,y) ← S(x,y) AND y<5
```

ahol az $S(B, C)$ reláció olyan, mint az 5.22. példában: $S = \{(2,3), (4,5), (4,5)\}$. Az első szabály S mindhárom sorát H-ba teszi, mivel mindegyik sor első komponense nagyobb 1-nél. A második szabály alapján csak a $(2,3)$ sor kerül be H-ba, mivel $(4,5)$-re nem teljesül, hogy $y < 5$. Azaz a H eredményreláció két darab $(2,3)$, illetve két darab $(4,5)$ sort tartalmaz. □

5.3.7. Feladatok

5.3.1. feladat. Írjuk fel Datalogban a 2.4.1. feladat valamennyi lekérdezését. Kizárólag biztonságos szabályokat használjunk. A bonyolultabb relációs algebrai részkifejezések megfogalmazásához használjunk IDB-predikátumokat.

5.3.2. feladat. Írjuk fel Datalogban a 2.4.3. feladat valamennyi lekérdezését. Itt is használhatunk IDB-predikátumokat, és most is kizárólag biztonságos szabályokkal dolgozzunk.

!! **5.3.3. feladat.** Ahhoz, hogy a fejhez rendelt reláció véges legyen abban az esetben, ha mindegyik részcél predikátumához rendelt reláció véges, a biztonságos Datalog-szabályokra adott feltétel elégséges, viszont túl erős. Adjunk példát olyan Datalog-szabályra, amely nem biztonságos és mégis teljesíti, hogy a relációs predikátumokhoz véges relációkat hozzárendelve a fejhez rendelt reláció is véges.

5.4. A relációs algebra és a Datalog

A 2.4. alfejezetben szereplő relációs algebrai operátorok mindegyike kifejezhető egy vagy több Datalog-szabály segítségével. Ebben az alfejezetben valamennyi operátort egyenként megvizsgáljuk. Meg kell vizsgálnunk, hogyan kombinálhatjuk a Datalog-szabályokat összetettebb algebrai kifejezések leírására. Az is igaz lesz, hogy bármely biztonságos Datalog-szabály kifejezhető lesz relációs algebrában is, habár mellőzni fogjuk ennek a bizonyítását. Ezek ellenére a Datalog-lekérdezések nagyobb kifejezőerővel bírnak, mint a relációs algebra, ha megengedjük néhány szabály kölcsönhatását is. Ebben az esetben kifejezhetjük a rekurziót is, amelyre a relációs algebrában nem volt lehetőségünk (lásd az 5.35. példánál).

5.4.1. Boole-műveletek

A relációs algebra Boole-műveletei (az unió, a metszet és a halmazkülönbség) könnyen kifejezhetők Datalogban. Használjuk a következő három módszert a leíráshoz. Az R és S relációkat ugyanannyi attribútummal rendelkezőnek tekintjük, számuk legyen n. A szükséges szabályok leírását minden esetben a `Válasz` nevű fejpredikátum használatával oldjuk meg. Tegyük ezt annak ellenére, hogy bármilyen nevet használhatnánk az eredmény leírására, és hogy különböző műveletek eredményéhez különböző predikátumokat kellene választanunk.

- Vegyük az $R \cup S$ uniót, használjunk a leíráshoz két szabályt és n különböző változót:

$$a_1, a_2, \ldots, a_n$$

Az egyik szabály csak az $R(a_1, a_2, \ldots, a_n)$ részcélból áll, míg a másik csak az $S(a_1, a_2, \ldots, a_n)$ részcélból. Legyen mindkét szabálynak a fejrésze $Válasz(a_1, a_2, \ldots, a_n)$. Így az eredmény az, hogy R összes sora és S összes sora is benne lesz a válaszrelációban.

- Vegyük az $R \cap S$ metszetet. Egyetlen szabály lesz, amelynek a törzse az alábbi:

$$R(a_1, a_2, \ldots, a_n) \texttt{ AND } S(a_1, a_2, \ldots, a_n)$$

A fej legyen a $Válasz(a_1, a_2, \ldots, a_n)$. Ebben az esetben egy sor pontosan akkor lesz benne a válaszrelációban, ha mind S-ben, mind R-ben benne van.

- Vegyük az $R - S$ különbséget. Egyetlen szabály lesz, amelynek a törzse az alábbi:

$$R(a_1, a_2, \ldots, a_n) \texttt{ AND NOT } S(a_1, a_2, \ldots, a_n)$$

A fej legyen a $Válasz(a_1, a_2, \ldots, a_n)$. Ekkor egy sor pontosan akkor lesz benne a válaszrelációban, ha benne lesz R-ben, de nem lesz benne S-ben.

5.24. példa. Legyenek az $R(A, B, C)$ és $S(A, B, C)$ sémák a két relációnk sémái. A félreértések elkerülése végett itt különböző predikátumneveket fogunk használni a végeredményekre ahelyett, hogy mindet `Válasz`-nak neveznénk.

Az $R \cup S$ meghatározásához a következő két szabályt használjuk:

```
1. U(x,y,z) ← R(x,y,z)
2. U(x,y,z) ← S(x,y,z)
```

Az 1. szabály fejezi ki, hogy R minden sora az U IDB-reláció egy sorát képezi. Hasonlóan a 2. szabály azt fejezi ki, hogy S minden sora benne van U-ban.

Az $R \cap S$ kiszámításához használjuk az alábbi szabályt:

```
I(a,b,c) ← R(a,b,c) AND S(a,b,c)
```

Végül a

```
D(a,b,c) ← R(a,b,c) AND NOT S(a,b,c)
```

szabály az $R - S$ halmazkülönbség kiszámítására való. □

Egy szabályban használt változónevek lokálisak

Meg kell jegyeznünk, hogy egy szabály felírásakor a változók nevei tetszőlegesek, és semmi kapcsolatuk nincs a többi szabályban használt változók neveivel. Ez azért lehetséges, mert minden egyes szabályt egyenként értékelünk ki, és a szabály fejéhez rendelt relációba a többi szabálytól függetlenül kerülnek be a megfelelő sorok.[a] Írjuk át például az 5.24. példában felírt második szabályt a következő módon:

$$\texttt{U(a,b,c)} \leftarrow \texttt{S(a,b,c)}$$

Az első szabályt hagyjuk meg változatlan formában. A két szabály természetesen így is az R és S relációk egyesítését adja meg. Figyeljük meg azonban, hogyha egy l nevű változó nevét kicseréljük d-re egy szabályban, akkor az l valamennyi előfordulását ki kell cserélnünk d-re ebben a szabályban. Továbbá az l változó nevét csak olyan névre cserélhetjük ki, amelyik nem szerepel máshol ebben a szabályban.

[a] Ezt a kiértékelést azért tehetjük meg, mert a Datalog-program is Horn-klóz program, amelyben a szabályok egymástól függetlenek, így a szabályok kiértékelésének sorrendje tetszőleges, mely Prolog-programokra nem érvényes. *(A lektor megjegyzése.)*

5.4.2. Vetítés

Az R reláció vetítésének kiszámításához egyetlen szabályt használunk. A szabálynak egyetlen részcélja van, az R predikátum. A részcél argumentumában a reláció különböző attribútumainak különböző változók felelnek meg. A fej argumentumában azoknak az attribútumoknak megfelelő változók szerepelnek, amelyekre a vetítés történik, és olyan sorrendben, ahogy a vetítésben megkívántuk.

5.25. példa. Tegyük fel, hogy a

`Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)`

relációt szeretnénk levetíteni az első három attribútumára. A

$$\texttt{P(fc,é,h)} \leftarrow \texttt{Filmek(fc,é,h,m,s,p)}$$

szabály pontosan a vetítés eredményének megfelelő P relációt adja meg. □

5.4.3. Kiválasztás

A kiválasztás operátor kifejezése Datalogban már bonyolultabb dolog, mint az eddigi operátorok átírása. A legegyszerűbb eset, amikor a kiválasztás feltétele egy vagy több aritmetikai összehasonlítás `AND` szócskával történő összekap-

csolásából áll. Ebben az esetben egyetlen szabályt kell készítenünk, amely a következőket fogja tartalmazni:

1. Egy relációs részcélt, amely megfelel annak a relációnak, amelyen a kiválasztás történik. Ez az atom annyi különböző változót tartalmaz, ahány attribútummal a reláció rendelkezik.

2. A kiválasztás minden egyes összehasonlításához megadunk egy aritmetikai részcélt, amely megegyezik az összehasonlítással. Amíg a kiválasztás feltételében attribútumnevek szerepelnek, addig az aritmetikai részcélokban azokat a változókat kell használnunk, amelyekkel az előző lépésben a relációs részcélt megadtuk.

5.26. példa. Az alábbi kiválasztást

$$\sigma_{\texttt{hossz} \geq 100 \texttt{ AND stúdióNév='Fox'}}(\texttt{Filmek})$$

a következő Datalog-szabállyal fejezhetjük ki:

```
S(fc,é,h,m,s,p) ← Filmek(fc,é,h,m,s,p) AND h ≥ 100 AND s = 'Fox'
```

A kiértékelés eredménye a K reláció. Figyeljük meg, hogy mindkét részcélban a h és s változók a `Filmek` reláció `hossz` és `stúdióNév` attribútumainak felelnek meg. □

Most pedig nézzük meg, mi történik azokkal a kiválasztásokkal, amelyek feltételében vannak olyan összehasonlítások, melyek `OR` (vagy) szócskával vannak összekapcsolva. Az ilyen kiválasztásokat nem tudjuk egyetlen Datalog-szabály segítségével kifejezni. Viszont az ilyen kiválasztások átalakíthatók több kiválasztás egyesítésévé. Azonban az olyan kiválasztás, amelynek feltétele két részfeltétel `OR` összekapcsolásából áll, ekvivalens két olyan kiválasztás egyesítésével, amelyek az egyik, illetve másik részfeltételhez tartoznak. Tehát az n feltétel `OR` szócskával történő összekapcsolását tartalmazó kiválasztás kifejezhető n darab, azonos fejű szabállyal. Az i-edik szabály a kiválasztás i-edik feltételének felel meg.

5.27. példa. Cseréljük ki az 5.26. példa kiválasztásában az `AND` szócskát `OR` szócskára:

$$\sigma_{\texttt{hossz} \geq 100 \texttt{ OR stúdióNév='Fox'}}(\texttt{Filmek})$$

Azaz, válasszuk ki azokat a filmeket, amelyek vagy hosszú filmek, vagy a Fox stúdióban készültek. Felírjuk a két feltételnek megfelelő két szabályt:

```
1. S(fc,é,h,m,s,p) ← Filmek(fc,é,h,m,s,p) AND l ≥ 100
2. S(fc,é,h,m,s,p) ← Filmek(fc,é,h,m,s,p) AND s = 'Fox'
```

Az első szabály megadja a 100 percnél hosszabb filmeket, a második pedig a Fox stúdióban készült filmeket. □

Az AND, OR és NOT logikai operátorok tetszőleges alkalmazásával már meglehetősen bonyolult kiválasztási feltételek is megfogalmazhatók. Egy jól ismert módszer segítségével (amelynek az ismertetésétől most eltekintünk), ezek a logikai kifejezések „diszjunktív normálformára" hozhatók. A diszjunktív normálforma „konjunkciók" OR szócskával történő összekapcsolása. Egy *konjunkció* literálok AND szócskával történő összekapcsolása, egy *literál* pedig egy negált vagy egy nem negált összehasonlítás.[2]

A literálokat részcélokkal tudjuk reprezentálni, amelyek előtt szerepelhet egy NOT szócska is. Negált aritmetikai részcél esetén a NOT szócska bevihető az összehasonlításba. Például a NOT x $\geq$ 100 feltétel felírható x < 100 alakban. Bármely konjunkció megfelel egyetlen olyan Datalog-szabálynak, amelynek részcéljai maguk az összehasonlítások. Végül, bármely diszjunktív normálformájú kifejezés felírható a konjunkcióinak megfelelő Datalog-szabályok segítségével. A szabályok kiértékelésének eredménye tulajdonképpen a konjunkciók eredményeinek egyesítése.

5.28. példa. Az 5.27. példában a fenti algoritmus egy egyszerű példáját láthattuk. Ha az előző példa feltételét negáljuk, akkor egy jóval bonyolultabb kifejezést kapunk:

$$\sigma_{\texttt{NOT (hossz}\geq\texttt{100 OR stúdióNév='Fox')}}(\texttt{Filmek})$$

Azaz, válasszuk ki azokat a filmeket, amelyek sem nem hosszúak, sem nem a Fox stúdióban készültek.

Ebben az esetben a NOT nem egy egyszerű összehasonlítás előtt áll, ezért a *DeMorgan-szabály* alkalmazásával be kell vinnünk a NOT operátort a kifejezés belsejébe. A következő kifejezést kapjuk:

$$\sigma_{\texttt{(NOT (hossz}\geq\texttt{100)) AND (NOT (stúdióNév='Fox'))}}(\texttt{Filmek})$$

A NOT operátort bevihetjük az összehasonlítások belsejébe:

$$\sigma_{\texttt{hossz}<\texttt{100 AND stúdióNév}\neq\texttt{'Fox'}}(\texttt{Filmek})$$

Ez a kifejezés Datalogban a következő szabállyal írható fel:

S(fc,é,h,m,s,p) $\leftarrow$ Filmek(fc,é,h,m,s,p) AND h < 100 AND s $\neq$ 'Fox'

□

5.29. példa. Vegyünk egy hasonló példát, ahol a kiválasztás feltételében AND operátorral összekapcsolt feltételek negáltja szerepel. Ezúttal a DeMorgan-szabály második formáját alkalmazzuk, amely szerint az AND negáltja a negációk OR-ja. Kezdjük a relációs algebrai kifejezéssel:

$$\sigma_{\texttt{NOT (hossz}\geq\texttt{100 AND stúdióNév='Fox')}}(\texttt{Filmek})$$

[2] Lásd: A. V. Aho, J. D. Ullman, *Foundations of Computer Science*, Computer Science Press, New York, 1992.

Azaz, válasszuk ki azokat a filmeket, amelyek nem a Fox stúdióban készültek és nem hosszú filmek.

A NOT operátort a DeMorgan-szabály segítségével bevisszük a kifejezés belsejébe:

$$\sigma_{(\texttt{NOT (hossz}\geq 100)) \texttt{ OR (NOT (stúdióNév}=\texttt{'Fox'))}}(\texttt{Filmek})$$

Ezután bevisszük a NOT operátort az összehasonlítások belsejébe:

$$\sigma_{\texttt{hossz}<100 \texttt{ OR stúdióNév}\neq\texttt{'Fox'}}(\texttt{Filmek})$$

Legvégül felírunk két Datalog-szabályt, egyet-egyet az OR két oldalán található feltételeknek megfelelően:

1. S(fc,é,h,m,s,p) ← Filmek(fc,é,h,m,s,p) AND l < 100
2. S(fc,é,h,m,s,p) ← Filmek(fc,é,h,m,s,p) AND s ≠ 'Fox'

□

5.4.4. Szorzat

Két reláció szorzata, $R \times S$ kifejezhető egyetlen Datalog-szabály segítségével. A szabálynak két részcélja van, egyik az R-nek, a másik az S-nek felel meg. A részcélok változói megfelelnek a relációk attribútumainak, de egymástól különbözők. A fejben szereplő IDB-predikátum argumentumában az összes olyan változó szerepel, amely valamely részcélban megjelenik, és a sorrendet tekintve, az R részcél változóit követik az S részcél változói.

5.30. példa. Tekintsük az 5.24. példa relációit. Az R és S egyaránt három attribútummal rendelkezik. Az $R \times S$ szorzatot a következő Datalog-szabállyal fejezhetjük ki:

P(a,b,c,x,y,z) ← R(a,b,c) AND S(x,y,z)

Az R részcél változóinak nevét az ábécé elejéről vettük, míg az S részcél változóinak nevét az ábécé végéről. Az R és S részcélok mind a hat változója megjelenik a szabály fejében. □

5.4.5. Összekapcsolás

Két reláció természetes összekapcsolásának kifejezése Datalogban nagyon hasonlít a két reláció szorzatának megfelelő szabályához. A különbség csak annyi, hogy amikor az $R \bowtie S$ természetes összekapcsolást akarjuk felírni, akkor az R és S relációk közös attribútumaihoz tartozó változóknak ugyanazt a nevet kell adnunk, és különböző változókat kell a többi helyen használni. Akár az attribútumok neveit is használhatjuk változónévként. A fej egy olyan IDB-predikátum kell legyen, amelyben minden egyes változó megjelenik, méghozzá egyszer.

5.31. példa. Vegyük az $R(A,B)$ és $S(B,C,D)$ relációkat a megadott sémákkal. Természetes összekapcsolásukat a következő szabállyal fejezhetjük ki:

```
TÖ(a,b,c,d) ← R(a,b) AND S(b,c,d)
```

Figyeljük meg, hogy a részcélokban használt változók egyértelműen megfelelnek az R és S relációk attribútumainak. □

A théta-összekapcsolás hasonló módon írható fel Datalogban. A 2.4.12. alfejezetben láthattuk, hogy a théta-összekapcsolás kifejezhető szorzás és egy ezt követő kiválasztás segítségével. Ha a kiválasztás feltétele egy konjunkció, azaz összehasonlítások `AND` szócskával történő összekapcsolása, akkor egyszerűen felírjuk a szorzásnak megfelelő Datalog-szabályt, majd ehhez hozzáadjuk az összehasonlításoknak megfelelő aritmetikai részcélokat.

5.32. példa. Tekintsük az $U(A,B,C)$ és $V(B,C,D)$ relációk théta-összekapcsolását:

$$U \bowtie_{A<D \texttt{ AND } U.B \neq V.B} V$$

A következő Datalog-szabállyal ugyanezt a műveletet végezzük el:

```
Ö(a,ub,uc,vb,vc,d)  ←  U(a,ub,uc) AND V(vb,vc,d) AND
                       a < d AND ub ≠ vb
```

Az U reláció B attribútumához rendelt változónak az *ub* nevet adtuk, és ugyanígy kapták nevüket a *vb*, *uc*, *vc* változók is. A két relációnak összesen hat attribútuma van, és a hozzájuk rendelt változók tetszőleges különböző nevet kaphattak volna. Az első két részcél az összekapcsolni kívánt relációknak, míg a két utolsó részcél a théta-összekapcsolás feltételében szereplő összehasonlításoknak felel meg. □

Ha a théta-összekapcsolás feltétele nem egy konjunkció, akkor a feltételt az 5.4.3. alfejezetben tárgyalt módon átalakítjuk diszjunktív normálformára. Ezután a feltétel minden egyes konjunkciójához felírjuk a megfelelő szabályt. Egy ilyen szabály a szorzathoz tartozó részcélokkal kezdődik, amelyet a konjunkcióban szereplő literálokhoz tartozó részcélok követnek. Minden szabály feje egyforma, és argumentumként annyi változót tartalmaz, ahány attribútuma a théta-összekapcsolásban részt vevő két relációnak összesen van.

5.33. példa. Módosítsunk egy kicsit az 5.32. példában megadott relációs algebrai kifejezésen. Cseréljük ki az `AND` operátort `OR` operátorra. Ily módon a feltétel rögtön diszjunktív normálformában van, hiszen nem szerepel benne negáció. Két konjunkciónk van, mindkettő egyetlen literálból áll. A kifejezés:

$$U \bowtie_{A<D \texttt{ OR } U.B \neq V.B} V$$

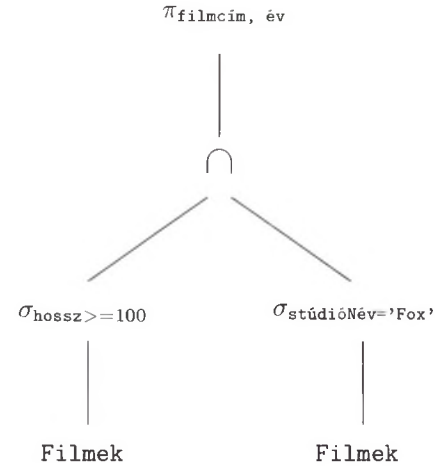
Ha a változókat ugyanúgy nevezzük, mint az 5.32. példában, akkor a következő két szabályt kapjuk:

1. Ö(a,ub,uc,vb,vc,d) ← U(a,ub,uc) AND V(vb,vc,d) AND a < d
2. Ö(a,ub,uc,vb,vc,d) ← U(a,ub,uc) AND V(vb,vc,d) AND ub ≠ vb

A szabályokban a két relációnak megfelelő részcél mellett az $A < D$ vagy az $U.B \neq V.B$ összehasonlításnak megfelelő részcél szerepel. □

5.4.6. Kifejezések megadása Datalogban

Datalog-szabályokkal nem csak elemi relációs algebrai operátorokat fejezhetünk ki, hanem alapjában véve bármilyen relációs algebrai kifejezést. Az ötlet mindössze annyi, hogy felrajzoljuk a relációs algebrai kifejezéshez tartozó kifejezésfát, és minden egyes belső csúcshoz megadjuk a megfelelő IDB-predikátumot. Az IDB-predikátumhoz tartozó szabályt vagy szabályokat úgy kapjuk, hogy a fa megfelelő csúcsában található operátorra alkalmazzuk. A kifejezésfa extenzionális operandusaira (amelyek tulajdonképpen az adatbázis relációi) a hozzájuk tartozó predikátumokkal hivatkozhatunk. A belső csúcsok operandusaira a megfelelő IDB-predikátumokkal hivatkozhatunk. Az algebrai kifejezés eredménye egy olyan reláció, amely a kifejezésfa gyökerében szereplő predikátumnak felel meg.



5.9. ábra. A kifejezésfa

5.34. példa. Vegyük a 2.17. példa relációs algebrai kifejezését.

$$\pi_{\texttt{filmcím,év}}\big(\sigma_{\texttt{hossz}\geq 100}(\texttt{Filmek}) \cap \sigma_{\texttt{stúdióNév}=\texttt{'Fox'}}(\texttt{Filmek})\big)$$

A megfelelő kifejezésfát a 2.18. ábrán láthattuk, de a könnyebb követhetőség érdekében az 5.9. ábrán is ugyanez látható. Négy IDB-predikátumot kell felírnunk a négy belső csúcsnak megfelelően. Mindegyik predikátumhoz egyetlen szabály fog tartozni, amint az az 5.10. ábrán látható.

A két alsó belső csúcs egyszerű kiválasztás a Filmek EDB-relációra alkalmazva. A hozzájuk tartozó IDB-predikátumok a W és az X, amelyeket az 5.10. ábra első két szabályában írtunk fel. Az első szabály például a Filmek reláció azon sorait adja meg, amelyekben a film hosszúsága legalább 100 perc.

```
1. W(fc,é,h,m,s,p) ← Filmek(fc,é,h,m,s,p) AND h ≥ 100
2. X(fc,é,h,m,s,p) ← Filmek(fc,é,h,m,s,p) AND s = 'Fox'
3. Y(fc,é,h,m,s,p) ← W(fc,é,h,m,s,p) AND X(fc,é,h,m,s,p)
4. Válasz(fc,é) ← Y(fc,é,h,m,s,p)
```

5.10. ábra. Összetett relációs algebrai kifejezésnek megfeleltetett Datalog-szabályok

A harmadik szabály a W és X metszetének megfelelő Y predikátumot adja meg az 5.4.1. alfejezetben tanult szabályforma használatával. Végezetül a negyedik szabály fogalmazza meg a kérdésre adott választ, az Y filmcím és év attribútumaira történő levetítése által. A vetítés leírására az 5.4.2. alfejezetben bemutatott módszert használtuk.

Vegyük észre, hogy Y-t egyetlen szabály írja le, ezért az 5.10. ábrán lévő 4. szabály Y részcéljának helyébe behelyettesíthetjük a szabály törzsét. Ezek után a W és az X részcélokat is helyettesíthetjük az 1., illetve 2. szabályok törzsével. Mivel a Filmek részcél mindkét törzsben szerepel, ezért az egyiket elhagyhatjuk. Eredményként így egyetlen szabályt kapunk.

```
Válasz(fc,é) ← Filmek(fc,é,h,m,s,p) AND h ≥ 100 AND s = 'Fox'
```

□

5.4.7. A relációs algebra és a Datalog összehasonlítása

Az 5.4.6. alfejezetnek megfelelően minden alap relációs algebrai kifejezés kifejezhető Datalog-lekérdezésként. A kiterjesztett relációs algebrának viszont vannak olyan műveletei, mint például az 5.2. alfejezetben leírt csoportosítás vagy összesítés, amelyeknek nincs megfelelő Datalog-változata. Hasonlóan a Datalog nem támogatja a multihalmaz-műveleteket sem, mint például az ismétlődések megszüntetését.

Másrészről igaz az, hogy tetszőleges Datalog-szabály kifejezhető relációs algebrában. Azaz, az alap relációs algebrában írhatunk olyan lekérdezést, amely ugyanazt a sorhalmazt adja, mint amelyet a szabály feje eredményez.

Amikor viszont Datalog-szabályok gyűjteményét vizsgáljuk, ez a helyzet megváltozik, hiszen Datalog-szabályokkal a rekurziót is tudjuk kezelni, amelyet a relációs algebrában viszont már nem tudunk megtenni. Ennek az oka, hogy az IDB-predikátumokat szabályok törzsében is használhatjuk, és így egy szabály fejéhez meghatározott sorokat tovább használhatjuk szabályok törzsében is. Ezáltal pedig több sort kaphatunk a fejrészben. Ezen a ponton nem tárgyaljuk, hogy milyen bonyodalmakat okozhat ez a megközelítés, főként azokban az

esetekben, mikor negált részcél is van. A következő példával szemléltetjük a rekurzív Datalogot.

5.35. példa. Tegyük fel, hogy adott egy `Él(X,Y)` reláció, ami irányított éleket jelent az X és Y csúcsok között. Megfogalmazhatjuk az él reláció tranzitív lezártját, azaz az `Út(X,Y)` relációt, ami azt fejezi ki, hogy X csúcsból Y csúcsba van egy legalább 1 hosszú út, azaz:

1. `Út(X,Y)` ← `Él(X,Y)`
2. `Út(X,Y)` ← `Él(X,Z) AND Út(Z,Y)`

Az első szabály azt jelenti, hogy minden él egyben út is. A második szabály pedig azt fejezi ki, hogyha van él az X csúcsból valamilyen Z csúcsba, illetve Z csúcsból van út Y csúcsba, akkor X-ből is van út Y-ba. Ha tekintjük a két szabályt, akkor 2 hosszú utakat kapunk. Ha vesszük az eddigi eljárásból nyert *Út* tényeket, és felhasználjuk azokat egy újabb alkalmazáshoz, akkor már 3 hosszú utakat kapunk. Ha ezekkel az *Út* tényekkel újra alkalmazzuk a szabályainkat, akkor 4 hosszú utakat nyerünk és így tovább. Végül megtaláljuk az összes utat, és a következő menetben már nem kapunk újabb tényeket. Ennél a pontnál befejezhetjük az eljárást. Ha nem kaptuk meg az $Út(a, b)$ tényt, akkor valójában nincs is út a gráfban az a és b csúcsok között. □

5.4.8. Feladatok

5.4.1. feladat. Adottak az $R(a, b, c)$, $S(a, b, c)$ és $T(a, b, c)$ relációk. Írjuk fel a következő relációs algebrai kifejezéseket egy vagy több Datalog-szabály segítségével:

a) $R \cup S$.

b) $R \cap S$.

c) $R - S$.

d) $(R \cup S) - T$.

! *e)* $(R - S) \cap (R - T)$.

f) $\pi_{a,b}(R)$.

! *g)* $\pi_{a,b}(R) \cap \rho_{U(a,b)}\big(\pi_{b,c}(S)\big)$.

5.4.2. feladat. Adott az $R(x, y, z)$ reláció. Írjuk fel egy vagy több Datalog-szabály egyesítésével a $\sigma_C(R)$ kifejezést, ahol C a következő alakú:

a) $x = y$.

b) $x < y$ `AND` $y < z$.

c) $x < y$ OR $y < z$.

d) NOT $(x < y$ OR $x > y)$.

! *e)* NOT $\big((x < y$ OR $x > y)$ AND $y < z\big)$.

! *f)* NOT $\big((x < y$ OR $x < z)$ AND $y < z\big)$.

5.4.3. feladat. Adottak az $R(a, b, c)$, $S(b, c, d)$ és $T(d, e)$ relációk. Írjuk fel a következő természetes összekapcsolásokat egyetlen Datalog-szabály segítségével:

a) $R \bowtie S$.

b) $S \bowtie T$.

c) $(R \bowtie S) \bowtie T$. (*Megjegyzés:* Mivel a természetes összekapcsolás asszociatív és kommutatív, ezért mindegy, hogy milyen sorrendben kapcsoljuk össze a három relációt.)

5.4.4. feladat. Adottak az $R(x, y, z)$ és $S(x, y, z)$ relációk. Írjuk fel az $R \bowtie_C S$ théta-összekapcsolásnak megfelelő Datalog-szabályt (illetve szabályokat), ha a C feltétel az 5.4.2. példában felsorolt feltételek egyike. A feltételekben található aritmetikai összehasonlításokat tekintsük úgy, mint amelyik sorrendben az R egy attribútumát hasonlítja össze az S egy attribútumával. Az $x < y$ alakú összehasonlítás értelmezése tehát $R.x < S.y$.

! **5.4.5. feladat.** Datalog-szabályok is felírhatók velük ekvivalens relációs algebrai kifejezésekkel. Próbálkozzunk meg néhány egyszerű esettel annak ellenére, hogy nem mutattunk be erre szolgáló általános módszert. A következő Datalog-szabályokat írjuk át olyan relációs algebrai kifejezésekké, amelyek eredménye megegyezik a fejnek megfelelő relációval.

a) `P(x,y) ← Q(x,z) AND R(z,y)`

b) `P(x,y) ← Q(x,z) AND Q(z,y)`

c) `P(x,y) ← Q(x,z) AND R(z,y) AND x < y`

5.5. Összefoglalás

✦ *Relációk multihalmazokként:* A forgalomban lévő adatbázisrendszerekben a relációkat multihalmaznak tekintik, amelyben ugyanazon sor többször is előfordulhat. A relációs algebra halmazokon értelmezett műveletei kiterjeszthetőek multihalmazokra is, de van néhány algebrai azonosság, amely sérülni fog.

- *A relációs algebra kiterjesztései:* Az SQL lehetőségeinek megfelelően néhány műveletet be kell vezetnünk a relációs algebrába. A relációk rendezése például egy ilyen művelet. Ugyanilyen példa a kiterjesztett vetítés, amely során az oszlopokon történő számítások is elvégezhetők. A csoportosítás, az összesítés és a külső összekapcsolás is hasonlóan szükséges műveletek.

- *Csoportosítás és összesítés:* Az összesítés összegzi egy reláció egyik oszlopát. A tipikus összesítési műveletek az összegzés, az átlag, a számlálás, a minimum és a maximum. A csoportosítás lehetővé teszi számunkra, hogy a sorokat bizonyos attribútumuknak vagy attribútumaiknak az értéke vagy az értékei alapján felosszuk. A csoportokra pedig összesítés(eke)t végezhetünk.

- *Külső összekapcsolás:* Két reláció külső összekapcsolása az érintett relációk összekapcsolásával kezdődik, majd bármelyik relációból származó lógó soroknak (azaz olyan sorok, melyeket nem sikerült más sorral összekapcsolni) a másik reláció értékeinek null értékekkel kitöltött változatának az eredményrelációhoz történő hozzáadásával folytatjuk.

- *Datalog:* Ezen logikai formalizmus lehetővé teszi a relációs modellbeli lekérdezések leírását. Egy Datalog-szabály részcélokból tevődik össze, azaz egy fejpredikátum vagy reláció a törzs feltételei által van definiálva.

- *Atomok:* Mind a fej, mind a részcélok atomok. Egy atom tartalmazhat egy (esetleg negált) predikátumot valahány argumentummal. A predikátumok reprezentálhatnak relációkat vagy aritmetikai összehasonlításokat (például $>$).

- *IDB- és EDB-predikátumok:* A tárolt relációkra hivatkozó predikátumokat EDB- (extenzionális adatbázis) predikátumoknak vagy relációknak nevezzük. A többi, szabályokkal meghatározott predikátumokat pedig IDB- (intenzionális adatbázis) predikátumoknak nevezzük. Az EDB-predikátumok nem szerepelhetnek a szabály fejében.

- *Biztonságos szabályok:* Általában a Datalog-szabályokról feltesszük, hogy biztonságosak, azaz hogy a szabályokban szereplő összes változó előfordul a törzs valamely nem negált relációs részcéljában. A biztonságosság biztosítja, hogy ha az EDB-relációk végesek, akkor az IDB-relációk is azok lesznek.

- *Relációs algebra és Datalog:* Minden olyan lekérdezés, amely felírható relációs algebrában, kifejezhető Datalogban is. Ha a szabályok biztonságosak és nem rekurzívak, akkor ugyanazt a lekérdezéseket tartalmazó halmazt kapjuk, mint a relációs algebránál.

5.6. Irodalomjegyzék

A relációs algebra [2] leírása alapján történt, ahogy azt már a 2. fejezetben említettük. A kiterjesztett γ operátor [5]-ből származik.

Codd a relációs modellről szóló korai publikációinak egyikében [3] még két másik elsőrendű logikát is bevezetett, amelyeket *sor alapú kalkulus*nak, illetve *tartomány alapú kalkulus*nak nevezett el. Ezek kifejezőereje megegyezik a relációs algebráéval, ahogy azt [3]-ban bizonyította is.

A Datalog leginkább olyan logikai szabályokra hasonlít, amelyeket a Prolog programozási nyelv ihletett. A logika használata lekérdező nyelvként leginkább [4]-ből származik, de az elméletet [1] vezette be az adatbázisrendszerekbe.

A relációs algebra és a relációs kalkulus bővebb leírása megtalálható [6]-ban, illetve [7]-ben.

[1] F. Bancilhon, R. Ramakrishnan, „An amateur's introduction to recursive query-processing strategies," *ACM SIGMOD Intl. Conf. on Management of Data*, pp. 16–52, 1986.

[2] E. F. Codd, „A relational model for large shared data banks," *Comm. ACM* **13**:6, pp. 377–387, 1970.

[3] E. F. Codd, „Relational completeness of database sublanguages," in *Database Systems* (R. Rustin, ed.), Prentice Hall, Englewood Cliffs, NJ, 1972.

[4] H. Gallaire, J. Minker, *Logic and Databases*, Plenum Press, New York, 1978.

[5] A. Gupta, V. Harinarayan, D. Quass, „Generalized projections: a powerful approach to aggregation," 21st Intl. Conf. on Very Large Database Systems, pp. 358–369, 1995.

[6] M. Liu, „Deductive database languages: problems and solutions," *Computing Surveys* **31**:1 (March, 1999), pp. 27–62.

[7] J. D. Ullman, *Principles of Database and Knowledge-Base Systems, Volumes I and II*, Computer Science Press, New York, 1988, 1989.

6. fejezet

Az SQL adatbázisnyelv

A legszélesebb körben használt relációs adatbázis-kezelő rendszerek egy SQL-nek nevezett nyelv segítségével kérdezik le és módosítják az adatbázist. Az SQL a „Structured Query Language" (Strukturált Lekérdező Nyelv) rövidítése. Az SQL lekérdező lehetőségei nagyon hasonlóak a relációs algebrának az 5.2. alfejezetben bemutatott kiterjesztéséhez. Ezenfelül az SQL-nek vannak az adatbázist módosító utasításai (például sorok beszúrása és sorok törlése), valamint az adatbázisséma megadását lehetővé tevő utasításai. Így az SQL adatmanipulációs és adatleíró nyelvként is szolgál. Az SQL-szabványban további adatbázis-kezelő utasítások is szerepelnek, ezekkel a 7. és 9. fejezetben foglalkozunk.

Az SQL-nek számos különböző verziója van. Először is három fő szabványa ismert. Az ANSI (American National Standards Institute – Amerikai Nemzeti Szabvány Intézet) SQL és egy 1992-ben elfogadott módosított szabvány, az SQL-92 vagy SQL2. A jelenleg SQL-99-nek (korábban SQL3-nak) nevezett szabványa kiterjeszti az SQL2-t az objektumrelációs környezetre, és számos új lehetőséggel bővíti. Az SQL-99-nek is vannak már kiterjesztései, ezeket együtt SQL:2003-nak nevezik. Ezenkívül léteznek az SQL-nek a nagy adatbázis-kezelő rendszereket forgalmazó cégek által készített verziói is. Ezek mindegyike kielégíti az eredeti ANSI-szabványt. Nagyrészt összhangban vannak az SQL2-vel is, mindegyikük az SQL2 egy változatának és kiterjesztésének tekinthető, megvalósítják az SQL-99 és az SQL:2003 szabványokban rögzített lehetőségek egy részét is.

Ez a fejezet az SQL alapjaival, a lekérdező nyelvvel és az adatbázis-módosító utasításokkal foglalkozik. Az adatbázisrendszerek alapvető programozási egységeként bevezetjük a „tranzakció" fogalmát is. Az áttekintésünk egyszerűsített, csak érzékeltetjük, hogy az adatbázis-műveletek hogyan hathatnak egymásra és milyen buktatóik vannak.

A következő fejezetben a megszorításokkal és a triggerekkel foglalkozunk, ezek az adatbázis tartalma feletti felhasználói ellenőrzés lehetőségeit bővítik. A 8. fejezetben olyan lehetőségeket mutatunk be, melyekkel az SQL-lekérdezéseinket tudjuk hatékonyabbá tenni, elsősorban az indexek és hasonló struktúrák deklarálásával. A 9. fejezet az adatbázisok programokból való hasz-

nálatával foglalkozik. E programok olyan nagy rendszerek részei, mint például a weben közösen használt nagy (szerver) kiszolgáló rendszerek. Látni fogjuk, hogy az SQL-lekérdezéseket és más SQL-műveleteket túlnyomó részben sosem önmagukban használjuk, hanem a hagyományos programozási nyelven írt programokba beágyazottan, így ezeknek együtt kell működniük.

Végül a 10. fejezet egy sor fejlett adatbázis-programozási koncepciót magyaráz el. Ilyenek a rekurzív SQL, az SQL hozzáférés-ellenőrzési és biztonsági lehetőségei, az objektumrelációs SQL, és az adatok adatkocka modellje.

Ebben és a következő fejezetekben a célunk az SQL megismertetése az olvasóval, mégpedig bevezetés, mintsem kézikönyv szintjén. Így csak a leggyakrabban használt részekre koncentrálunk, és olyan példákat mutatunk, melyek nemcsak szabványosak, hanem a kereskedelmi adatbázisrendszerekkel is összhangban vannak. Az irodalomjegyzékbeli hivatkozások olyan publikációkat tartalmaznak, amelyekben megtalálható a nyelv részletesebb leírása és a különböző verziók.

6.1. Egyszerű lekérdezések az SQL-ben

Az SQL legegyszerűbb lekérdezései azon sorokra vonatkoznak, melyek egy bizonyos relációban eleget tesznek egy adott feltételnek. Egy ilyen lekérdezés a relációs algebra kiválasztási műveletének felel meg. Ez az egyszerű lekérdezés, mint ahogy a legtöbb SQL-lekérdezés, az SQL három alapvető kulcsszavát használja, mégpedig a `SELECT`, `FROM` és `WHERE` kulcsszavakat.

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

6.1. ábra. Minta adatbázisséma, ismétlés

6.1. példa. A következő példákban a film adatbázis 2.2.8. alfejezetben leírt adatbázissémáját fogjuk alkalmazni. Ezt az adatbázissémát a 6.1. ábrán ismét bemutatjuk.

Első lekérdezésként a

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

relációból kérdezzük le az összes olyan filmet, melyet a Disney stúdió 1990-ben készített. A megfelelő SQL-utasítás:

```
SELECT *
FROM Filmek
WHERE stúdióNév = 'Disney' AND év = 1990;
```

Hogyan is használjuk az SQL-t?

Ebben a fejezetben feltételezzük, hogy van egy *általános lekérdező programunk*, melynek begépelhetjük az SQL-lekérdezéseket és más SQL-utasításokat, és a lekérdező program ezeket végrehajtja. A gyakorlatban ilyen általános lekérdező programot ritkán használnak. Sokkal inkább hagyományos programozási nyelveken – mint a C vagy a JAVA *(befogadó nyelvek)* – írott programokkal dolgozunk. Ezen programokban az adott befogadó nyelv speciális könyvtárait használva az adatbázisra vonatkozó SQL-utasításokat is kiadhatnak. Az adatokat a befogadó nyelv változóiból az SQL-utasításokba, ezen utasítások végrehajtásának eredményeit pedig az adatbázisból a befogadó nyelv változóiba mozgatják át. Ezt a 9. fejezetben sokkal részletesebben tárgyaljuk.

Ez a lekérdezés az SQL-lekérdezések jellegzetes, select-from-where alakját mutatja be.

- A `FROM` záradék azon relációt vagy relációkat adja meg, melyekre a lekérdezés vonatkozik. A példánkban a lekérdezés a `Filmek` relációra vonatkozik.
- A `WHERE` záradék egy feltétel, nagyon hasonlít a relációs algebrában használt kiválasztási feltételhez. A lekérdezés válaszába azon sorok kerülnek, melyek kielégítik az adott feltételt. A példában a feltétel az, hogy a `stúdióNév` attribútum értéke `'Disney'` és az `év` attribútum értéke 1990 legyen. Azon sorok, melyek mindkét feltételt kielégítik, megjelennek a lekérdezés eredményében, a többi nem.
- A `SELECT` záradék megadja a feltételeknek megfelelő sorok azon attribútumait, melyeket a lekérdezésre adott válasz tartalmazni fog. A példabeli `*` azt jelzi, hogy a teljes sort tartalmazni fogja a válasz. A lekérdezés eredménye az a reláció, amely tartalmazza az összes sort, melyeket ez az eljárás előállít.

A lekérdezés feldolgozásának egyik módja az, hogy a `FROM` záradékbeli relációnak az összes sorát egymás után megvizsgáljuk. A `WHERE` záradék feltételét alkalmazzuk a sorra. Pontosabban, a `WHERE` záradékban szereplő attribútumokba behelyettesítjük a sor megfelelő komponenseinek értékét. A feltételt kiértékeljük, és ha igaz, akkor a `SELECT`-ben szereplő attribútumok által alkotott sort az eredményhez hozzávesszük. Így a lekérdezés eredményei azon `Filmek`-beli sorok lesznek, melyeknek megfelelő filmeket a Disney gyártott 1990-ben, például az eredményben lesz a *Micsoda nő!*

Amikor az SQL-t feldolgozó processzor a következő sort vizsgálja meg:

filmcím	*év*	*hossz*	*műfaj*	*stúdióNév*	*producerAzon*
`Micsoda nő!`	`1990`	`119`	`vígjáték`	`Disney`	`999`

Hogyan olvassunk és írjunk lekérdezéseket?

A select-from-where lekérdezések értelmezésének általában legegyszerűbb módja, ha először a FROM záradékot keressük meg, innen megtudjuk, hogy a lekérdezésben mely relációkra hivatkozunk. Aztán a WHERE záradékot keressük, belőle megtudhatjuk, hogy a lekérdezés szempontjából mely sorok számítanak fontosnak. Végül a SELECT záradékból látjuk, milyen lesz az eredmény. Ugyanez a sorrend – from, majd where, aztán select – gyakran hasznosnak bizonyulhat akkor is, amikor a lekérdezést mi magunk írjuk.

(ahol 999 a producer elképzelt azonosítója), a WHERE záradék feltételében a stúdióNév attribútum értéke 'Disney', míg az év attribútum értéke 1990 lesz, mert ezek lesznek a megfelelő attribútumok értékei a szóban forgó sor esetében. Így a WHERE feltétel alakja a következő lesz:

```
WHERE 'Disney' = 'Disney' AND 1990 = 1990
```

Mivel ez a feltétel nyilván igaz, a *Micsoda nő!*-re vonatkozó sor megfelel a WHERE záradék feltételeinek, így a sor az eredményhez fog tartozni. □

6.1.1. Vetítés az SQL-ben

Ha azt szeretnénk, hogy a kiválasztott sorok bizonyos komponenseit kizárjuk az eredményből, akkor levetítjük az SQL-lekérdezéssel előállított relációt néhány attribútumára. A SELECT záradékban a * helyett felsorolhatjuk a FROM záradékban adott reláció bármely attribútumát. Az eredményt levetítjük a felsorolt attribútumokra.[1]

6.2. példa. Tételezzük fel, hogy a 6.1. példa lekérdezését szeretnénk úgy módosítani, hogy csak a film címét és hosszát adja vissza. A megfelelő utasítás:

```
SELECT filmcím, hossz
FROM Filmek
WHERE stúdióNév = 'Disney' AND év = 1990;
```

Az eredmény egy kétoszlopos tábla, a filmcím és hossz oszlopokkal. A tábla sorai a filmek címeiből és hosszából álló azon párosok, amelyekben a filmet a Disney készítette 1990-ben. Például a relációséma és egy sora így néz ki:

[1] Így a SELECT kulcsszó főleg a relációs algebra vetítés műveletének, míg az algebra kiválasztás művelete az SQL-lekérdezés WHERE záradékának felel meg.

filmcím	*hossz*
Micsoda nő!	119
...	...

□

Néha olyan relációt szeretnénk készíteni, melyben az oszlopnevek különböznek a FROM záradékban megadott reláció attribútumneveitől. Ezért az attribútumnév után az AS kulcsszót írhatjuk, utána pedig egy *másodnév* következik, azaz egy új név, mely az eredményrelációban az eredeti oszlopnév helyett fog szerepelni. Az AS nem kötelező. A másodnév, minden elválasztójel (pont, vessző) nélkül, rögtön azon attribútum után következik, melynek neveként fog szerepelni.

6.3. példa. A 6.2. példát módosíthatjuk úgy, hogy olyan relációt eredményezzen, melynek attribútumai név és időtartam, az eredeti filmcím és hossz helyett:

```
SELECT filmcím AS név, hossz AS időtartam
FROM Filmek
WHERE stúdióNév = 'Disney' AND év = 1990;
```

Az eredmény megegyezik a 6.2. példa eredményével, de az oszlopnevek név és időtartam. Például az eredményreláció kezdődhet így:

név	*időtartam*
Micsoda nő!	119
...	...

□

További lehetőség a SELECT záradékban a kifejezés használata az attribútum helyett, valamint, hogy az 5.2.5. alfejezetben tárgyalt kiterjesztett vetítés listájához hasonlót adjunk meg a SELECT listában is. A 6.4. alfejezetben láthatjuk, hogy a SELECT lista ugyanúgy tartalmazhat összesítéseket, mint ahogy az 5.2.4. alfejezetben bemutatott γ művelet is.

6.4. példa. Tételezzük fel, hogy ugyanazt az eredményt szeretnénk kapni, mint a 6.3. példában, de a hossz órákban legyen kifejezve. A SELECT záradékot kicserélhetjük a következőre:

```
SELECT filmcím AS név, hossz*0.016667 AS hosszÓrákban
```

Így ugyanazokat a filmeket adja, de az időtartamot órákban kapjuk meg, és a második oszlop neve hosszÓrákban lesz:

Kisbetű/nagybetű

Az SQL nem különbözteti meg a kis- és nagybetűket. Például, ugyan mi úgy döntöttünk, hogy az olyan kulcsszavakat mint a FROM nagybetűkkel írjuk, azonban ugyanúgy helyes a From vagy from, vagy akár FrOm is. Az attribútumnevek, a relációnevek, a másodnevek stb. mind függetlenek attól, hogy kis- vagy nagybetűvel írtuk őket. Az SQL csak az idézőjelek közé tett kifejezések esetén tesz különbséget kis- és nagybetűk között. Így, a 'FROM' és a 'from' különböző karaktersorok, melyek egyike sem egyezik meg a FROM kulcsszóval.

név	*hosszÓrákban*
Micsoda nő!	1.98334
...	...

□

6.5. példa. A SELECT záradékban a tételek között konstansokat is megadhatunk. Ez értelmetlennek tűnik, de alkalmazható például arra, hogy fontos szavakat illesszünk az eredménybe. A következő lekérdezés:

```
SELECT filmcím, hossz*0.16667 AS hossz, 'óra' AS Órákban
FROM Filmek
WHERE stúdióNév = 'Disney' AND év = 1990;
```

olyan sorokat generál, mint:

filmcím	*hossz*	*Órákban*
Micsoda nő!	1,98334	óra
...	...	...

A harmadik oszlop neve Órákban, mely a második oszlop címével összetartozik. A válasz minden sorában ott van az 'óra' konstans, mely így olyan benyomást kelt, mintha a második oszlopbeli érték egységeként szerepelne. □

6.1.2. Kiválasztás az SQL-ben

Az SQL WHERE záradéka kiterjeszti a relációs algebra kiválasztás operátorát. A WHERE-t ahhoz hasonló feltételkifejezések követhetik, mint amelyeket a közismert C vagy Java nyelvek is használnak.

A hat alapvető összehasonlítási operátor: =, <>, <, >, <=, >= segítségével értékeket hasonlíthatunk össze, s ezekkel építhetjük fel a feltételkifejezéseket. Ezeknek az operátoroknak jelentése megegyezik a C-ben használtakkal, de az

SQL „<>" (jelentése: „nem egyenlő") a C-ben használt „!=" -vel azonos, az „=" (jelentése: egyenlő) a C-ben használt „==" -vel egyezik meg.

Az összehasonlítható értékek között lehetnek konstansok és a FROM utáni relációk attribútumai. Az értékekre alkalmazhatjuk a szokásos matematikai alapműveleteket is, mint például a +, * stb., mielőtt összehasonlítjuk őket. Például az (év − 1930) * (év − 1930) < 100 kifejezés igaz az 1930-tól legfeljebb 9 év távolságra levő évekre. Alkalmazhatjuk az összekapcsolás műveletet is: (||) karakterláncokra; például 'labda' || 'rúgás' értéke 'labdarúgás'.

Egy példát láthattunk az összehasonlításra a 6.1. példában:

```
stúdióNév = 'Disney'
```

A Filmek reláció stúdióNév attribútumának értékét összehasonlítjuk a 'Disney' konstanssal. Ennek a konstansnak karakterlánc értéke van, melyet az SQL-ben egy idézőjellel jelölünk. Megengedettek numerikus konstansok is, mint az egész számok vagy a valós számok; az SQL a szokásos jelöléseket alkalmazza a valós számokra: -12.34 vagy 1.23E45.

Az összehasonlítás eredménye egy igazságérték: TRUE (igaz) vagy FALSE(hamis).[2] A logikai értékeket kombinálhatjuk az AND, OR és NOT (az „és", „vagy" és „nem") logikai műveletek segítségével, amelyeket a szokásos jelentésükkel használunk. A 6.1. példában láthattuk, hogyan lehet két feltételt az AND művelettel összekapcsolni. Ebben a példában a WHERE feltétel igazságértéke akkor és csak akkor lesz igaz, ha mindkét összehasonlítás eredménye igaz, azaz a stúdió neve 'Disney' és az év 1990. A következőkben még néhány összetett WHERE feltételű példát mutatunk be.

6.6. példa. Tekintsük a következő lekérdezést:

```
SELECT filmcím
FROM Filmek
WHERE (év > 1970 OR hossz < 90) AND stúdióNév = 'MGM';
```

Ez a lekérdezés azon filmcímeket adja meg, amelyeket az MGM Stúdió készített, és amelyek vagy 1970 után készültek vagy 90 percnél rövidebbek. Figyeljük meg, hogy az összehasonlításokat zárójelek segítségével csoportosíthatjuk. A zárójelekre azért van szükség, mert az SQL-ben a logikai műveletek megelőzési sorrendje ugyanolyan, mint a legtöbb programozási nyelvben: az AND megelőzi az OR műveletet, a NOT pedig megelőzi mindkettőjüket. □

6.1.3. Karakterláncok összehasonlítása

Két karakterlánc egyenlő, ha a karaktereknek ugyanabból az egymás után következő sorozatából állnak. A 2.3.2. alfejezetből más ismerjük, hogy a karakterláncokat fix hosszú karakterláncként (CHAR), vagy változó hosszúságú karakterláncokban (VARCHAR) tároljuk. Ha különbözően deklarált karakterláncokat

[2] Valójában többféle igazságérték is létezik, lásd a 6.1.7. alfejezetet.

SQL-lekérdezések és a relációs algebra

Az eddig látott egyszerű SQL-lekérdezések mind az alábbi alakúak:

SELECT L
FROM R
WHERE C

ahol L kifejezések listája, R reláció, C pedig feltétel. Az ilyen lekérdezés jelentése megegyezik a relációs algebrai

$$\pi_L(\sigma_C(R))$$

kifejezés jelentésével. Ez az oka annak, hogy az értelmezést a FROM záradék relációjával kezdtük, melynek minden sorára ellenőriztük, hogy kielégíti-e a WHERE záradékban megadott feltételt, majd vetítettünk a SELECT záradékban megadott attribútumokra és/vagy kifejezésekre.

hasonlítunk össze, akkor csak az aktuális karakterláncok hasonlítódnak, azaz az SQL figyelmen kívül hagyja a „kitöltő" („pad") karaktereket, amelyeket azért használ, hogy az adatbázisban a karakterlánc elérje a deklarációjában megkívánt hosszát.

Amikor két karakterláncot összehasonlítunk egy aritmetikai összehasonlító operátorral, mint a $<$ vagy a $>=$, azt szeretnénk tudni, hogy az egyik megelőzi-e a másikat lexikografikus rendezésben (azaz ábécérend szerint). Ha $a_1a_2\cdots a_n$ és $b_1b_2\cdots b_m$ két karakterlánc, akkor az első „kisebb mint" a második, ha $a_1 < b_1$, vagy $a_1 = b_1$ és $a_2 < b_2$, vagy $a_1 = b_1$, $a_2 = b_2$ és $a_3 < b_3$ stb. $a_1a_2\cdots a_n < b_1b_2\cdots b_m$ akkor is, ha $n < m$ és $a_1a_2\cdots a_n = b_1b_2\cdots b_n$, azaz az első karakterlánc egy valódi prefixe, eleje a másodiknak. Például `'labda'<'lap'`, mivel az első két karakter azonos a két karakterláncban, a harmadik karaktere az elsőnek pedig megelőzi a második karakterlánc harmadik karakterét. `'kar'<'kard'`, mert az első a második valódi prefixe.

6.1.4. Mintával való összehasonlítás SQL-ben

Az SQL lehetővé teszi, hogy mintákkal is összehasonlíthassunk karakterláncokat. Az ilyen jellegű összehasonlítás formája:

```
s LIKE p
```

ahol s egy karakterlánc, p pedig egy *minta*. A minta egy olyan karakterlánc, melyben használhatjuk a speciális % és _ karaktereket. A p többi karaktere csak önmagának felel meg az s-ben. A % jel a p-ben megfelel az s-ben bármilyen karakterek 0 vagy nagyobb hosszúságú sorozatának, míg az _ jel a p-ben

Bitláncok megadása

Egy bitláncot egy `B` karakter és két idézőjel közötti 0 és 1 karakterek sorozata jelképez. Így a `B'011'` egy hárombites láncot jelképez, melyek közül az első 0, a másik kettő 1. Hexadecimális jelölést is alkalmazhatunk, melyben egy `X`-et követ egy idézőjelek közötti hexadecimális szám (a számjegyek 0-tól 9-ig és a-tól f-ig tartanak, melyek a 10-től 15-ig terjedő „számjegyeket" jelképezik). Például `X'7ff'` egy tizenkét bites láncot jelképez, az első bit 0, utána pedig tizenegy 1-es következik. Minden hexadecimális számjegy négy bitet jelképez, és a számkezdő 0-kat nem hagyhatjuk el.

megfelel az s-ben egy bármilyen karakternek. Ez a feltétel akkor és csak akkor igaz, ha az s karakterlánc megfelel a p mintának. Hasonlóképpen s `NOT LIKE` p akkor és csak akkor igaz, ha az s karakterlánc nem felel meg a p mintának.

6.7. példa. Egy olyan filmet keresünk, mely úgy kezdődik, hogy 'Halálos' és tudjuk, hogy a címben lévő második szó 7 karakterből áll. Mi lehet a film címe? Az ilyen típusú címeket a következő lekérdezéssel tudhatjuk meg:

```
SELECT filmcím
FROM Filmek
WHERE filmcím LIKE 'Halálos _______';
```

Ez a lekérdezés olyan filmcímeket keres, melyek 15 karakter hosszúak, az első 7 karakter: `'Halálos'`, a nyolcadik üres, az utolsó 7 karakter pedig bármi lehet. A lekérdezés eredménye a feltételnek megfelelő filmcímek, mint például a *Halálos fegyver* vagy a *Halálos játszma*. □

6.8. példa. Azokat a filmeket keressük, melyeknek a címében megtalálható az *'s* karaktersor. A megfelelő lekérdezés:

```
SELECT filmcím
FROM Filmek
WHERE filmcím LIKE '%''s%';
```

Hogy megértsük a példát, először is vegyük észre, hogy mivel az idézőjel a karaktersorok határoló karaktere az SQL-ben, nem képviselheti önmagát. Az SQL azt a szabályt alkalmazza, hogy két egymás utáni idézőjel egy karaktersorban egy idézőjelet jelképez és nem zárja le a sort. Így az `''s` karaktersor egy szimpla idézőjelből és azt követő `s`-ből álló karaktersorozatra illeszkedik.

A minta két végén a % karakterek bármilyen karaktersornak megfelelnek, így a válasz olyan filmeket fog tartalmazni, mint például a *Wayne's World* (Wayne világa). □

Escape karakterek a LIKE típusú kifejezésekben

Mi történik, ha az a karaktersor típus, melyre a LIKE kifejezéssel szeretnénk keresni, tartalmazza a % vagy _ karaktereket? Szükségünk van egy escape karakterre, mely semlegesíti a % vagy _ karakterek speciális jellegét. Ez az escape karakter nincs rögzítve, mint a UNIX esetében a \, hanem az SQL-utasításban meg lehet adni, mi legyen az, mégpedig úgy, hogy a minta után az ESCAPE kulcsszót írjuk, majd idézőjelek közé a kiválasztott karaktert. A % és _ karakterek, ha egy escape karakter áll előttük, csak önmaguknak felelnek meg a karaktersorban, és nem egy bármilyen karakterekből álló karaktersorozatnak, illetve egy bármilyen karakternek. Például az

```
s LIKE 'x%%x%' ESCAPE 'x'
```

kifejezésben x az escape karakter az x%%x% mintában. Az x% karaktersor egyetlen %-nak fog megfelelni. Így a minta minden olyan karaktersorra illeszkedik, mely a % karakterrel kezdődik és fejeződik be. Jegyezzük meg, hogy csak a középső % bír „bármilyen karaktersorozat" jelentéssel.

6.1.5. Dátumok és időpontok

Az SQL különböző megvalósításai általában speciális típusként kezelik a dátum- és időtípusokat. Ezek az adatok aztán sokféleképpen tárolhatók, mint például 05/14/1948 vagy 14 May 1948. Ebben a részben csak az SQL-szabványt adjuk meg, amely aprólékosan leírja ezeket a formátumokat.

Egy *dátum* típusú konstanst a DATE kulcsszóval, utána pedig egy idézőjelek közötti speciális karaktersorral írhatunk le. Például, a DATE '1948-05-14' megfelel a leírásnak. Az első négy karakter az év számjegyeit jelképezi. Utána következik egy gondolatjel, majd két számjegy, amely a hónapot jelképezi. Figyeljük meg, hogy egy egyszámjegyű hónap ki van egészítve elöl egy 0 karakterrel. Végül következik egy újabb gondolatjel és két számjegy, amely a napot jelképezi. Ugyanúgy, mint a hónapok esetében, ha szükséges, a napot is kiegészítjük egy kezdő 0-val, hogy kétszámjegyű számot kapjunk.

Egy *idő* konstans értéket hasonlóan a TIME kulcsszó és egy idézőjelek közötti karakterlánc segítségével lehet ábrázolni. A karaktersorban az órának két számjegy felel meg a 24 órás rendszerben. Ezután kettőspont következik, két számjegy a perceknek, újabb kettőspont, majd két számjegy a másodperceknek. Ha a másodperc törtrészeit is szeretnénk használni, következhet egy pont és annyi számjegy, amennyire szükség van. Így a TIME '15:00:02.5' azt az időt jelképezi, amikor már az összes diák elhagyta az osztálytermet egy olyan óra után, mely délután 3-kor ért véget, tehát két és fél másodperccel három után.

Az időt a greenwich-i időhöz (GMT: Greenwich Mean Time) képesti + vagy – eltéréssel is megadhatjuk. Például a `TIME '12:00:00-8:00'` (az USA) nyugati parti zónaidőben delet jelent, mely 8 órával kevesebb a greenwich-i időnél.

A dátum és az idő együtt jelenik meg a `TIMESTAMP` típusú értékben. A `TIMESTAMP` értékben a dátumot egy üres karakter, majd az időpont követi. Így 1948. május 14-én déli 12 órát a `TIMESTAMP '1948-05-14 12:00:00'` érték reprezentálja.

A dátum- és időértékeket ugyanúgy összehasonlíthatjuk, mint a karakterlánc- vagy numerikus értékeket. A < jel dátumok esetén azt jelenti, hogy az első korábbi dátum mint az utóbbi, időértékek esetében pedig azt jelenti, hogy az első korábbi időpont (egy napon belül), mint a második.

6.1.6. A nullérték és műveletek nullértékekkel

Az SQL lehetővé teszi, hogy az attribútum értéke egy speciális `NULL` legyen, amit *nullértéknek* nevezünk. A nullértékek értelmezésére több lehetőségünk is van. Lássuk ezek közül a leggyakoribbakat:

1. *Ismeretlen érték:* „Tudom, hogy valamilyen értéknek ott kell lenni, de nem tudom, melyik ez az érték." Ilyen például egy nem ismert születésnap.

2. *Alkalmazhatatlan érték:* „Nincs olyan érték, aminek itt értelme lenne." Például, ha a `FilmSzínész` relációnak lenne egy `hitves` attribútuma is, akkor az egyedülálló színészeknél ennek az attribútumnak nullértéke lenne, hiszen nem tudhatjuk valaki hitvesének a nevét, ha egyszer nincs is hitvese.

3. *Visszatartott érték:* „Nem vagyunk feljogosítva rá, hogy ismerjük a megfelelő értéket." Például a telefonszám attribútumhoz tartozó komponenshez `NULL` értéket írunk egy titkos `telefonszám` esetén.

Az 5.2.7. alfejezetben láttuk, hogy a külső összekapcsolás során is kerülhetnek egyes sorok bizonyos komponenseibe nullértékek. Az SQL is alkalmazza a külső összekapcsolás műveletet, így ha egy lekérdezés kiváltja a külső összekapcsolás művelet végrehajtását, akkor itt is keletkezhetnek nullértékek, bővebben lásd a 6.3.8. alfejezetben. Az SQL más esetekben is produkál nullértékeket. Például sorok beszúrásakor is keletkezhetnek nullértékek, amint azt a 6.5.1. alfejezetben láthatjuk.

A `WHERE` záradékban fel kell készülnünk annak lehetőségére, hogy valamely sor éppen felhasznált komponensének értéke `NULL` is lehet. Két fontos szabályt kell figyelembe vennünk, amikor `NULL` értékekkel dolgozunk.

1. Amikor egy aritmetikai műveletben, mint a × vagy a +, legalább az egyik tag `NULL`, akkor az eredmény is `NULL`.

Nullértékekkel kapcsolatos csapdák

Feltételezhetnénk, hogy a NULL egy olyan érték, amit nem ismerünk, de amely biztosan létezik. A valóságban azonban ez a feltételezés nem helytálló. Például tételezzük fel, hogy x egy sor egy bizonyos komponense, melynek az értelmezési tartománya egész számokból áll. Úgy érvelhetnénk, hogy $0 * x$ értéke biztosan 0, mivel függetlenül attól, hogy mennyi az x értéke, a 0-val való szorzás eredménye 0. Mégis, ha x értéke NULL, akkor a 6.1.6. alfejezet 1. szabálya lép életbe, azaz 0 és NULL szorzata NULL. Hasonlóképpen azt hihetnénk, hogy $x - x$ értéke 0, függetlenül attól, hogy mennyi az x értéke. Mégis, ebben az esetben is az 1. szabály alapján az eredmény NULL.

2. Amikor egy NULLértéket hasonlítunk össze bármely más értékkel, beleértve a NULL-t is, egy összehasonlítási operátor segítségével, mint az = vagy >, az eredmény ISMERETLEN. Az ISMERETLEN egy logikai érték, olyan mint az IGAZ és a HAMIS, hamarosan részletesen is bemutatjuk.

Annak ellenére, hogy a NULL megjelenhet értékként a sorokban, *nem* tekinthető konstansnak. Míg a fenti szabályok olyan esetben alkalmazandók, amikor egy kifejezés értéke NULL, a NULL-t nem használhatjuk direkt módon egy kifejezésben.

6.9. példa. Legyen x értéke NULL. Ekkor $x + 3$ értéke is NULL. Ennek ellenére NULL + 3 nem egy szabályos SQL-kifejezés. Hasonlóképpen $x = 3$ logikai értéke ISMERETLEN, hiszen nem tudjuk eldönteni, hogy az x értéke (amely NULL) egyenlő-e 3-mal. A NULL = 3 nem egy szabályos összehasonlítás az SQL-ben. □

A szabványos útja annak, hogy megtudjuk, egy kifejezés értéke NULL-e vagy sem, az „x IS NULL" kifejezés segítségével történik. A kifejezés értéke IGAZ, ha x értéke NULL, illetve HAMIS, ha nem. Hasonlóképpen „x IS NOT NULL" értéke IGAZ, ha x nem NULL.

6.1.7. Az ISMERETLEN igazságérték

A 6.1.2. alfejezetben bemutattuk, hogy egy összehasonlítás eredménye vagy IGAZ, vagy HAMIS, és ezek a logikai értékek a szokásos logikai műveletekkel – AND, OR és NOT – kombinálhatók. Az előző szakaszban viszont azt ismertük meg, hogy amikor NULL érték is szerepel az összehasonlításban, akkor az eredmény egy harmadik igazságérték: az ISMERETLEN. A továbbiakban bemutatjuk, hogyan működnek a logikai műveletek a háromértékű logikában.

A szabályt könnyű megjegyezni, ha úgy tekintjük, hogy az IGAZ értéke 1 (azaz teljes mértékben igaz), a HAMIS értéke 0 (azaz egyáltalán nem igaz) és az ISMERETLEN értéke 1/2 (vagyis valahol az igaz és a hamis között). Ekkor:

1. Két logikai értékre alkalmazott AND eredménye a két érték minimuma. Tehát x AND y értéke HAMIS, ha legalább az egyik hamis; ISMERETLEN, ha egyik sem HAMIS, de legalább az egyik ISMERETLEN; és IGAZ, ha mindkettő IGAZ.

2. Két logikai értékre alkalmazott OR eredménye a két érték maximuma. Tehát x OR y értéke IGAZ, ha legalább az egyik IGAZ; ISMERETLEN, ha egyik sem IGAZ és legalább az egyik ISMERETLEN; és HAMIS, ha mindkettő HAMIS.

3. A v logikai érték tagadásának értéke $1 - v$. Tehát NOT x értéke IGAZ, ha x HAMIS; HAMIS, ha x IGAZ; és ISMERETLEN, ha x értéke ISMERETLEN.

A 6.2. ábra a három logikai művelet alkalmazásának eredményét mutatja be az x és y logikai változók különböző értékeire. Az utolsó művelet, a NOT csak az x értékétől függ.

x	y	x AND y	x OR y	NOT x
IGAZ	IGAZ	IGAZ	IGAZ	HAMIS
IGAZ	ISMERETLEN	ISMERETLEN	IGAZ	HAMIS
IGAZ	HAMIS	HAMIS	IGAZ	HAMIS
ISMERETLEN	IGAZ	ISMERETLEN	IGAZ	ISMERETLEN
ISMERETLEN	ISMERETLEN	ISMERETLEN	ISMERETLEN	ISMERETLEN
ISMERETLEN	HAMIS	HAMIS	ISMERETLEN	ISMERETLEN
HAMIS	IGAZ	HAMIS	IGAZ	IGAZ
HAMIS	ISMERETLEN	HAMIS	ISMERETLEN	IGAZ
HAMIS	HAMIS	HAMIS	HAMIS	IGAZ

6.2. ábra. Igazságértékek táblázata a háromértékű logika esetén

A select-from-where utasítások WHERE záradékában szereplő SQL-feltételeket a rendszer minden egyes sorra ellenőrzi, és az ellenőrzés eredménye minden egyes sor esetén a három logikai érték (IGAZ, HAMIS, ISMERETLEN) valamelyike. Az eredménybe csak azok a sorok kerülnek bele, melyekre a feltétel kiértékelése IGAZ értéket hozott, azokat a sorokat melyekre a feltétel HAMIS vagy ISMERETLEN, kizárjuk az eredményből. Ez a helyzet – amint a következő példa is szemlélteti – egy újabb meglepő eredményhez vezet a nullértékekkel kapcsolatban, hasonlóan a „Nullértékekkel kapcsolatos csapdák" című bekeretezett részhez.

6.10. példa. Tételezzük fel, hogy a következő relációra

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

kipróbáljuk a következő lekérdezést:

```
SELECT *
FROM Filmek
WHERE hossz <= 120 OR hossz > 120;
```

Ösztönösen arra gondolhatnánk, hogy a Filmek reláció egy másolatát fogjuk kapni, mivel mindegyik film hossza kielégíti a feltételt.

Vizsgáljuk meg azonban azt az esetet, amikor léteznek olyan sorok, amelyekben a hossz attribútum értéke NULL. Így a hossz > 120 és a hossz <= 120 feltételek kiértékelésének eredménye ISMERETLEN lesz. A 6.2. ábra szerint két ISMERETLEN érték OR művelettel történő összekapcsolása ISMERETLEN értéket eredményez. Tehát minden olyan sor esetén, amelyben a hossz attribútumban NULL érték van, a WHERE feltétel eredménye ISMERETLEN lesz. Az ilyen sorok *nem* kerülnek bele az eredménybe. Összefoglalva, a lekérdezés értelme „keressük meg az összes nem NULL hosszértékű Filmek sort". □

6.1.8. Az eredmény rendezése

Szükség lehet arra, hogy egy lekérdezés eredménye bizonyos sorrendbe legyen rendezve. A sorrend valamely attribútum értékén alapulhat, egyenlőség esetén egy második attribútum értékén, további egyenlőség esetén egy harmadik attribútum értékén és így tovább, ugyanúgy, ahogy az 5.2.6. alfejezet τ műveletében is. Az eredmény rendezése érdekében a select-from-where utasításhoz a következő záradékot adjuk hozzá:

ORDER BY<attribútumlista>

A rendezés alapértelmezésben növekvő sorrendben történik, de csökkenően is lekérhetjük az adatokat, ezt az attribútumlistában az érintett attribútum mellett a DESC kulcsszó megadásával jelezzük (a „descending" – „csökkenő" rövidítése). Hasonlóképpen megadhatjuk azt is, hogy a rendezés növekvően történjen az ASC kulcsszóval, de ez felesleges, mert ha nem adjuk meg hogyan történjen, automatikusan így rendeződik.

Az ORDER BY záradék a WHERE és minden más záradék (mint a GROUP BY és a HAVING, melyekkel a 6.4. alfejezetben foglalkozunk) után következik. A rendezés a lekérdezés eredményére vonatkozik, csak annak megjelenítésében van szerepe.

6.11. példa. Írjuk át a 6.1. példát, amely az 1990-es Disney-filmeket kérdezte le a

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

relációból. A filmeket hossz szerint szeretnénk lekérdezni növekvően, és az egyenlő hosszúakat ábécérendben. A megfelelő utasítás:

```
SELECT *
FROM Filmek
WHERE stúdióNév = 'Disney' AND év=1990
ORDER BY hossz, filmcím;
```

A rendezéskor a Filmek összes attribútuma rendelkezésünkre áll akkor is, ha az eredményben nem jelennek meg. Így ha a „SELECT *" kifejezést „SELECT produceAzon"-ra cseréljük, a lekérdezés továbbra is érvényes marad. □

A rendezés további lehetősége, hogy az ORDER BY záradékot követő listában kifejezést is használhatunk éppen úgy, mint a SELECT záradékában is. Például az $R(A, B)$ reláció sorait rendezhetjük a sorok két komponensének összege szerint csökkenő sorrendbe is:

```
SELECT *
FROM R
ORDER BY A+B DESC;
```

6.1.9. Feladatok

6.1.1. feladat. Ha egy lekérdezésben megtalálható a következő SELECT záradék:

```
SELECT A B
```

honnan tudjuk, hogy A és B két különböző attribútum, vagy B másodneve az A-nak?

6.1.2. feladat. Adjuk meg SQL-ben a következő lekérdezéseket, a fejezet elején említett film adatbázisra vonatkozóan:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

a) Keressük meg az MGM stúdió címét.

b) Keressük meg Sandra Bullock születési dátumát.

c) Keressük meg az összes olyan filmsztárt, akik vagy egy 1980-as filmben szerepeltek, vagy egy olyanban, amelynek a címében szerepel a „Szerelem" szó.

d) Keressük meg az összes olyan gyártásirányítót, akiknek a nettó bevétele legalább 10 000 000 $.

e) Keressük meg az összes olyan filmsztárt, akik vagy férfiak, vagy Malibuban laknak (a címük tartalmazza a Malibu szót).

6.1.3. feladat. Adjuk meg a következő SQL-lekérdezéseket, amelyek a 2.4.1. feladat adatbázisára vonatkoznak:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

A lekérdezések eredményeit szemléltessük a 2.4.1. feladat adataival.

a) Keressük meg a modellszámát, sebességét, merevlemez-kapacitását azoknak a PC-knek, melyek ára 1000 $ alatt van.

b) Ugyanaz, mint az *a)* pontban, de nevezzük át a `sebesség` oszlopot `gigahertz`-re, a `merevlemez` oszlopot pedig `gigabájt`-ra.

c) Keressük meg a nyomtatók gyártóit.

d) Keressük meg azon laptopok modellszámát, memóriakapacitását és képernyőnagyságát, melyek több mint 1500 $-ba kerülnek.

e) Keressük meg a `Nyomtató` reláció azon sorait, melyek a színes nyomtatókra vonatkoznak. Vegyük figyelembe, hogy a `színes` attribútum logikai típusú.

f) Keressük meg azon PC-k modellszámát és merevlemezméretét, melyek sebessége 3.2 és 2000 $-nál olcsóbbak.

6.1.4. feladat. Adjuk meg SQL-ben a következő lekérdezéseket, melyek a 2.4.3. feladat adatbázissémájára alapulnak:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

és a lekérdezések eredményeit szemléltessük a 2.4.3. példa adataival.

a) Keressük meg az osztálynevét és országát azon hajóosztályoknak, melyek legalább 10 ágyúval rendelkeznek.

b) Keressük meg az 1918 előtt felavatott hajókat, de az eredményoszlop neve legyen `hajóNév`.

c) Keressük meg a csatákban elsüllyesztett hajók nevét és azon csaták nevét, melyben elsüllyedtek.

d) Keressük meg azon hajókat, amelyek neve megegyezik a hajóosztályuk nevével.

e) Keressük meg az „R” betűvel kezdődő hajók nevét.

! *f)* Keressük meg az olyan hajóneveket, melyek három vagy több szóból állnak (például „Nagy Lajos király”).

6.1.5. feladat. Legyenek a és b egész szám típusú attribútumok, melyek NULL értéket is felvehetnek. Az alábbi feltételek mindegyikére (ilyenek fordulhatnak elő például a WHERE záradékban) adjuk meg pontosan a feltételt kielégítő (a, b)-k halmazát, vegyük figyelembe azokat az eseteket is, amikor a és/vagy b NULL értékű.

a) `a = 10 OR b = 20`

b) `a = 10 AND b = 20`

c) `a < 10 OR a >= 10`

! *d)* `a = b`

! *e)* `a <= b`

! **6.1.6. feladat.** A 6.10. példában a

```
SELECT *
FROM Filmek
WHERE hossz <= 120 OR hossz > 120;
```

lekérdezéssel foglalkoztunk, amely a NULL hosszú filmek esetén ösztöneinkkel ellentétesen viselkedett. Keressünk egyszerűbb, ezzel ekvivalens lekérdezést, amelynek WHERE záradékában egyszerű feltétel áll (olyan feltétel, amelyben nincs sem AND sem OR).

6.2. Több relációra vonatkozó lekérdezések

A relációs algebra legfőbb erőssége az, hogy két vagy több relációt tud kombinálni összekapcsolásokon, szorzatokon, egyesítéseken, metszeteken és különbségeken keresztül. Ezen műveletek mindegyikét alkalmazhatjuk az SQL-ben. A halmazműveletek – egyesítés, metszet és különbség – direkt módon jelennek meg az SQL-ben, ahogy a 6.2.5. alfejezetben szemléltetni fogjuk. Először azonban vizsgáljuk meg, hogy a select-from-where utasítás milyen módon teszi lehetővé szorzatok és összekapcsolások képzését.

6.2.1. Szorzat és összekapcsolás az SQL-ben

Több reláció összekapcsolása nagyon egyszerűen valósítható meg az SQL-ben: fel kell sorolni az összes relációt a FROM záradékban. Ezután a SELECT és WHERE záradékok a FROM záradék minden relációjának minden attribútumát felhasználhatják.

6.12. példa. Tegyük fel, hogy szeretnénk megtudni a *Csillagok háborúja* című film producerének nevét. A következő két relációra van ehhez szükség:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevételI)
```

A producer azonosító száma a `Filmek` relációban megtalálható, így ez a `Filmek` reláción végrehajtott egyszerű lekérdezéssel megszerezhető. Ezután egy második, a `GyártásIrányító` reláción végrehajtott lekérdezéssel megtaláljuk a producerazonosítóhoz tartozó személy nevét.

Ezt a két lépést összevonhatjuk egyetlen lekérdezésbe, mely a `Filmek` és a `GyártásIrányító` relációkra vonatkozik:

```
SELECT név
FROM Filmek, GyártásIrányító
WHERE Filmek.filmcím = 'Csillagok háborúja'
     AND producerAzon = azonosító;
```

Ez a lekérdezés az összes `Filmek` és `GyártásIrányító` sorpárt megvizsgálja, és ezekre ellenőrzi a `WHERE` feltételt:

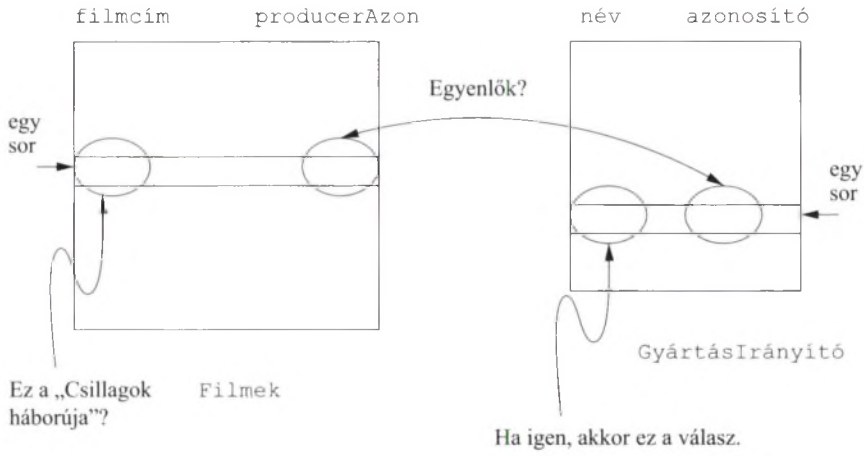
1. A `Filmek` reláció sorának `filmcím` attribútuma `'Csillagok háborúja'` értékű kell legyen.

2. A `Filmek` sor `producerAzon` attribútumának ugyanaz kell legyen az értéke, mint a `GyártásIrányító` sor azonosító attribútumának, azaz a két sor ugyanarra a gyártásirányítóra vonatkozik.

Valahányszor találunk egy sorpárt, amely mindkét feltételnek eleget tesz, a `GyártásIrányító` sor `név` attribútumértéke belekerül az eredménybe. Mindkét feltétel csak az általunk kívánt adatok esetén teljesül, nevezetesen amikor a `Filmek` közül éppen a *Csillagok háborúja* sort, a `GyártásIrányító`-k közül pedig George Lucas sorát vizsgáljuk. Ekkor és csakis ekkor lesz a filmcím megfelelő és az azonosítók (`producerAzon` és `azonosító`) megegyezők. Így `George Lucas` lesz az egyetlen válasz a lekérdezésre. Ezt a folyamatot a 6.3. ábra mutatja. A 6.2.4. alfejezetben írjuk le részletesebben, hogyan is interpretáljuk a több relációra hivatkozó lekérdezéseket. □

6.2.2. Attribútumok megkülönböztetése

Előfordulhatnak olyan lekérdezések, amelyekben két vagy több reláció szerepel, és ezekben a relációkban két vagy több attribútumnak ugyanaz a neve. Ilyen esetben valamilyen módon jeleznünk kell, hogy melyik attribútumról van szó, amikor a közös név szerepel a lekérdezésben. A problémát az SQL úgy oldja meg, hogy megengedi egy relációnévnek és egy pontnak a használatát egy attribútum előtt. Így $R.A$ az R reláció A nevű attribútumát jelképezi.



6.3. ábra. A 6.12. példa lekérdezése a `Filmek` reláció minden sorát párosítja a `GyártásIrányító` összes sorával, és ellenőrzi a két feltétel teljesülését

6.13. példa. Tekintsük a következő két relációt:

```
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Mindkettőben szerepelnek a `név` és `cím` attribútumok. Tételezzük fel, hogy olyan filmszínész-gyártásirányító párosokat keresünk, akiknek azonos a lakcímük. A megfelelő lekérdezés a következő:

```
SELECT FilmSzínész.név, GyártásIrányító.név
FROM FilmSzínész, GyártásIrányító
WHERE FilmSzínész.cím = GyártásIrányító.cím;
```

Ebben a lekérdezésben olyan `FilmSzínész` és `GyártásIrányító` sorpárokat keresünk, melyekben a `cím` attribútumérték megegyezik. A `WHERE` záradék fejezi ki a `cím` attribútumokra vonatkozó feltételt. Mindegyik olyan sorpár esetén, amely megfelel a feltételnek, mindkét `név` attribútumot hozzáadjuk az eredményhez. Így az eredményben olyan névpárok lesznek, mint:

FilmSzínész.név	*GyártásIrányító.név*
Jane Fonda	Ted Turner
...	...

□

A relációnevet és a pontot olyan esetben is az attribútum elé írhatjuk, amikor nincs névazonosságból származó kétértelműség. Például a 6.12. példa lekérdezését így is írhatnánk:

```
SELECT GyártásIrányító.név
FROM Filmek, GyártásIrányító
WHERE Filmek.filmcím = 'Csillagok háborúja'
      AND Filmek.producerAzon = GyártásIrányító.azonosító;
```

A relációnév és pont kombinációt a lekérdezésben szereplő bármely attribútumra alkalmazhatjuk.

6.2.3. Sorváltozók

A relációnév előtagként történő alkalmazása jó megoldás az attribútumok névütközésének elkerülésére addig, amíg több különböző reláció szerepel a lekérdezésben. Néha azonban olyan lekérdezésre van szükségünk, amely ugyanazon reláció két vagy több sorát kombinálja össze. Az R relációt annyiszor sorolhatjuk fel a `FROM` záradékban, ahányszor szükségünk van rá, de valamilyen módon meg kell tudnunk különböztetni az előfordulásait. Az SQL-ben lehetőségünk van arra, hogy a `FROM` záradékban R minden előfordulásához hozzárendeljünk egy másodnevet, melyet sorváltozónak is nevezünk. A `FROM` záradékban R minden előfordulása után következhet az `AS` kulcsszó és a sorváltozó neve; az `AS` kulcsszó általában elhagyható.

R attribútumait a `SELECT` és `WHERE` záradékokban megkülönböztethetjük egy előtag segítségével, mely a megfelelő sorváltozóból és egy pontból áll. Tehát a sorváltozó az R reláció másodneveként szerepel, és bárhol használható az R helyett.

6.14. példa. Míg a 6.13. példában olyan filmszínészeket és gyártásirányítókat kerestünk, akiknek azonos a lakcímük, hasonlóképpen kereshetünk olyan színészpárokat is, akiknek megegyezik a lakcímük. A lekérdezés lényegében ugyanaz, de most a `FilmSzínész` reláció két sorát kell párosítani, és nem a `FilmSzínész` reláció egy sorát a `GyártásIrányító` reláció egy sorával. A `FilmSzínész` reláció két előfordulására sorváltozókat használunk másodnévként, így a lekérdezés alakja a következő lesz:

```
SELECT Színész1.név, Színész2.név
FROM FilmSzínész AS Színész1, FilmSzínész AS Színész2
WHERE Színész1.cím = Színész2.cím
      AND Színész1.név < Színész2.név;
```

A `FROM` záradékban a `Színész1` és `Színész2` sorváltozókat használjuk a `FilmSzínész` reláció másodneveiként. A `SELECT` záradékban a sorváltozók segítségével különböztetjük meg a két sor azonos komponenseit. A másodneveket a `WHERE` záradékban is használjuk annak kifejezésére, hogy a két (`Színész1`- és `Színész2`-beli) `FilmSzínész` sor `cím` attribútumának ugyanaz az értéke.

Sorváltozók és relációnevek

Gyakorlatilag a SELECT és WHERE záradékokban az attribútumokra *mindig* úgy utalunk, mint egy sorváltozó komponenseire. Viszont ha egy reláció csak egyszer szerepel a FROM záradékban, akkor a relációnevet használhatjuk mint sorváltozót. Így a FROM záradékbeli R relációnevet tekinthetjük úgy, mint egy rövidítést az R AS R helyett. Továbbá, ahogy már láttuk, ha egy attribútum egyértelműen egy relációhoz tartozik, akkor a reláció (sorváltozóként használt) neve elhagyható.

A WHERE záradék második feltétele, `Színész1.név < Színész2.név`, azt fejezi ki, hogy az első színész neve ábécérendben megelőzi a második színész nevét. Ha ezt a feltételt elhagynánk, akkor `Színész1` és `Színész2` vonatkozhatnának ugyanarra a sorra. Ebben az esetben azt találnánk, hogy a két sorban a címek természetesen megegyeznek, tehát az eredménybe belekerülne minden színész neve önmagával párosítva.[3] A második feltétel azt is kikényszeríti, hogy az eredményben minden közös lakcímű színészpár csak egyszer szerepeljen, ábécérendben. Ha a <> (nem egyenlő) jelet használnánk összehasonlítási operátorként, akkor az eredmény minden színész házaspárt kétszer tartalmazna, például:

Színész1.név	*Színész2.név*
Paul Newman	Joanne Woodward
Joanne Woodward	Paul Newman
...	...

□

6.2.4. Lekérdezések értelmezése

Az előzőekben bemutatott select-from-where kifejezések értelmezésére számos lehetőség van. Ezek az értelmezések mind *egyenértékűek* abban az értelemben, hogy ugyanazt a választ eredményezik, ha ugyanarra az adatbázis-előfordulásra alkalmazzuk ugyanazt a lekérdezést. Vizsgáljuk meg őket sorban!

Beágyazott ciklusok

A példákban eddig a sorváltozók szemantikáját alkalmaztuk. Idézzük fel, hogy egy reláció másodneve egy sorváltozó, amely a relációhoz tartozó összes sor értékét felveszi. Egy másodnév nélküli relációnév értelmezhető saját másodneveként, amint azt a „sorváltozók és relációnevek" bekeretezett részben említettük.

[3] Ugyanaz a probléma előfordulhat a 6.13. példában is, ha egy személy filmszínész és gyártásirányító is. A problémát hasonlóan tudjuk megoldani, ha feltételként szabjuk, hogy a két név legyen különböző.

Ha több sorváltozónk van, mindegyikhez rendelhetünk egy-egy ciklust, melyben a változó végigmegy a neki megfelelő reláció összes során. A ciklusok egymásba vannak ágyazva. A sorváltozók mindegyik értékadásakor leellenőrizzük, hogy a WHERE feltétel igaz-e vagy sem. Ha igaz, akkor a SELECT záradékbeli komponenseknek megfelelő értékekből összeállított sort az eredménybe illesztjük. A lekérdezést feldolgozó algoritmust a 6.4. ábra érzékelteti.

```
Legyenek a FROM záradékban megadott relációk R1, R2, ..., Rn;
FOR t1 sorra az R1 relációban DO
    FOR t2 sorra az R2 relációban DO
        ...
        FOR tn sorra az Rn relációban DO
            IF a where záradék igaz, amikor az attribútumokban
            t1, t2, ..., tn megfelelő értékei találhatóak
            THEN
                t1, t2, ··· , tn-nek megfelelően kiértékeljük a
                select záradék attribútumait és az értékekből
                alkotott sort az eredményhez adjuk
```

6.4. ábra. Egy egyszerű SQL-lekérdezés kiértékelése

Párhuzamos értékadás

Ebben az esetben nem kell explicit módon beágyazott ciklusokat létrehozni. Ehelyett tekintsük tetszőleges sorrendben vagy párhuzamosan a sorváltozók összes lehetséges értékadását a megfelelő relációkból. Minden egyes értékadásra megvizsgáljuk, hogy a WHERE feltétel igaz-e. Minden egyes olyan értékadás, amelyre a WHERE igaz, egy sort ad a válaszhoz. A válaszbeli sorok a SELECT záradék attribútumaiból épülnek fel, az aktuális értékadásnak megfelelően.

Konverzió a relációs algebrába

Egy harmadik megközelítés az SQL-lekérdezést a relációs algebrával köti össze. A FROM záradék sorváltozóiból indulunk ki, és tekintjük a hozzájuk tartozó relációk Descartes-szorzatát. Ha két változó ugyanarra a relációra vonatkozik, akkor a reláció kétszer szerepel a szorzatban, és az attribútumait átnevezzük úgy, hogy minden attribútumnak egyedi neve legyen. Hasonlóan, a különböző relációkban szereplő azonos attribútumneveket is átnevezzük, hogy elkerüljük a kétértelműséget.

A WHERE záradékot átalakítjuk egy kiválasztási feltétellé, amelyet alkalmazunk az elkészített szorzatra. A WHERE záradékbeli mindegyik attribútumutalást kicseréljük arra a szorzatbeli attribútumra, amelynek megfelel. Végül a SELECT záradék alapján létrehozzuk a kifejezések listáját, a záró (kiterjesztett) vetítési művelet számára. Mint ahogyan a WHERE záradéknál is tettük, a SELECT

Az SQL-szemantika egy meglepő következménye

Tételezzük fel, hogy R, S és T unáris (egyoszlopos) relációk, és mindegyik egyedüli attribútuma az A. Szeretnénk megkeresni azokat az elemeket, amelyek az R-ben és vagy az S-ben, vagy a T-ben (vagy mindkettőben) megtalálhatók. Tehát az $R \cap (S \cup T)$ halmazt szeretnénk meghatározni. Azt hihetnénk, hogy a következő lekérdezés a megfelelő választ eredményezi:

```
SELECT R.A
FROM R, S, T
WHERE R.A = S.A OR R.A = T.A;
```

Vizsgáljuk meg azt a speciális esetet, amikor T üres. Mivel az $R.A = T.A$ egyenlőség nem teljesülhet, az „OR" művelettel kapcsolatos ismereteink alapján azt várnánk, hogy az eredmény $R \cap S$ legyen. Mégis, a 6.2.4. alfejezet három értelmezése közül bármelyiket is választjuk ki, a lekérdezés eredménye az üres halmaz lesz, függetlenül attól, hogy R-nek és S-nek hány közös eleme van. Ha a 6.4. ábrán bemutatott beágyazott ciklus szemantikát használjuk, észrevehető, hogy a T változó ciklusa 0 hosszúságú, mivel annak a relációnak, mely fölött a T változik, nincs egy sora sem. Így a legbelső ciklusbeli if utasítás egyszer sem kerül végrehajtásra, tehát az eredmény üres lesz. Hasonlóképpen, ha a sorváltozók lehetséges értékadásait tekintjük, kiderül, hogy a T változóhoz nem rendelhető érték, tehát nincs egy lehetőség sem a sorváltozók érték-hozzárendelésére. Végül, ha a Descartes-szorzatos megközelítést alkalmazzuk, akkor a kiindulópontunk az $R \times S \times T$ szorzat, amely üres, mivel T is üres.

záradékban minden attribútumot kicserélünk a szorzat neki megfelelő attribútumára.

6.15. példa. Írjuk át a 6.14. példát relációs algebrára. Először is a `FROM` záradékban két sorváltozó van, mindkettő a `FilmSzínész` relációra vonatkozik. Ezért az algebrai kifejezés így kezdődik:

$$\texttt{FilmSzínész} \times \texttt{FilmSzínész}$$

Az eredményrelációnak nyolc attribútuma van, az első négy a `FilmSzínész` reláció első másolatának `név`, `cím`, `nem` és `születésiDátum` attribútumainak felel meg, a következő négy pedig a `FilmSzínész` reláció második másolatának ugyanazon attribútumainak. Az attribútumokat jelölhetnénk a sorváltozóval és egy ponttal – például `Színész1.nem` –, de a tömörség kedvéért nevezzük át az attribútumokat $A_1, A_2, \ldots, A_8$-nak. Így A_1 megfelel `Színész1.név`-nek, A_5 pedig `Színész2.név`-nek és így tovább.

Ezzel az attribútumátnevezési stratégiával a WHERE záradék kiválasztási feltétele így alakul: $A_2 = A_6$ és $A_1 < A_5$. A vetítési lista A_1, A_5. Így a következő kifejezés adja meg a lekérdezésnek megfelelő algebrai kifejezést:

$$\pi_{A_1,A_5}\Big(\sigma_{A_2=A_6 \text{ AND } A_1<A_5}\big(\rho_{M(A_1,A_2,A_3,A_4)}(\texttt{FilmSzínész}) \times \rho_{N(A_5,A_6,A_7,A_8)}(\texttt{FilmSzínész})\big)\Big)$$

□

6.2.5. Egyesítés, metszet és különbség az SQL-ben

Néha a relációs algebra halmazműveleteit: az egyesítést, a metszetet és a különbséget kell használnunk relációk kombinálására. Az SQL biztosítja a megfelelő operátorokat, amelyeket lekérdezések eredményeire lehet alkalmazni, feltéve, hogy a lekérdezések ugyanolyan attribútumhalmazú (megegyező nevű és típusú attribútumhalmazok) relációkat eredményeznek. A ∪, ∩ és − műveleteknek megfelelő kulcsszavak: UNION, INTERSECT és EXCEPT. Ezeket a kulcsszavakat két zárójel közé írt lekérdezés közé kell helyezni.

6.16. példa. Tételezzük fel, hogy azokat a színésznőket keressük, akik gyártásirányítók is, 10 000 000 $ feletti nettó bevétellel. A következő két relációra van szükségünk:

```
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

A lekérdezést a 6.5. ábra mutatja be. Az 1–3. sorok egy olyan relációt eredményeznek, melynek sémája (név, cím), és sorai a színésznők nevét és címét tartalmazzák.

```
1)  (SELECT név, cím
2)   FROM FilmSzínész
3)   WHERE nem = 'N')
4)       INTERSECT
5)  (SELECT név, cím
6)   FROM GyártásIrányító
7)   WHERE nettóBevétel > 10000000);
```

6.5. ábra. Színésznők és gazdag gyártásirányítók metszete

Hasonlóképpen, az 5–7. sorok a „gazdag" gyártásirányítókat eredményezik, akiknek nettó bevétele meghaladja a 10 000 000 $-t. Ez a lekérdezés szintén egy olyan relációt eredményez, melynek sémája csak a név és cím attribútumokból áll. Mivel a két relációséma megegyezik, alkalmazhatjuk rá a metszet műveletet, melyet a 4. sor ad meg. □

SQL-lekérdezések olvashatósága

Általában az SQL-lekérdezéseket úgy írjuk, hogy az olyan fontos kulcsszavak, mint a FROM vagy WHERE új sorban kezdődjenek. Ez a stílus sokkal jobban láthatóvá teszi a lekérdezés struktúráját. Egy rövid lekérdezést azonban egy sorba is írhatunk, mint ahogy a 6.17. példában tettük. A lekérdezés tömörségét megtartva, ez a stílus is jó olvashatóságot biztosít.

6.17. példa. Hasonló módon meghatározhatjuk a két személyhalmaz különbségét. A következő lekérdezés:

```
(SELECT név, cím FROM FilmSzínész)
   EXCEPT
 (SELECT név, cím FROM GyártásIrányító);
```

azon színészek nevét és címét adja meg, akik nem gyártásirányítók, függetlenül a nemtől és a nettó bevételtől. □

A fenti két példában a két halmaz attribútumai szerencsére megegyeztek. Szükség esetén azonban – amint a 6.3. példában láttuk – átnevezhetjük az attribútumokat, hogy azonos attribútumhalmazokat kapjunk.

6.18. példa. Keressük meg azon filmcímeket és éveket, amelyek a Filmek vagy a SzerepelBenne relációkban megtalálhatók:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
```

Ideális esetben a két relációban megtalálható filmhalmazoknak meg kellene egyezniük. Gyakorlatilag azonban könnyen megtörténhet, hogy van olyan film, amelyhez nincsenek felsorolva a benne szereplő színészek vagy olyan SzerepelBenne sor is létezhet, amelynek megfelelő sor nincs a film relációban[4]. Így a lekérdezés:

```
(SELECT filmcím, év FROM Filmek)
    UNION
(SELECT filmCím AS filmcím, filmÉv AS év FROM SzerepelBenne);
```

Az eredményreláció attribútumai: filmcím és év, azokat a filmeket fogja tartalmazni, amelyek legalább az egyik relációban szerepelnek. □

[4] Léteznek módszerek az ilyen eltérések megakadályozására, lásd a 7.1.1. alfejezetben.

6.2.6. Feladatok

6.2.1. feladat. Az aktuális adatbázist használva:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

adjuk meg SQL-ben a következő lekérdezéseket:

a) Keressük meg a *Titanic* férfi szereplőit.

b) Kik szerepeltek az MGM által 1995-ben gyártott filmekben?

c) Ki az MGM stúdió elnöke?

! *d)* Mely filmek hosszabbak az *Elfújta a szél* című filmnél?

! *e)* Kik azok a gyártásirányítók, akiknek több a nettó bevételük, mint Merv Griffinnek?

6.2.2. feladat. A 2.4.1. feladatbeli adatbázis:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

alapján adjuk meg a megfelelő lekérdezéseket és a 2.4.1. feladat adataival szemléltessük az eredményt.

a) Keressük meg azon laptopok gyártóit és sebességét, amelyeknek legalább harminc gigabájtos merevlemeze van.

b) Keressük meg a B gyártó által készített bármilyen típusú termékek számát és árát.

c) Keressük meg azon gyártókat, akik laptopokat gyártanak, de PC-ket nem.

! *d)* Keressük meg azon merevlemezméreteket, amelyek legalább két PC-ben előfordulnak.

! *e)* Keressük meg azon PC-modellpárokat, amelyeknek ugyanakkora a sebessége és a memóriája. Egy párt csak egyszer listázzunk ki, azaz ha (i, j) az eredményben van, akkor (j, i) ne kerüljön bele.

!! *f)* Keressük meg azokat a gyártókat, akik legalább két különböző, 3.0-nál nagyobb sebességű számítógépet gyártanak (PC-t vagy laptopot).

6.2.3. feladat. A 2.4.3. feladat adatbázisát és adatait felhasználva adjuk meg a következő lekérdezéseket és eredményeiket:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

a) Keressük meg a 35 000 tonnánál súlyosabb hajókat.

b) Keressük meg a guadalcanali csatában részt vevő hajók nevét, vízkiszorítását és ágyúszámát.

c) Keressük meg az adatbázisban megtalálható összes hajót. (Vegyük figyelembe, hogy nem biztos, hogy az összes hajót tároltuk a `Hajók` relációban.)

! *d)* Keressük meg azokat az országokat, amelyeknek csatahajóik és cirkálóik is vannak.

! *e)* Keressük meg azokat a hajókat, amelyek megsérültek egy csatában, de részt vettek később egy másikban.

! *f)* Keressük meg azokat a csatákat, amelyekben legalább három azonos országbeli hajó vett részt.

! **6.2.4. feladat.** A relációs algebra egyik leggyakoribb kifejezése a következő:

$$\pi_L\big(\sigma_C(R_1 \times R_2 \times \cdots \times R_n)\big)$$

ahol L az attribútumok tetszőleges listája, míg C egy tetszőleges feltétel. Az $R_1, R_2, \ldots, R_n$ lista tartalmazhatja ugyanazt a relációt többször is, ilyenkor az attribútumokat átnevezzük. Mutassuk meg, hogyan lehet kifejezni egy ilyen lekérdezést az SQL-ben.

! **6.2.5. feladat.** A relációs algebra egy másik tipikus lekérdezése:

$$\pi_L\big(\sigma_C(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n)\big)$$

Hasonló feltevésekkel élünk, mint a 6.2.4. feladatban azzal a különbséggel, hogy szorzat helyett természetes összekapcsolást használunk. Mutassuk meg, hogyan lehet egy ilyen lekérdezést SQL-ben megfogalmazni.

6.3. Alkérdések

Az SQL-ben egy lekérdezés kiértékelését egy másik lekérdezés különféle módokon elősegítheti. Az olyan lekérdezést, amely egy másik lekérdezés része, *alkérdés*nek nevezzük. Az alkérdéseknek is lehetnek alkérdéseik, és így tovább a megkívánt mélységig. Az alkérdések egy példájával már találkoztunk a 6.2.5. alfejezetben, amikor két alkérdés eredményein végrehajtott egyesítéssel, metszetképzéssel vagy különbségképzéssel állítottuk elő a teljes lekérdezés eredményét. Számos egyéb helyzetben is használhatunk alkérdéseket:

1. Ha az alkérdés egy egyszerű konstans eredményt szolgáltat, akkor ezt használhatjuk a `WHERE` záradékban, más értékekkel való összehasonlításokban.
2. Ha az alkérdés eredménye reláció, akkor a `WHERE` záradékban ezt különböző módokon használhatjuk.
3. Ha az alkérdés eredménye reláció, akkor ezt az eredményrelációt a `FROM` záradékban – nevet is adva neki – ugyanúgy használhatjuk, mint ahogy bármely más relációt is.

6.3.1. Skalár értéket adó alkérdések

Az olyan atomi értékeket, amelyek egy sor egyik komponenseként előfordulhatnak, *skalár*nak nevezzük. Egy select-from-where kifejezés bármilyen oszlopszámú relációt eredményezhet és bármennyi sor lehet a relációban. Néha viszont csak egy bizonyos attribútum értékeire van szükségünk. Ezenkívül a kulcsok vagy más információk alapján arra következtethetünk, hogy annak az attribútumnak csak egy értéke lesz az eredményben.

Ha így van, a zárójelek közé tett select-from-where kifejezést konstansként használhatjuk. Minden olyan helyen megjelenhet a `WHERE` záradékban, ahol egy konstans vagy egy sor egy attribútuma szerepelhet. Például az alkérdés eredményét összehasonlíthatjuk egy konstanssal vagy attribútummal.

6.19. példa. Térjünk vissza a 6.12. példához és keressük meg a *Csillagok háborújá*nak gyártásirányítóját. A következő két relációt kell lekérdeznünk:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

mert csak az első tartalmazza a filmcímre vonatkozó információkat és csak a második a gyártásirányító neveket. Az információk az azonosító számon kapcsolódnak össze. Ezek a számok egyértelműen azonosítják a gyártásirányítókat. A megfelelő lekérdezés:

```
SELECT név
FROM Filmek, GyártásIrányító
WHERE Filmek.filmcím = 'Csillagok háborúja'
     AND producerAzon = azonosító;
```

A WHERE záradékban a filmcím attribútum elé írtuk a reláció nevét és a pontot, mivel az attribútum mindkét relációban megtalálható.

Másképpen is vizsgálhatjuk ezt a lekérdezést. A Filmek relációból megtudhatjuk a *Csillagok háborúja* című film gyártásirányítójának azonosítóját. A GyártásIrányító relációból az azonosító alapján megtudhatjuk a megfelelő személy nevét. Az azonosító meghatározása egy alkérdésként fogalmazható meg. Az alkérdés eredményét, amely várhatóan egy érték lesz, a „fő" lekérdezésben használhatjuk fel. A lekérdezést a 6.6. ábra szemlélteti.

```
1) SELECT név
2) FROM GyártásIrányító
3) WHERE azonosító =
4)     (SELECT producerAzon
5)      FROM Filmek
6)      WHERE filmcím = 'Csillagok háborúja'
        );
```

6.6. ábra. A *Csillagok háborúja* producereinek megkeresése beágyazott alkérdéssel

A 4–6. sorok tartalmazzák az alkérdést. Ha csak ezt a lekérdezést vizsgáljuk, észrevehető, hogy az eredmény egy egyoszlopos reláció lesz, amely várhatóan csak egy sort fog tartalmazni. A sor (12345) alakú lesz, azaz egyetlen oszlop, amelyben az az egész szám található, amely George Lucas azonosítója. Ha nulla, vagy egynél több sort eredményez a 4–6. sorok közötti lekérdezés, akkor a lekérdezés futás közbeni hibát fog jelezni.

Az alkérdés lefutása után az 1–3. sorok közötti lekérdezés is végrehajtható, mintha az egész alkérdés helyett az 12345 konstans lenne behelyettesítve. Azaz a „fő" lekérdezés úgy hajtódik végre, mint az alábbi:

```
SELECT név
FROM GyártásIrányító
WHERE azonosító = 12345;
```

A lekérdezés eredménye George Lucas kell legyen. □

6.3.2. Relációkat tartalmazó feltételek

Léteznek olyan SQL-operátorok, amelyeket alkalmazhatunk egy R relációra, és az eredmény logikai érték lesz. Az R relációt egy alkérdéssel kell előállítani. Ha például a Foo tárolt táblára akarjuk a most tárgyalandó operátorokat alkalmazni, akkor erre a célra a (SELECT * FROM Foo) alkérdést használjuk. Ugyanez a trükk használható a relációk uniójának, metszetének és különbségének eseteire is.

Az operátorok közül néhánynak – IN, ALL és ANY – először olyan egyszerű használatát mutatjuk be, amelyben használunk még egy s skaláris értéket is, ebben az esetben R egyoszlopos reláció kell legyen. Az operátorok definíciói:

1. EXISTS R egy olyan feltétel, amely akkor és csak akkor igaz, ha R nem üres.

2. s IN R akkor és csak akkor igaz, ha s egyenlő valamelyik R-beli értékkel. Hasonlóképpen, s NOT IN R akkor és csak akkor igaz, ha s egyetlen R-beli értékkel sem egyenlő. Itt feltételeztük, hogy R egyoszlopos reláció. A 6.3.3. alfejezetben vizsgálni fogjuk az IN és NOT IN operátorok kiterjesztését, amikor R-nek több attribútuma van és s egy sor.

3. s > ALL R akkor és csak akkor igaz, ha s nagyobb mint az R egyoszlopos reláció minden értéke. Hasonlóképpen, a > operátort analóg módon kicserélhetjük bármelyikkel a többi öt összehasonlítási operátor közül. Például, s <> ALL R ugyanazt eredményezi, mint az s NOT IN R.

4. s > ANY R akkor és csak akkor igaz, ha s nagyobb az R egyoszlopos reláció legalább egy értékénél. Hasonlóképpen, bármelyiket használhatjuk a többi öt összehasonlítási operátorból a > helyett. Például, s = ANY R ugyanaz mint az s IN R.

Hasonlóan más logikai kifejezésekhez, az EXISTS, ALL és ANY operátorokat tagadhatjuk, ha a NOT kulcsszót az egész kifejezés elé helyezzük. Így a NOT EXISTS R akkor és csak akkor igaz, ha R üres. NOT s > ALL R akkor és csak akkor igaz, ha s nem nagyobb minden R-beli értéknél, míg a NOT s > ANY R akkor és csak akkor igaz, ha s nem nagyobb egyetlen R-beli értéknél sem. Hamarosan példákon keresztül fogjuk bemutatni az operátorok használatát.

6.3.3. Sorokat tartalmazó feltételek

Az SQL-ben egy sor: skalárértékek zárójelek közötti felsorolása. Ilyen például az (123, 'labda') és a (név, cím, nettóBevétel). Az első példában a sor komponensei konstansok, a másodikban attribútumok. Megengedett a konstansok és attribútumok kombinálása.

Ha egy t sornak és R relációnak ugyanannyi komponense van, akkor a 6.3.2. alfejezetben leírt kifejezésekben hasonlíthatjuk össze azokat. Ilyen lehet például a t IN R vagy a t <> ANY R. A második kifejezés azt jelenti, hogy létezik az R-ben t-től különböző sor. Jegyezzük meg, hogy amikor egy sort összehasonlítunk egy R reláció soraival, a komponenseket az attribútumok R-beli sorrendjének megfelelően kell összehasonlítani.

6.20. példa. A 6.7. ábrán látható SQL-lekérdezés a következő három relációra vonatkozik:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

```
1) SELECT név
2) FROM GyártásIrányító
3) WHERE azonosító IN
4)     (SELECT producerAzon
5)      FROM Filmek
6)      WHERE (filmcím, év) In
7)          (SELECT filmCím, filmÉv
8)           FROM SzerepelBenne
9)           WHERE színész = 'Harrison Ford'
             )
       );
```

6.7. ábra. Így találjuk meg Harrison Ford filmjeinek gyártásirányítóit

A lekérdezés azon filmek gyártásirányítóit keresi, amelyekben Harrison Ford szerepelt. Egy „fő" lekérdezésből áll, azon belül két beágyazott lekérdezésből.

A beágyazott lekérdezéseket belülről kifelé értékeljük ki. Induljunk ki a legbelső beágyazott lekérdezésből, amely a 7–9. sorok között található. A lekérdezés a `SzerepelBenne` reláció sorait vizsgálja és azokat találja meg, amelyekben a `színészNév` komponens értéke `'Harrison Ford'`. Ezeknek a filmeknek a címét és gyártási évét tartalmazza az eredmény. Emlékezzünk vissza, hogy a `filmcím` és az `év` együtt alkotja a `Filmek` kulcsát, amely alapján pontosan azonosítani lehet. Így a 7–9. sorok közötti lekérdezés eredménye a 6.8. ábrán bemutatotthoz hasonló sorokat fog tartalmazni.

filmcím	*év*
Csillagok háborúja	1977
Az elveszett frigyláda fosztogatói	1981
A szökevény	1993
...	...

6.8. ábra. Filmcím-év párok, amelyeket a legbelső alkérdés eredményez

Tekintsük a 4–6. sorok közötti középső alkérdést. Olyan `Filmek` sorokat keres, melyek címe és gyártási éve a 6.8. ábrán szemléltetett relációban található. Minden egyes megtalált sor esetén az eredménybe kerül a gyártásirányító azonosítója. Így a középső alkérdés eredménye a Harrison Ford-filmek gyártásirányítóinak azonosítóit fogja tartalmazni.

Végül nézzük meg az 1–3. sorok közötti „fő" lekérdezést. Ez megvizsgálja a `GyártásIrányító` reláció sorait, hogy megtalálja azokat, amelyek `azonosító` komponense a középső lekérdezés által eredményezett halmazban van.

Minden egyes megfelelő sor esetén a gyártásirányító neve az eredménybe kerül, amely így azon filmek gyártásirányítóit fogja tartalmazni, amelyekben Harrison Ford szerepelt. □

A 6.7. ábra beágyazott lekérdezését, ugyanúgy mint sok más beágyazott lekérdezést, átírhatjuk egy egyszerű select-from-where lekérdezésre. A lekérdezés `FROM` záradékában szerepelni fog az összes olyan reláció, amely a 6.7. ábra fő lekérdezésében vagy valamelyik alkérdésben megjelenik. Az `IN` kapcsolat helyett egyenlőségek lesznek a `WHERE` záradékban. Így a 6.9. ábrán látható lekérdezés lényegében megegyezik a 6.7. ábra lekérdezésével. Különbségek csak a gyártásirányítók ismétlődésének kezelésében vannak, ahogyan a 6.4.1. alfejezetben látni fogjuk.

```
SELECT név
FROM GyártásIrányító, Filmek, SzerepelBenne
WHERE azonosító = producerAzon AND
      Filmek.filmcím = SzerepelBenne.filmCím AND
      Filmek.év = SzerepelBenne.filmÉv AND
      színészNév = 'Harrison Ford';
```

6.9. ábra. Ford gyártásirányítói beágyazott lekérdezések nélkül

6.3.4. Korrelált alkérdések

A legegyszerűbb alkérdések csak egyszer kerülnek kiértékelésre, majd az eredményt egy magasabb rendű lekérdezés hasznosíthatja. A beágyazott alkérdéseket bonyolultabb módon is lehet használni úgy, hogy az alkérdés többször legyen kiértékelve. Minden egyes kiértékelés megfelel egy olyan értékadásnak, amely az alkérdésen kívüli sorváltozóból származik. Egy ilyen típusú alkérdést *korrelált* alkérdésnek nevezünk. Kezdjük egy példával:

6.21. példa. Keressük meg azokat a filmcímeket, amelyek két vagy több filmhez is tartoznak. Kezdjük a megoldást egy külső lekérdezéssel, amely a következő reláció minden sorát megvizsgálja:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

Minden egyes sor esetén egy alkérdésen keresztül megvizsgáljuk, hogy létezik-e egy ugyanolyan című film, amelyet később gyártottak (feltesszük, hogy azonos című filmek nem készültek ugyanabban az évben). Az egész lekérdezést a 6.10. ábra szemlélteti.

Mint a többi beágyazott lekérdezés esetén is, kezdjük a vizsgálatot a 4–6. sorok közötti belső lekérdezéssel. Ha a `Régi.filmcím` komponenst a 6. sorban egy konstanssal cserélnénk ki, mint például a `'King Kong'`, a lekérdezés a `'King`

Kong' című filmek gyártási évét keresné. Az aktuális lekérdezés ettől kicsit különbözik. Az egyedüli probléma az, hogy nem tudjuk, mi a Régi.filmcím értéke. Amikor azonban az 1–3. sorok közötti külső lekérdezés folyamán a Filmek sorokon haladunk végig, a Régi.filmcím-nek mindig van valamilyen értéke. Ezt az értéket használva végrehajtjuk a 4–6. sorok közötti lekérdezést azért, hogy a 3–6. sorok közötti WHERE feltétel igazságértékét meghatározzuk.

```
1) SELECT filmcím
2) FROM Filmek Régi
3) WHERE év < ANY
4)     (SELECT év
5)      FROM Filmek
6)      WHERE filmcím = Régi.filmcím
        );
```

6.10. ábra. A több mint egyszer előforduló filmcímek megkeresése

A 3. sorbeli feltétel akkor lesz igaz, ha létezik olyan film, amelynek a címe megegyezik a Régi.filmcím értékével és később gyártották, mint a Régi sorváltozó aktuális értékéhez tartozó filmet. Ez a feltétel igaz lesz mindaddig, amíg a Régi sorváltozó év értéke nem az utolsó év, amelyben egy olyan című filmet gyártottak. Tehát az 1–3. sorok közötti lekérdezés eggyel kevesebbszer fog kiírni egy filmcímet, mint ahányszor ilyen című filmet gyártottak. Egy kétszer gyártott filmet egyszer fog az eredmény tartalmazni, egy háromszor gyártottat kétszer stb.[5] □

A korrelált lekérdezések használata közben figyelembe kell vennünk a nevek *érvényességi körére* vonatkozó szabályokat. Általában egy lekérdezésben szereplő attribútum a lekérdezés FROM záradékában szereplő valamelyik sorváltozóhoz tartozik, ha a sorváltozó relációja tartalmazza azt az attribútumot. Ha nem tartalmazza egyik sem, akkor megvizsgáljuk az őt körbevevő lekérdezést, azután az azt körbevevőt stb. Így a 6.10. ábra 4. sorában szereplő év és a 6. sorában szereplő cím annak a sorváltozónak az attribútumai, amely az 5. sorban bevezetett Filmek reláció sorain megy végig – azaz annak a Filmek relációnak a másolatán, amelyet a 4–6. sorok közötti lekérdezés vizsgál.

Egy attribútumot azonban egy másik sorváltozóhoz is kapcsolhatunk, ha a sorváltozó nevét és egy pontot írunk az attribútum elé. Ezért vezettük be a külső lekérdezésben a Filmek reláció Régi másodnevét, és ezért hivatkoztunk a 6. sorban a Régi.filmcím komponensre. Jegyezzük meg, hogy ha a 2. és 5. sorokban található FROM záradékok relációi nem egyeztek volna meg, akkor nem lett volna szükség egy másodnévre. Ehelyett az alkérdésben nyugodtan használhattuk volna a 2. sorbeli reláció attribútumait.

[5] Ez a példa az első olyan alkalom, amikor fel kell idéznünk, hogy az SQL relációi nem halmazok, hanem multihalmazok. Az ismétlődések több módon is eltávolíthatók, ahogy azt a 6.4. alfejezetben vizsgálni fogjuk.

6.3.5. Alkérdések a FROM záradékban

Az alkérdéseket – pontosabban az alkérdés által létrehozott relációt – a FROM záradékban is használhatjuk. A FROM záradék listájában a tényleges reláció(k) helyett előfordulhatnak alkérdések is. Ilyenkor ezeket zárójelek között adhatjuk meg. Minthogy az alkérdés eredményrelációjának nincs neve, másodnévadás lehetőségével élve nevet kell adnunk neki. Így az alkérdés által létrehozott reláció soraira már ugyanúgy tudunk hivatkozni, mint a FROM lista többi relációinak soraira.

6.22. példa. Vizsgáljuk meg újra a 6.20. példabeli feladatot, melyben Harrison Ford filmjeinek gyártásirányítóit kerestük. Tegyük fel, hogy van olyan relációnk, amely tartalmazza ezen filmek gyártásirányítóinak producerAzon-jait. Ezután már egyszerű dolog megtalálni a gyártásirányítók neveit a GyártásIrányító relációban. A 6.11. ábrán ezt a lekérdezést látjuk.

```
1) SELECT név
2) FROM GyártásIrányító,
          (SELECT producerAzon
3)        FROM Filmek, SzerepelBenne
4)        WHERE Filmek.filmcím = SzerepelBenne.filmCím AND
5)              Filmek.év = SzerepelBenne.filmÉv AND
6)              színészNév = 'Harrison Ford'
7)        ) Prod
8) WHERE azonosító = Prod.producerAzon ;
```

6.11. ábra. Ford gyártásirányítóinak keresése, alkérdést használva a FROM záradékban

A 2–7. sorokban látjuk a FROM záradékot, amelyben a GyártásIrányító-n kívül, a beágyazott alkérdéssel létrehozott, Prod másodnévvel ellátott reláció is szerepel. Ez a beágyazott alkérdés a 3–5. sorokban a Filmek és a SzerepelBenne összekapcsoltjából kiválogatja azon filmek producerAzon-jait (2. sor), amelyekben szerepel Harrison Ford (6. sor). A 7. sorban az alkérdéssel előállított halmaz másodneveként Prod-ot adtunk.

A 8. sorban a GyártásIrányító és az alkérdéssel létrehozott Prod relációkra vonatkozó válogatási feltételt – az azonosítók egyenlősége – használjuk. A GyártásIrányító előbbi válogatási feltételnek megfelelő soraiból a nevek adják a lekérdezés végeredményét (1. sor). □

6.3.6. Összekapcsolások az SQL-ben

Két relációra az összekapcsolás-művelet különféle változatait alkalmazva számos új relációt tudunk előállítani. A változatok közé tartozik a Descartes-szorzat, a természetes összekapcsolás, a théta-összekapcsolás és a külső összekapcsolások. Az összekapcsolás eredménye lehet, hogy maga a válasz az adott lekérdezésre,

de – minthogy az összekapcsolás eredménye is reláció – az összekapcsoló kifejezések a select-from-where lekérdezés FROM záradékában előforduló alkérdésnek is tekinthetők. Az ilyen kifejezések leginkább a jóval összetettebb select-from-where kifejezéseket rövidítik (lásd 6.3.11. feladat).

Az összekapcsolási kifejezések legegyszerűbb formája a *keresztszorzat*; ez a kifejezés rokon értelmű a Descartes-szorzat vagy csak „szorzat" kifejezéssel, amelyet a 2.4.7. alfejezetben használtunk. Például, ha a következő két reláció szorzatát szeretnénk megkapni:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
```

a megfelelő utasítás:

```
Filmek CROSS JOIN SzerepelBenne;
```

és az eredmény egy kilencoszlopos reláció lesz, mégpedig a `Filmek` és a `SzerepelBenne` relációk attribútumaival. Minden egyes `Filmek` sort párosítunk minden egyes `SzerepelBenne` sorral, amelyek belekerülnek az eredményrelációba.

A szorzatreláció attribútumnevei lehetnek $R.A$ alakúak, ahol R az összeszorzott relációk közül az egyik, A pedig egy attribútuma. Ha A csak az egyik relációban található meg, akkor a névben az R és a pont elhagyható. (Ebben a példában az év attribútumon kívül a többiek esetében nincs szükség a reláció megjelölésére az attribútum nevében.)

A szorzatművelet használata önmagában azonban csak ritkán indokolt. Sokkal megfelelőbb a théta-összekapcsolás, amelyet az `ON` kulcsszóval lehet elérni. Az R és S relációnevek közé tesszük a `JOIN` kulcsszót, utánuk pedig az `ON` kulcsszót és egy feltételt. A `JOIN...ON` jelentése, hogy az $R \times S$ szorzatot az `ON` utáni feltétel szerinti kiválasztás követi.

6.23. példa. Tételezzük fel, hogy a következő két relációt szeretnénk összekapcsolni:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
```

azzal a feltétellel, hogy csak az ugyanarra a filmre vonatkozó sorok legyenek összekapcsolva. Azaz a cím és a gyártási év mindkét sorban meg kell egyezzen. Ezt az eredményt a következő lekérdezéssel érhetjük el:

```
Filmek JOIN SzerepelBenne ON
   filmcím = filmCím AND Filmek.év = SzerepelBenne.filmÉv;
```

Az eredmény ismét egy kilencoszlopos reláció a nyilvánvaló attribútumnevekkel. Most azonban egy `Filmek` sort csak akkor párosítunk össze egy `SzerepelBenne` sorral, ha a címre és az évre vonatkozó attribútumaik értéke megegyezik. Így

az oszlopok közül kettő felesleges lesz, mivel a `filmcím` és a `filmCím`, illetve a `Filmek.év` és a `SzerepelBenne.filmÉv` attribútumok értéke minden sorban ugyanaz lesz.

Ha zavar bennünket, hogy az előbbi példa eredményében két felesleges oszlop van, az egész kifejezést behelyezhetjük a `FROM` záradékba, és a megfelelő `SELECT` záradék segítségével eltüntethetjük a felesleges oszlopokat. Így a következő utasítás:

```
SELECT filmcím, Filmek.év, hossz, műfaj, stúdióNév,
    producerAzon, színészNév
FROM Filmek JOIN SzerepelBenne ON
    filmcím = filmCím AND Filmek.év = SzerepelBenne.filmÉv;
```

egy hétoszlopos relációt eredményez, mely a `Filmek` reláció sorait tartalmazza, kibővítve az összes lehetséges módon a filmben szereplő színészekkel. □

6.3.7. Természetes összekapcsolás

A 2.4.8. alfejezetben ismertettük, hogy miben különbözik a természetes összekapcsolás a théta-összekapcsolástól:

1. Az összekapcsolási feltétel a megegyező nevű attribútumok egyenlőségéből áll és nincs más feltétel.

2. Az egyenlővé tett attribútumok közül az egyiket kihagyjuk az eredményből.

Az SQL természetes összekapcsolása pontosan így működik. A `NATURAL JOIN` kulcsszavak jelennek meg az összekapcsolandó relációk között a $\bowtie$ művelet kifejezésére.

6.24. példa. Tételezzük fel, hogy a következő két reláció természetes összekapcsolását szeretnénk meghatározni:

```
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Az eredmény egy olyan reláció lesz, amely tartalmazza a `név` és a `cím` attribútumokat és az összes olyan attribútumot, amely az egyik vagy a másik relációban szerepel. A reláció sorai olyan személyekre fognak vonatkozni, akik színészek és gyártásirányítók is, és az összes lehetséges információt tartalmazni fogják: a címet, a nevet, a nemet, a születési dátumot, az azonosító számot és a nettó bevételt. Ezt a relációt a következő kifejezés írja le:

```
FilmSzínész NATURAL JOIN GyártásIrányító;
```

□

6.3.8. Külső összekapcsolások

A külső összekapcsolást, amely eredménye tartalmazza a nullértékekkel feltöltött lógó sorokat is, az 5.2.7. alfejezetben tárgyaltuk. Az SQL-ben használhatunk külső összekapcsolást; a nullérték jelölésére a `NULL` szolgál.

6.25. példa. Tételezzük fel, hogy a következő két reláció külső összekapcsolását szeretnénk meghatározni:

```
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Az SQL a szabványos külső összekapcsolást, amely a mindkét kapcsolandó táblából származó nem kapcsolható sorokat is tartalmazza, *teljes* külső összekapcsolásnak (full outerjoin) nevezi. A szintaxis:

```
FilmSzínész NATURAL FULL OUTER JOIN GyártásIrányító;
```

Az eredménye egy ugyanolyan hatoszlopos reláció, mint a 6.24. példa eredménye. A relációnak háromféle sora van. Léteznek olyan sorok, amelyek azokat a színészeket tartalmazzák, akik egyben gyártásirányítók is. Ezeknek a soroknak minden attribútuma kitölthető értéket tartalmaz. Ezek a sorok megtalálhatók a 6.24. példa eredményében is.

A második típusú sorok azokat a színészeket tartalmazzák, akik nem gyártásirányítók. Ezekben a sorokban a `név`, `cím`, `nem` és `születésiDátum` attribútumok értékét a `FilmSzínész` reláció alapján töltjük ki, míg a kizárólag a `GyártásIrányító`-hoz tartozó attribútumok – `azonosító` és `nettóBevétel` – `NULL` értéket tartalmaznak.

A harmadik típusú sorok azokat a gyártásirányítókat tartalmazzák, akik nem színészek. Ezen sorok `GyártásIrányító`-ból származó attribútumait a megfelelő `GyártásIrányító` sorok alapján töltjük ki, míg a `nem` és `születésiDátum` attribútumok, amelyek csak a `FilmSzínész` relációhoz tartoznak, `NULL` értéket tartalmaznak. A 6.12. ábrán látható három sor például megfelel a három sortípusnak. □

név	*cím*	*nem*	*születésiDátum*	*azonosító*	*nettóBevétel*
Mary Tyler Moore	Maple St.	'N'	9/9/99	12345	100··· $
Tom Hanks	Cherry Ln.	'F'	8/8/88	NULL	NULL
George Lucas	Oak Rd.	NULL	NULL	23456	200··· $

6.12. ábra. Három sor a `FilmSzínész` és `GyártásIrányító` külső összekapcsolásából

Az 5.2.7. alfejezetben a külső összekapcsolások összes fajtáit tárgyaltuk, ezek mind elérhetők az SQL-ben is. Ha bal vagy jobb oldali külső összekapcsolást kívánunk használni, akkor a megfelelő `LEFT` (bal) vagy `RIGHT` (jobb) szót kell használnunk a `FULL` helyett. Például a

```
FilmSzínész NATURAL LEFT OUTER JOIN GyártásIrányító;
```

összekapcsolás tartalmazza a 6.12. ábra első két sorát, de a harmadikat nem. Hasonlóképpen a

```
FilmSzínész NATURAL RIGHT OUTER JOIN GyártásIrányító;
```

tartalmazza a 6.12. ábra első és harmadik sorát, de a másodikat nem.

Tegyük most fel, hogy a természetes összekapcsolás helyett a külső théta-összekapcsolást (feltételes külső összekapcsolást) kívánjuk használni. Ekkor a `NATURAL` kulcsszó helyett az összekapcsolás után írhatjuk az `ON` kulcsszót, majd azt a feltételt, amelyet a sorok ki kell hogy elégítsenek. Ha megadjuk a `FULL OUTER JOIN` kulcsszavakat is, akkor az összepárosított sorokon kívül azok a sorok is bekerülnek az eredménybe, amelyek a másik relációból egy sorral sem kapcsolódtak össze, természetesen a hiányzó attribútumokban `NULL` értékkel.

6.26. példa. Tekintsük ismét a 6.23. példát, ahol a `Filmek` és a `SzerepelBenne` relációkat kapcsoltuk össze azzal a feltétellel, hogy a `filmcím` és `filmCím`, illetve az `év` és `filmÉv` attribútumok a két relációban megegyezzenek. Ha módosítjuk a feltételt egy külső összekapcsolás segítségével:

```
Filmek FULL OUTER JOIN SzerepelBenne ON
   filmcím = filmCím AND Filmek.év = SzerepelBenne.filmÉv;
```

akkor nemcsak azokat a filmeket kapjuk meg, amelyeknek legalább egy szereplője megtalálható a `SzerepelBenne` relációban, hanem azokat is, amelyekhez nem tartozik egy szereplő sem, a `filmCím`, `SzerepelBenne.filmÉv` és a `színészNév` attribútumokban `NULL` értékkel. Hasonlóképpen megkapjuk azokat a színészeket is, akik nem szerepelnek olyan filmben, amely a `Filmek` relációban megtalálható, és a `Filmek` reláció hat attribútuma ezekben a sorokban `NULL` értéket fog tartalmazni. □

A 6.26. példában bemutatott külső összekapcsolásokban a `FULL` kulcsszó kicserélhető a `LEFT` vagy a `RIGHT` kulcsszavakra. Például a

```
Filmek LEFT OUTER JOIN SzerepelBenne ON
   filmcím = filmCím AND Filmek.év = SzerepelBenne.filmÉv;
```

azokat a filmeket eredményezi, amelyeknek legalább egy felsorolt színészük van és azokat, amelyeknek nincs egy sem, de azok a színészek, akik egy nem említett filmben szerepelnek, nem lesznek az eredményben. Hasonlóképpen a

```
Filmek RIGHT OUTER JOIN SzerepelBenne ON
   filmcím = filmCím AND Filmek.év = SzerepelBenne.filmÉv;
```

utasítás kihagyja azokat a filmeket, amelyekhez nem tartozik egy színész sem, de tekintetbe veszi azokat a színészeket is (a filmekből származó attribútumokat `NULL` értékkel feltöltve), akik a vizsgált filmek egyikében sem szerepelnek.

6.3.9. Feladatok

6.3.1. feladat. Az 2.4.1. feladat adatbázissémáját használva adjuk meg a következő lekérdezéseket:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

A válaszokban használjunk legalább egy alkérdést, és minden lekérdezésre adjunk két lényegesen különböző megoldást (azaz használjunk különböző operátorokat az `EXISTS`, `IN`, `ALL` és `ANY` közül).

a) Keressük meg a legalább 3.0 sebességű PC-k gyártóit.

b) Keressük meg a legdrágább nyomtatókat.

! *c)* Keressük meg azokat a laptopokat, amelyek minden PC-nél lassúbbak.

! *d)* Keressük meg a modellszámát a legdrágább terméknek (PC, laptop vagy nyomtató).

! *e)* Keressük meg a legolcsóbb színes nyomtató gyártóját.

!! *f)* Keressük meg az olyan PC-k gyártóit, amelyek a leggyorsabbak a legkisebb memóriával rendelkező PC-k között.

6.3.2. feladat. Adjuk meg a következő lekérdezéseket a 2.4.3. feladat adatbázissémájára vonatkozóan:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

A lekérdezés tartalmazzon legalább egy alkérdést, és mindegyik feladathoz adjunk meg két megoldást (azaz különböző operátorokat használjunk az `EXISTS`, `IN`, `ALL` és `ANY` közül).

a) Keressük meg a legtöbb ágyúval rendelkező hajók országait.

! *b)* Keressük meg azokat a hajóosztályokat, amelyekből legalább egy hajót elsüllyesztettek egy csatában.

c) Keressük meg azoknak a hajóknak a neveit, melyek ágyúinak kalibere 16 hüvelyk.

d) Keressük meg azokat a csatákat, amelyekben a Kongó hajóosztályba tartozó hajók vettek részt.

!! *e)* Keressük meg azoknak a hajóknak a neveit, melyeknek az ágyúszáma a legnagyobb a velük megegyező kaliberű ágyúkat tartalmazó hajók között.

! **6.3.3. feladat.** Adjuk meg a 6.10. ábra lekérdezését alkérdések nélkül.

! **6.3.4. feladat.** Tekintsük a $\pi_L(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n)$ relációs algebrai kifejezést, ahol L csak az R_1 attribútumait tartalmazza. Mutassuk meg, hogy ezt a lekérdezést csak alkérdések felhasználásával át lehet írni SQL-be. Pontosabban, adjunk meg egy olyan SQL-kifejezést, amelyben mindegyik `FROM` záradékban csak egy sorváltozó található.

! **6.3.5. feladat.** Adjuk meg a következő lekérdezéseket metszet és különbség használata nélkül:

a) A 6.5. ábra lekérdezését.

b) A 6.17. példa lekérdezését.

!! **6.3.6. feladat.** Észrevehettük, hogy néhány SQL-operátor felesleges abban az értelemben, hogy más operátorokkal helyettesíthető. Például láthattuk, hogy s `IN` R helyettesíthető az s `= ANY` R kifejezéssel. Mutassuk meg, hogy az `EXISTS` és `NOT EXISTS` operátorok feleslegesek, azaz olyan kifejezéssel helyettesíthetőek, amelyben nincs `EXISTS`. *Tanács:* A `SELECT` záradékban konstans is használható.

6.3.7. feladat. Az aktuális adatbázisunk alábbi relációira nézve:

```
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

írjuk le azokat a sorokat, amelyek a következő SQL-kifejezések eredményében szerepelnek:

a) `Stúdió CROSS JOIN GyártásIrányító;`

b) `SzerepelBenne NATURAL FULL OUTER JOIN FilmSzínész;`

c) `SzerepelBenne FULL OUTER JOIN FilmSzínész ON név = színészNév;`

! **6.3.8. feladat.** Felhasználva a következő adatbázissémát, adjunk meg egy SQL-lekérdezést, amely információt szolgáltat minden termékről – PC-k, laptopok, nyomtatók – megadva a gyártót, ha van erre vonatkozó információ, és megadja az összes információt a termékről (ami a terméknek megfelelő relációban található).

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

6.3.9. feladat. A 2.4.3. feladatban szereplő adatbázis két relációját használva adjunk meg egy SQL-lekérdezést, amely minden megtalálható információt közöl a hajókról, beleértve a `Hajóosztályok` relációban is található információkat. Ha egy hajóosztályhoz nem tartoznak hajók a `Hajók` relációban, akkor erről az osztályról nem kell információkat szolgáltatni.

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
```

! **6.3.10. feladat.** Ismételjük meg a 6.3.9. feladatot, de vegyük az eredménybe azokat a hajókat is, amelyeknek a neve megegyezik a C osztály nevével, azokra a C osztályokra is, amelyekhez nem tartozik hajó a `Hajók` relációban. Feltehetjük, hogy létezik hajó az osztály nevével akkor is, ha nincs a `Hajók` relációban.

! **6.3.11. feladat.** Az ebben a fejezetben bemutatott összekapcsolási műveletek (a külső összekapcsolást kivéve) tulajdonképpen feleslegesek abban az értelemben, hogy mindegyik helyettesíthető select-from-where utasításokkal. Magyarázzuk meg, hogyan írhatók át a következő kifejezések select-from-where alakra:

a) `R CROSS JOIN S;`

b) `R NATURAL JOIN S;`

c) `R JOIN S ON` C`;`, ahol C egy SQL-feltétel.

6.4. Relációkra vonatkozó műveletek

Ebben az alfejezetben olyan műveletekkel foglalkozunk, melyek a relációk egészére vonatkoznak, nem pedig a relációk egyes (vagy néhány) soraira. Figyelembe kell vennünk azt a tényt, hogy az SQL a relációkat nem tekinti halmaznak, azaz ugyanaz a sor többször is előfordulhat. A 6.4.1. alfejezetben bemutatjuk, hogyan lehet az ismétlődéseket megszüntetni, míg a 6.4.2. alfejezetben ismertetjük, hogyan lehet az ismétlődések megőrzését biztosítani olyan esetekben, amikor az SQL-rendszer különben megszüntetné az ismétlődéseket.

Tárgyaljuk azt is, hogy az 5.2.4. alfejezetben bevezetett γ csoportképző és összesítő operátor funkcióit hogyan valósítja meg az SQL. Az SQL-nek vannak összesítő műveletei, és `GROUP-BY` csoportképző záradéka. Az SQL-ben van „`HAVING`" záradék is, amely nem egyedi sorokra, hanem a csoportokra vonatkozó válogatást teszi lehetővé.

6.4.1. Ismétlődések megszüntetése

Ahogyan azt a 6.3.4. alfejezetben is említettük, az SQL relációi különböznek a 2.2. alfejezetben absztrakt módon definiált relációtól. A reláció, halmazként tekintve, nem tartalmazhatja ugyanazt a sort többször. Amikor azonban az SQL-lekérdezés eredményeként létrejön egy reláció, akkor az SQL nem szünteti meg automatikusan az ismétlődéseket. Így az SQL egy lekérdezésre adott válasza tartalmazhatja ugyanazt a sort többször is.

Elevenítsük fel a 6.2.4. alfejezetben említetteket: az SQL select-from-where műveletének egyik értelmezése szerint a `FROM` záradékban szereplő relációk Descartes-szorzatát számoljuk ki először. A szorzat minden egyes sorára ellenőrizzük a `WHERE` feltételt, majd a megfelelő sorok a `SELECT` záradéknak megfelelően levetítve az eredménybe kerülnek. Ez a vetítés azt eredményezheti, hogy több különböző sorból ugyanaz az eredménysor keletkezik. Ezenkívül, mivel a `FROM` záradékban lévő SQL-relációk is tartalmazhatnak ismétlődéseket, ezeket az ismétlődéseket párosítva a többi reláció soraival újabb ismétlődéseket kaphatunk az eredményben.

Ha szeretnénk megszüntetni az ismétlődéseket az eredményben, a `SELECT` kulcsszó után a `DISTINCT` kulcsszót kell írnunk. Ez a kulcsszó jelzi az SQL-nek, hogy minden egyes sor csak egyszer szerepeljen az eredményben. Ez a lekérdezés eredményére alkalmazott – az 5.2.1. alfejezetben bemutatott – δ művelet SQL-beli megfelelője.

6.27. példa. Tekintsük a 6.9. ábra lekérdezését, melyben a Harrison Ford-filmek gyártásirányítóit kerestük, alkérdések nélkül. A lekérdezés jelenlegi formájában George Lucas többször is szerepelni fog az eredményben, mivel több olyan filmet is gyártott, amelyben Harrison Ford szerepelt. Ha azt szeretnénk, hogy minden gyártásirányító csak egyszer szerepeljen, a lekérdezés 1. sorát a következőre kell kicserélni:

```
1)  SELECT DISTINCT név
```

Így a gyártásirányítók listájából az eredmény kiírása előtt az ismétlődések kitörlődnek.

Valószínűleg a 6.7. ábra lekérdezése, amelyben alkérdést használtunk, nem fog az eredményben ismétlődéseket tartalmazni. Igaz ugyan, hogy a 4. sor lekérdezése többször is tartalmazhatja George Lucas azonosítószámát, de az 1. sor „fő" lekérdezésében minden `GyártásIrányító` sort csak egyszer vizsgálunk. Ellvileg ebben a relációban George Lucas csak egyszer szerepel, és ez az egyedüli sor, amely a 3. sor `WHERE` feltételének eleget tesz. Így George Lucas csak egyszer fog szerepelni az eredményben. □

6.4.2. Ismétlődések kezelése halmazműveletek során

A `SELECT` utasítással ellentétben, amely csak akkor szünteti meg az ismétlődéseket, ha jelen van a `DISTINCT` kulcsszó, a 6.2.5. alfejezetben bevezetett egyesítés,

Az ismétlődések megszüntetésének költsége

Hajlamosak lennénk minden SELECT után a DISTINCT kulcsszót írni, mivel hiba nem keletkezhet belőle. Valójában azonban nagyon költséges az ismétlődések megszüntetése egy relációban. A relációt sorba kell rendezni vagy szét kell osztani, hogy a megegyező sorok egymás mellé kerüljenek. Csak a sorok ilyen jellegű csoportosításán keresztül tudjuk azt eldönteni, hogy egy adott sor többször szerepel-e. Sokszor az az idő, amíg a relációt rendezzük, hosszabb, mint magának a lekérdezésnek a végrehajtása. Ha azt kívánjuk, hogy a lekérdezéseink gyorsak legyenek, akkor az ismétlődések megszüntetését csak indokolt esetben használjuk.

metszet és különbség operátorok normális esetben megszüntetik az ismétlődéseket. Tehát a multihalmazok halmazzá konvertálódnak és a halmazműveletek halmaz jellegű változatait alkalmazzák. Azért, hogy az ismétlődések megmaradjanak, az UNION, EXCEPT vagy INTERSECT kulcsszavak után az ALL kulcsszót kell írni. Ebben az esetben ezek a műveletek az 5.1.2. alfejezetben tárgyalt multihalmaz-szemantika szerint fognak működni.

6.28. példa. Tekintsük ismét a 6.18. példa lekérdezését, de adjuk hozzá az ALL kulcsszót:

```
(SELECT filmcím, év FROM Filmek)
    UNION ALL
(SELECT filmCím AS filmcím, filmÉv AS év FROM SzerepelBenne);
```

Így minden egyes filmcím és gyártási év annyiszor fog szerepelni, ahányszor a Filmek és a SzerepelBenne relációkban összesen megjelenik. Például, ha egy film egyszer található meg a Filmek relációban és három színész tartozik hozzá a SzerepelBenne relációban, akkor az egyesítés eredményében a film négyszer fog szerepelni. □

Hasonlóan az egyesítéshez, az INTERSECT ALL és az EXCEPT ALL műveletek is multihalmazok felett dolgoznak. Tehát, ha R és S két reláció, akkor az

$$R \texttt{ INTERSECT ALL } S$$

kifejezés eredménye az a reláció, amelyben egy t sor előfordulásainak száma egyenlő a t sor R-beli és S-beli előfordulásai számának a minimumával. Továbbá, az

$$R \texttt{ EXCEPT ALL } S$$

kifejezés eredménye az a reláció, amelyben egy t sor előfordulásainak száma egyenlő a t sor R-beli előfordulásainak számából kivonva a t sor S-beli előfordulásainak számát, ha az eredmény pozitív. Ezek a definíciók megegyeznek az 5.1.2. alfejezetben tárgyalt definíciókkal.

6.4.3. Csoportosítás és összesítések az SQL-ben

Az 5.2.4. alfejezetben tárgyaltuk a kiterjesztett relációs algebra γ csoportosító és összesítő műveletét. Idézzük fel, hogy – amint az 5.2.3. alfejezetben láttuk –, ez a művelet a sorok egy vagy több attribútumának értéke szerint lehetővé teszi a reláció sorainak csoportosítását. Lehetőségünk van a reláció oszlopaira összesítő műveletek alkalmazásával különböző összesítések előállítására. Ha csoportokat képeztünk, akkor az összesítések csoportonként működnek. Az SQL a `SELECT` záradék listájában megengedi összesítő függvények használatát, a csoportosítást pedig a `GROUP BY` záradékkal oldja meg, így biztosítja a γ művelet összes lehetőségét.

6.4.4. Összesítő függvények

Az SQL öt összesítő függvényével az 5.2.2. alfejezetben találkoztunk, ezek a `SUM`, `AVG`, `MIN`, `MAX` és `COUNT`. Ezek tipikusan a `SELECT` záradékban egy-egy oszlopra alkalmazva skalár értékeket szolgáltatnak. Kivétel a `COUNT(*)` kifejezés, amely a `FROM`-ban megadott reláció(k)ból a `WHERE`-ben leírt feltétel(ek)nek megfelelően megkonstruált összes sor számát adja meg.

Az eddigieken felül a `DISTINCT` kulcsszó alkalmazásával lehetőségünk van arra, hogy az összesítések előállítása előtt az érintett oszlopból a duplikátumokat kihagyjuk. Tehát olyan kifejezéseket használhatunk, mint például a `COUNT(DISTINCT x)`, amely az x oszlopban található különböző értékek számát adja meg. A `COUNT` helyett bármelyik függvényt használhatjuk, de például a `SUM(DISTINCT x)` csak ritkán ad hasznos választ, ez ugyanis például az x oszlopban található különböző értékek összegét adja meg.

6.29. példa. A következő lekérdezés megadja az összes gyártásirányító átlagos nettó bevételét:

```
SELECT AVG(nettóBevétel)
FROM GyártásIrányító;
```

Figyeljük meg, hogy nincs feltétel a sorokra, így a `WHERE` záradékot el lehet hagyni. A lekérdezés a következő reláció `nettóBevétel` oszlopát vizsgálja:

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Összeadja az oszlopban található értékeket, mindegyik sor esetén egy értéket (akkor is, ha a sor egy másik sor ismétlődése), majd az összeget elosztja a sorok számával. Ha nincsenek ismétlődések, a lekérdezés az átlagos nettó bevételt eredményezi, ahogy szerettük volna. Ha vannak ismétlődések, akkor annak a gyártásirányítónak a nettó bevételét, aki n-szer szerepel a relációban, n-szer fogja az összeghez hozzáadni. □

6.30. példa. A következő lekérdezés a SzerepelBenne reláció sorainak számát határozza meg:

```
SELECT COUNT(*)
FROM SzerepelBenne;
```

Az előbbihez hasonló lekérdezés:

```
SELECT COUNT(SzínészNév)
FROM SzerepelBenne;
```

a reláció SzínészNév oszlopában előforduló összes név számát adja meg. Mivel a duplikátumokat nem zártuk ki, így a két fenti kérdés válasza egymással megegyezik.

Ha a duplikátumokat nem óhajtjuk többszörösen figyelembe venni, akkor az összegzett attribútum neve előtt a kérdésben a DISTINCT kulcsszót is használnunk kell, tehát:

```
SELECT COUNT(DISTINCT SzínészNév)
FROM SzerepelBenne;
```

Így minden színészt csak egyszer veszünk figyelembe, függetlenül attól, hogy hány filmben szerepelt. □

6.4.5. Csoportosítás

A sorok csoportosítását a WHERE záradékot követő GROUP BY záradékban tudjuk megadni. A GROUP BY kulcsszót a *csoportosító* attribútumok listája követi. A legegyszerűbb esetben a FROM záradék csak egy sorváltozót tartalmaz, és a reláció sorait csoportosítjuk a csoportosító attribútumoknak megfelelően. A SELECT záradékban szereplő összesítési operátorokat a csoportokra kell alkalmazni.

6.31. példa. A Filmek relációból:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

szeretnénk megtudni stúdiónként a gyártott filmek hosszának percben mért összegét. A választ a következő lekérdezés szolgáltatja:

```
SELECT stúdióNév, SUM(hossz)
FROM Filmek
GROUP BY stúdióNév;
```

A lekérdezés értelmezése úgy képzelhető el, hogy a Filmek reláció sorait az 5.4. ábrán érzékeltetett módon átszervezzük és csoportosítjuk úgy, hogy az összes sor, amely a Disney stúdióra vonatkozik, egymás mellé kerüljön, ugyanígy egymás mellé kerülnek az MGM-re vonatkozó sorok stb. A hossz komponensek összegét csoportonként kiszámítjuk és minden csoportra az eredménybe kerül a stúdió neve és a hozzá tartozó összeg. □

Figyeljük meg, hogy a 6.31. példában a SELECT záradékban kétféle elem található:

1. Összesítések, amelyekben egy összesítési operátort alkalmazunk egy attribútumra vagy egy attribútumot tartalmazó kifejezésre. Ezek a kifejezések csoportonként kerülnek kiértékelésre.

2. Attribútumok, amelyek a GROUP BY záradékban szerepelnek, mint a példában stúdióNév. Egy összesítéseket tartalmazó SELECT záradékban csak a GROUP BY záradékban is megtalálható attribútumok jelenhetnek meg összesítési operátor nélkül.

A GROUP BY záradékot tartalmazó lekérdezésekben általában csoportosító attribútumok és összesítések is vannak a SELECT záradékban, de elvileg nincs akadálya olyan lekérdezést írni, amelyben valamelyik ezek közül hiányzik. Például a következő lekérdezés is helyes:

```
SELECT stúdióNév
FROM Filmek
GROUP BY stúdióNév;
```

Ez a lekérdezés csoportosítja a sorokat stúdióNév szerint, majd minden csoport esetén kiírja a stúdióNév értékét. Azaz a lekérdezés ugyanazt eredményezi, mint a következő:

```
SELECT DISTINCT stúdióNév
FROM Filmek;
```

A GROUP BY záradékot többrelációs lekérdezésben is használhatjuk. Egy ilyen lekérdezés kiértékelése a következő módon történik:

1. A FROM és WHERE záradékokból kialakul egy R reláció. Az R relációt úgy kapjuk, hogy a FROM záradékban lévő relációk Descartes-szorzatára alkalmazzuk a WHERE feltételt.

2. R sorait csoportosítjuk a GROUP BY attribútumainak megfelelően.

3. Eredményként a SELECT záradék attribútumai és összesítései jelennek meg csoportonként úgy, mintha a lekérdezés az R relációra történt volna.

6.32. példa. Tételezzük fel, hogy szeretnénk meghatározni mindegyik gyártásirányító által gyártott filmek összhosszát. A következő két relációra van szükségünk:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Tekintsük a théta-összekapcsolásukat, egyenlővé téve az azonosító számokat. Ez a lépés egy olyan relációt hoz létre, amelyben minden `GyártásIrányító` sort összepárosítja az összes olyan `Filmek` sorral, amelyet az a gyártásirányító készített. Megjegyezzük, hogy az olyan gyártásirányító, aki nem producer, nem lesz párosítva egyetlen filmmel sem, következésképpen nem fordul majd elő a relációban. Ezután a relációt csoportosíthatjuk a gyártásirányító azonosító száma szerint, és minden csoportban meghatározhatjuk a filmek hosszainak összegét. A lekérdezést a 6.13. ábra szemlélteti. □

```
SELECT név, SUM(hossz)
FROM GyártásIrányító, Filmek
WHERE producerAzon = azonosító
GROUP BY név;
```

6.13. ábra. Gyártásirányítónként a filmek hosszainak összege

6.4.6. Csoportosítás, összegzés és nullértékek

Ha a sorban nullérték is előfordul, akkor van néhány szabály, amire gondolnunk kell:

- A `NULL` értéket bármely összesítés során figyelmen kívül hagyjuk. Nem vesz részt az oszlopra vonatkozó összeg, az átlag és a darabszám képzésében, és nem befolyásolja a maximum- és minimumértékeket. Például a `COUNT(*)` a relációban a sorokat számlálja össze, de a `COUNT(A)` csak az olyan sorokat veszi fegyelembe, melyekben az A attribútum nem `NULL` értékű.
- Más szempontból, ha egy csoportosító attribútum értéke `NULL`, akkor közönséges értéknek tekintjük. Ez azt jelenti, hogy a csoportosító attribútum `NULL` értékeihez is létrejön az ilyen sorokból álló csoport.
- Az összesítő függvények az üres csoportokra `NULL` értéket adnak eredményül, kivéve a `COUNT`, mely üres csoportra 0 eredmény ad.

6.33. példa. Tegyük fel, hogy az $R(A, B)$ relációnak egyetlen sora van, és a sor minden komponense nullértékű, azaz:

A	B
NULL	NULL

Ekkor a:

```
SELECT A, COUNT(B)
FROM R
GROUP BY A;
```

A záradékok sorrendje az SQL-lekérdezésekben

Most már megismertük mind a hat záradékot, amelyek az SQL „select-from-where" lekérdezésében szerepelhetnek: SELECT, FROM, WHERE, GROUP BY, HAVING és ORDER BY. Csak a SELECT és FROM használata kötelező. Ha a többi közül bármelyik szerepel, akkor a fenti sorrendet be kell tartani.

lekérdezés eredménye a (NULL, 0) sor lesz. Azért ez az eredmény, mert amikor A szerint csoportosítottunk, akkor egyetlen, NULL értékekből álló csoportot tudtunk képezni. Ez a csoport egy sorból áll, melyben B értéke NULL. Amikor pedig megszámoltuk a csoport nem NULL értékű sorainak számát, azt 0-nak találtuk.

Ugyanakkor a

```
SELECT A, SUM(B)
FROM R
GROUP BY A;
```

lekérdezés eredménye a (NULL, NULL) sor lesz. A magyarázat a következő: A NULL értékhez tartozó csoport az R reláció egyetlen sorából áll. Amikor e csoportban a B értékeket adjuk össze, akkor egyetlen NULL-t találunk, amelyet nem veszünk figyelembe az összeg képzésekor. Így egy üres halmaz értékeinek összegét képezzük, amely definíció szerint NULL. □

6.4.7. HAVING záradék

Tegyük fel, hogy a 6.32. példában nem szeretnénk mindegyik gyártásirányítót figyelembe venni. A sorokra olyan feltételt tehetnénk előzőleg, hogy a csoportosítás során egyes csoportok üresek legyenek. Például, ha olyan gyártásirányítókat szeretnénk vizsgálni, akiknek a nettó bevétele nagyobb, mint 10 000 000 $, akkor a 6.13. ábra harmadik sorát kicserélhetnénk a következőre:

```
WHERE producerAzon = azonosító AND nettóBevétel >= 10000000
```

Néha azonban a csoportokat a csoport bizonyos összesített tulajdonsága alapján szeretnénk kiválogatni. Ilyenkor a GROUP BY záradék után a HAVING záradékot írhatjuk. A záradék a HAVING kulcsszóból és egy feltételből áll, amely a csoportra vonatkozik.

6.34. példa. Keressük meg azokat a gyártásirányítókat és filmhosszaik összegét, akik legalább egy 1930 előtti filmet is készítettek. A 6.13. ábrához csatolhatjuk a következő sort:

```
HAVING MIN(év) < 1930
```

Az így kapott kérdés a 6.14. ábrán látható. A lekérdezés csak azokat a csoportokat veszi majd figyelembe, melyekben legalább egy 1930 előtti film található. □

```
SELECT név, SUM(hossz)
FROM GyártásIrányító, Filmek
WHERE producerAzon = azonosító
GROUP BY név
HAVING MIN(év) < 1930;
```

6.14. ábra. A korai gyártásirányítók filmhosszainak összege

Néhány szabály, amelyekre gondolnunk kell a HAVING záradék alkalmazásakor:

- A HAVING záradékban hivatkozott összesítés csak az éppen feldolgozott csoport soraira vonatkozik.
- A FROM záradékban megadott relációk bármely attribútumára képezhetünk a HAVING záradékban összesítést, összesítés nélkül a HAVING záradékban csak azok az attribútumok fordulhatnak elő, amelyek a GROUP BY listában is szerepeltek. (Ugyanaz a szabály, mint ami a SELECT záradékra is vonatkozott.)

6.4.8. Feladatok

6.4.1. feladat. Adjuk meg a 2.4.1. feladatban szereplő lekérdezéseket SQL-ben, biztosítva az ismétlődések megszüntetését.

6.4.2. feladat. Adjuk meg a 2.4.3. feladatban szereplő lekérdezéseket SQL-ben, biztosítva az ismétlődések megszüntetését.

! **6.4.3. feladat.** A 6.3.1. feladat mindegyik lekérdezésénél döntsük el, hogy az eredmény tartalmazhat-e ismétlődéseket, és ha igen, írjuk át a lekérdezést úgy, hogy ismétlődések ne szerepeljenek benne. Ha nem tartalmaznak ismétlődéseket, akkor adjunk meg egy alkérdések nélküli olyan lekérdezést, amely ugyanazt az ismétlődésmentes eredményt adja.

! **6.4.4. feladat.** Ugyanaz a feladat, mint a 6.4.3. feladatban, ezúttal azonban a 6.3.2. feladat lekérdezéseire vonatkozóan.

! **6.4.5. feladat.** A 6.27. példában említettük, hogy a „keressük meg Harrison Ford filmjeinek gyártásirányítóit" feladatot megvalósító különböző lekérdezések különböző válaszokat adhatnak. Ezek lehetnek multihalmazok is annak ellenére, hogy ugyanazon választ adják. Vizsgáljuk meg a 6.22. példában szereplő kérdést, melynek a FROM záradékában alkérdést használtunk. Ez a verzió eredményezhet-e duplikátumokat, és ha igen, miért?

6.4.6. feladat. Adjuk meg a következő lekérdezéseket, a 2.4.1. feladat alábbi adatbázisára vonatkozóan:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

Szemléltessük az eredményt a 2.4.1. feladat adataival.

a) Keressük meg a PC-k átlagos sebességét.

b) Keressük meg az 1000 $-nál drágább laptopok átlagos sebességét.

c) Keressük meg az „A" gyártó által gyártott PC-k átlagos árát.

! *d)* Keressük meg a „D" gyártó által gyártott PC-k és laptopok átlagos árát.

e) Keressük meg minden egyes PC sebességéhez az ilyen sebességű PC-k átlagos árát.

! *f)* Keressük meg minden gyártó esetén a laptopok átlagos képernyőméretét.

! *g)* Keressük meg azokat a gyártókat, akik legalább háromfajta PC-t gyártanak.

! *h)* Keressük meg minden gyártó esetén a maximális PC-árat.

! *i)* Keressük meg a 2.0-nál nagyobb sebességű PC-k átlagos árát.

!! *j)* Keressük meg minden olyan gyártóhoz, akik nyomtatót gyártanak, a PC-k átlagos merevlemezméretét.

6.4.7. feladat. Adjuk meg a következő SQL-lekérdezéseket, amelyek a 2.4.3. feladat adatbázissémájára vonatkoznak:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

és a lekérdezések eredményeit szemléltessük az 2.4.3. példa adataival.

a) Keressük meg a hajóosztályok számát.

b) Keressük meg a hajóosztályok átlagos ágyúszámát.

! *c)* Keressük meg a hajók átlagos ágyúszámát. Figyeljük meg a különbséget *b)* és *c)* között: melyik esetben súlyoztuk a hajóosztályokat a hajóik számával?

! *d)* Keressük meg minden hajóosztály esetén azt az évet, amikor az ebbe az osztályba tartozó első hajót felavatták.

! *e)* Keressük meg minden hajóosztály esetén a csatában elsüllyesztett hajók számát.

!! *f)* Keressük meg a legalább három hajóból álló osztályokra a csatában elsüllyesztett hajók számát.

!! *g)* Egy ágyú által kilőtt golyó súlya (fontokban) körülbelül akkora, mint a kaliber (hüvelykben) köbének a fele. Keressük meg minden országra a hozzá tartozó hajók ágyúgolyóinak átlagos súlyát.

6.4.8. feladat. Az 5.10. példában feltettünk egy γ formában megfogalmazott kérdést. („Keressük meg azokat a színészeket, akik első filmszerepük évében legalább három filmben szerepeltek!") Adjuk meg ezt a lekérdezést SQL-ben.

! **6.4.9. feladat.** A kibővített relációs algebra γ operátorának nincsen az `SQL HAVING`-jének megfelelő lehetősége. Utánozható-e a kiterjesztett relációs algebrában a `HAVING` záradékkal megfogalmazott SQL-lekérdezés? Ha igen, akkor milyen általános szabály szerint?

6.5. Változtatások az adatbázisban

Eddig a pontig a select-from-where lekérdező SQL-utasításra koncentráltunk. Léteznek további SQL-utasítások is, amelyek nem egy eredményt adnak vissza, hanem megváltoztatják az adatbázis állapotát. Ebben a fejezetben három utasítástípust ismertetünk, amelyek a következő hatással járnak:

1. Sorok beszúrása egy relációba.
2. Bizonyos sorok törlése egy relációból.
3. Bizonyos létező sorok meghatározott komponensei értékeinek módosítása.

Az ilyen típusú műveleteket általában *módosítások*nak nevezzük.

6.5.1. Beszúrás

A beszúrási művelet legegyszerűbb alakja:

`INSERT INTO` $R(A_1, \ldots, A_n)$ `VALUES` $(v_1, \ldots, v_n)$;

A sor úgy keletkezik, hogy az A_i attribútumhoz v_i értéket rendelünk, $i = 1, 2, \ldots, n$. Ha az attribútumlista nem tartalmazza R összes attribútumát, akkor a hiányzó attribútumok az alapértelmezés szerinti értéket kapják.

6.35. példa. Tételezzük fel, hogy Kate Winsletet szeretnénk a *Titanic* szereplőinek listájába felvenni. A megfelelő utasítás:

```
1) INSERT INTO SzerepelBenne(filmCím, filmÉv, színészNév)
2) VALUES('Titanic', 1998, 'Kate Winslet');
```

Az utasítás hatása, hogy egy sort, amelynek három komponense a 2. sorban található, beszúrunk a `SzerepelBenne` relációba. Mivel a `SzerepelBenne` reláció minden attribútumát felsoroltuk az 1. sorban, nincs szükség alapértelmezett értékekre. A 2. sorban található értékeket összepárosítjuk az 1. sorban felsorolt megfelelő attribútumokkal, így például a `filmCím` értéke `'Titanic'` lesz stb. □

Ha a 6.35. példához hasonlóan a reláció minden attribútumának megadjuk az új értékét, az attribútumlistát elhagyhatjuk. Azaz az utasítás így alakul:

```
INSERT INTO SzerepelBenne
VALUES('Titanic', 1998, 'Kate Winslet');
```

Ebben az esetben azonban nagyon kell vigyáznunk arra, hogy az értékek sorrendje megegyezzen az attribútumok relációbeli sorrendjével.

- Ha nem vagyunk biztosak az attribútumok sorrendjében, jobb, ha felsoroljuk azokat az `INSERT` záradékban, abban a sorrendben, ahogy a `VALUES` záradékban megadtuk az értékeiket.

Az előbb ismertetett legegyszerűbb `INSERT` utasítás csak egy sort szúr be a relációba. A sorhoz megadott értéklista helyett egy alkérdés segítségével meghatározhatunk több beszúrandó sort is. Ez az alkérdés helyettesíti a `VALUES` kulcsszót és az `INSERT` utáni sorkifejezést.

6.36. példa. Tételezzük fel, hogy a következő relációba be szeretnénk szúrni a `Filmek` relációban megtalálható összes olyan stúdiót, melyek azonban a `Stúdió` relációban nem szerepelnek:

```
Stúdió(név, cím, elnökAzon)
```

A `Filmek` reláció sémája:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

Mivel nem tudjuk ezen stúdiók címét és elnökének azonosítóját, ezek az attribútumok `NULL` értéket kapnak. Az utasítást a 6.15. ábra szemlélteti.

Mint a legtöbb SQL-utasítást, a 6.15. ábra utasítását is legkönnyebb belülről kifelé értelmezni. Az 5–6. sorok az összes `Stúdió` relációbeli stúdiónevet eredményezik. Ezt használva a 4. sor azt ellenőrzi, hogy a `Filmek` sor stúdióneve szerepel-e ezen stúdiók között.

```
1) INSERT INTO Stúdió(név)
2)     SELECT DISTINCT stúdióNév
3)     FROM Filmek
4)     WHERE stúdióNév NOT IN
5)         (SELECT név
6)          FROM Stúdió);
```

6.15. ábra. Új stúdiók beszúrása

A beszúrások időzítése

Az SQL-szabvány előírja, hogy a lekérdezést teljes egészében ki kell értékelni, mielőtt bármely sort beszúrnánk. A 6.15. ábra példájában a 2–6. sorok közötti lekérdezést az 1. sor beszúrása előtt kellene elvégezni. Így az 1. sorban beszúrt új stúdiók nem lehetnek hatással a 4. sor feltételére.

Ebben a példában nincs jelentősége, hogy a beszúrások késleltetve vannak-e addig, amíg a lekérdezés teljes kiértékelése megtörténik. Tegyük fel azonban azt, hogy a 6.15. ábra 2. sorából kivesszük a `DISTINCT` kulcsszót. Ha a 2–6. sorok közötti lekérdezést teljesen kiértékeljük a beszúrások előtt, akkor egy új stúdiónév, amely többször szerepel a `Filmek` relációban, az eredményben is többször szerepelne, tehát a `Stúdió` relációba is többször kerülne be. Viszont ha minden egyes megtalált stúdiót – a szabványtól eltérően – rögtön beszúrná az adatbázis-kezelő rendszer, akkor egyik sem lenne többször beszúrva a `Stúdió` relációba. Ennek az az oka, hogy mivel az új stúdiónevet már beszúrtuk, a név nem fogja kielégíteni a 4–6. sorok közötti feltételt, tehát a 2–6. sorok közötti lekérdezés nem fogja ismét eredményként jelezni.

Tehát a 2–6. sorok azokat a stúdióneveket eredményezik, amelyek a `Filmek`-ben benne vannak, de a `Stúdió`-ban nincsenek benne. A 2. sorban a `DISTINCT` azt biztosítja, hogy ebben a halmazban mindegyik stúdió csak egyszer szerepel függetlenül attól, hogy hány filmet gyártottak. Végül az 1. sor beszúrja ezeket a stúdiókat, `NULL` értéket adva a `cím` és `elnökAzon` komponenseknek. □

6.5.2. Törlés

A törlési utasítás alakja:

`DELETE FROM` *R* `WHERE` <feltétel>;

Az utasítás azt eredményezi, hogy az *R* relációból kitörlődik minden olyan sor, amely megfelel a feltételnek.

6.37. példa. A következő relációból:

```
SzerepelBenne(filmCím, filmÉv, színészNév)
```

a következő SQL-utasítás segítségével kitörölhetjük azt a tényt, hogy Kate Winslet játszott a *Titanic*ban:

```
DELETE FROM SzerepelBenne
WHERE filmCím = 'Titanic' AND
      év = 1998 AND
      színészNév = 'Kate Winslet';
```

Figyeljük meg, hogy a 6.35. példa beszúrási műveletével ellentétben nem tudjuk könnyen megadni a törölni kívánt sort. A `WHERE` feltételben kell egészen pontosan megadni, hogy melyik sort akarjuk törölni. □

6.38. példa. Egy újabb példa a törlésre. Ezúttal töröljünk ki a következő relációból

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

több olyan sort egyszerre, amelyek megfelelnek egy bizonyos feltételnek:

```
DELETE FROM GyártásIrányító
WHERE nettóBevétel < 10000000;
```

Ezzel az utasítással kitöröljük a relációból a kis bevételű – tízmillió dollár alatti – gyártásirányítókat. □

6.5.3. Módosítás

A beszúrást és a törlést tulajdonképpen tekinthetjük módosításoknak, de valójában a *módosítás* az SQL-ben az adatbázis egy speciális változtatása, melyben egy vagy több létező sor bizonyos komponenseinek értékét megváltoztatjuk. A módosítás általános formája a következő:

`UPDATE` R `SET` <új értékadások> `WHERE` <feltétel>;

Mindegyik új értékadás egy attribútumból, az egyenlőségjelből és egy kifejezésből áll. Ha több mint egy értékadás van, vesszővel kell azokat elválasztani. Az utasítás eredményeképpen az összes olyan R-beli sorban, amelyek megfelelnek a feltételnek, az értékadáslistának megfelelően a komponensek módosulnak.

6.39. példa. Módosítsuk a következő relációt:

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

a név elé téve az 'Ig.' rövidítést minden olyan gyártásirányító esetén, aki elnöke (igazgatója) egy stúdiónak. A feltétel az, hogy az azonosító szerepeljen a Stúdió reláció valamelyik sorának az elnökAzon komponensében. A megfelelő utasítás:

```
1) UPDATE GyártásIrányító
2) SET név = 'Ig.' || név
3) WHERE azonosító IN (SELECT elnökAzon FROM Stúdió);
```

A 3. sor azt ellenőrzi, hogy a gyártásIrányító azonosítója megegyezik-e a Stúdió reláció valamelyik sorának elnökAzon komponensével.

A 2. sor végrehajtja a módosítást a megfelelő sorokon. Emlékezzünk vissza, hogy a || operátor karaktersorok összeillesztését jelenti, tehát az = utáni kifejezés az 'Ig.' karaktersort helyezi a név komponens régi értéke elé. Az így kapott karaktersor lesz a név komponens új értéke. □

6.5.4. Feladatok

6.5.1. feladat. Adjuk meg a következő adatbázis-módosításokat a 2.4.1. feladat adatbázissémájára vonatkozóan. Írjuk le ezen feladat adatmódosításainak hatását.

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

a) Két INSERT utasítás segítségével tároljuk az adatbázisban azt a tényt, hogy az 1100-as PC-modellt a C gyártó gyártja, 3.2 a sebessége, a memóriája 1024, merevlemeze 180 és az ára 2499 $.

! *b)* Szúrjuk be az adatbázisba azt, hogy minden egyes PC esetén létezik egy vele megegyező gyártójú, sebességű, memóriájú, merevlemezű laptop, amelynek 17 hüvelykes képernyője van, modellszáma 1100-zal nagyobb és 500 $-ral drágább.

c) Töröljük ki a 100 gigabájtnál kisebb merevlemezű PC-ket.

d) Töröljük ki az összes olyan laptopot, amelyeket olyan cég gyárt, amely nem gyárt nyomtatókat.

e) Az A cég megveszi a B céget. Módosítsuk a B által gyártott termékeket olyan módon, hogy az A gyártja őket.

f) Minden egyes PC esetén kétszerezzük meg a memória nagyságát, és adjunk 60 gigabájtot a merevlemez méretéhez. (Ne feledjük, hogy egyetlen UPDATE utasítás segítségével több attribútum is módosítható.)

! *g)* A B cég által gyártott laptopok esetén adjunk egy hüvelyket a képernyő méretéhez és vonjunk ki 100 $-t az árból.

6.5.2. feladat. Adjuk meg a következő adatbázis-módosításokat a 2.4.3. feladat adatbázissémájára vonatkozóan. Írjuk le ezen feladat adatmódosításainak hatását.

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

a) A Nelson osztályba tartozó két brit csatahajót – a Nelsont és a Rodney-t – 1927-ben avatták fel, mindkettőnek 16 hüvelykes kaliberű ágyúja van, és 34 000 tonna a vízkiszorításuk. Szúrjuk be ezeket az adatokat az adatbázisba.

b) A Vittorio Veneto hajóosztályba tartozó három olasz csatahajóból kettőt – a Vittorio Venetót és az Italiát – 1940-ben avattak fel; a harmadik hajót, a Rómát pedig 1942-ben. Mind a háromnak 9 db 15 hüvelykes kaliberű ágyúja van és vízkiszorításuk 41 000 tonna. Szúrjuk be ezeket az adatokat az adatbázisba.

c) Töröljük ki a `Hajók` relációból az összes elsüllyesztett hajót.

d) Módosítsuk a `Hajóosztályok` relációt úgy, hogy a kalibert centiméterben (1 hüvelyk = 2,5 centiméter) és vízkiszorítást metrikus tonnában (1 metrikus tonna = 1,1 tonna) adjuk meg.

e) Töröljük ki a háromnál kevesebb hajóból álló osztályokat.

6.6. Tranzakciók SQL-ben

Az eddigiekben feltételeztük, hogy az adatbázishoz egyidejűleg egy felhasználó fér hozzá, ezért a program által elvégzett adatbázis-műveletek egymás után realizálódnak: az egyik művelet eredményeként létrejött adatbázis-állapot szolgál a következő adatbázis-művelet kiindulási állapotaként. Továbbá feltételeztük, hogy az összes művelet teljes egészében (atomosan) lefut: sem szoftver-, sem pedig hardverhibák nem idézhetnek elő olyan helyzeteket, amelyekben az adatbázis állapota egy félbemaradt művelet eredménye.

A valós alkalmazásoknál sokkal bonyolultabb a helyzet. Ebben a részben először megvizsgáljuk, hogy milyen problémák származhatnak abból, hogy az adatbázis állapota nem a rajta végrehajtott műveletek eredményét tükrözi, majd megvizsgáljuk azt, hogy az SQL milyen eszközöket ad az ilyen és ehhez hasonló problémák elkerülésére.

6.6.1. Sorbarendezhetőség

Valós webes, banki vagy például repülőgép-helyfoglalási alkalmazásoknál az adatbázist egyidejűleg akár több száz művelet is elérheti vagy módosíthatja. E műveleteket sok ezer, akár millió felhasználó kezdeményezheti (például banki alkalmazottak, bankjegykiadó automaták, repülőjegy-irodák stb.), és nem zárható ki, hogy két felhasználó egyidejűleg ugyanazt az adatelemet akarja elérni (például két felhasználó ugyanarra a számlára akar pénzt átutalni, vagy ugyanarra a repülőjáratra, vagy akár ugyanarra a meccsre akar jegyet váltani), és ilyenkor meglepő konfliktushelyzetek alakulhatnak ki.

Egy példán keresztül mutatjuk be, milyen hibák származhatnának abból, ha az adatbázis-kezelő rendszerek egyáltalán nem lennének korlátozva az egyes adatbázis-műveletek végrehajtási sorrendjében. Ezen felül az adatbázisrendszerek nem tudják irányítani azt, hogy mennyi ideig várakozzanak arra, hogy a felhasználó meghozzon egy döntést. Az adatbázisrendszernek csak az SQL-utasítások végrehajtási sorrendjének befolyásolására van lehetősége.

6.40. példa. Megszokott, hogy a légitársaság webes lehetőséget biztosít arra, hogy utasai a repülőre ülőhelyet válasszanak. Ez a szolgáltatás megmutatja az ülések térképét, jelezve a szabad helyeket. Az üléstérkép előállításához az adatok a légitársaság adatbázisából származnak. Az adatokat tartalmazó reláció lehet például:

```
Légijáratok(járatSzám, Dátum, ülésSzám, ülésStátusz)
```

Erre a relációra vonatkozóan kiadhatjuk a következő lekérdezést:

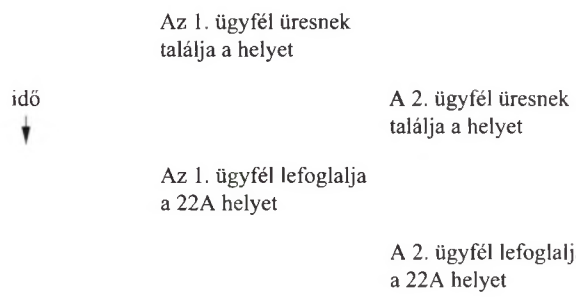
```
SELECT ülésSzám
FROM Légijáratok
WHERE járatSzám = 123 AND Dátum = DATE'2008-12-25'
      AND ülésStátusz ='szabad';
```

A járat száma és a dátum természetesen csak példaadatok, amelyeket a felhasználóval való párbeszéd során szerzett meg az ügyfél.

Ha a felhasználó ezután ráklikkel mondjuk a 22A szabad ülésre, akkor ez az ülés számára lefoglalódik. Az adatbázis módosul az alábbi módosító utasítással:

```
UPDATE Légijáratok
SET ülésStátusz = 'foglalt'
WHERE járatSzám = 123 AND Dátum = DATE'2008-12-25'
          AND ülésSzám ='22A';
```

Ez az utas nem az egyetlen utas, aki ebben a pillanatban a 123-as járatra, 2008. december 25-re helyet kíván biztosítani magának. Másik utas is lekérdezheti a helyek térképét ugyanebben a pillanatban, ebben az esetben a 22A ülést számára is szabad ülésként mutatja a rendszer. Ő szintén kiválaszthatja a 22A ülést, és ezt szeretné lefoglalni magának. Az események időbeli bekövetkezésének sorrendjét a 6.16. ábra érzékelteti. □



6.16. ábra. Két ügyfél egy időben próbálja meg ugyanazt a helyet lefoglalni

Ahogy a 6.40. példából látjuk, elképzelhető olyan eset, amikor két önmagában helyes művelet egyidejű (átlapolt) végrehajtása hibás végeredményhez vezethet: esetünkben ahhoz, hogy mindkét ügyfél azt gondolja, hogy a 22A hely az ő részére lett lefoglalva. A problémát az SQL a „tranzakciónak" nevezett technikával oldja meg. A tranzakció az utasítások egy csoportja, amelyet az adatbázisrendszer együttesen hajt végre. A 6.40. példában a lekérdezés és a módosítás egy tranzakcióba csoportosítható.[6] Az SQL lehetőséget ad a programozónak arra, hogy bizonyos tranzakciók más tranzakciókkal *sorbarendezhetők* legyenek. Azaz az ilyen tranzakciók végrehajtása *sorban történik* – átfedés nélkül, egyszerre csak egy kerül végrehajtásra.

Világos, hogy ha a bemutatott két helyfoglalás sorban egymás után (más szóval sorosan) valósul meg, akkor az említett hiba nem fordulhat elő. Az egyik felhasználó kérése előbb kerül sorra. Ő azt látja, hogy a 22A hely még nem foglalt, így ő lefoglalja. A másik felhasználó hívása ezután kezdődik, és számára a 22A választásának lehetőségét már fel sem kínálja a rendszer, hiszen az már foglalt. Hogy végül melyik felhasználó kapja meg a helyet, az már rajtuk múlik, az adatbázis-kezelő számára csak az a fontos, hogy egy helyet csak egyszer foglalhat le.

6.6.2. Műveletek atomisága

Az előző részben szemléltetett, két vagy több művelet sorba nem rendezettségéből eredő problémák mellett előfordulhat az is, hogy egyetlen művelet miatt kerül az adatbázis nem megfelelő állapotba, ha a művelet végrehajtása alatt valamilyen hardveres vagy szoftveres hiba történik. Egy ilyen lehetséges helyzetet mutatunk be a 6.40. példában. Itt is megjegyezzük, amint azt a 6.40. példa ismertetésénél tettük, hogy a valóságban használt adatbázis-kezelők és a jól elkészített alkalmazások képesek az ilyen problémák kivédésére is.

[6] Mindemellett az különösen szerencsétlen dolog lenne, ha egy tranzakcióba fognánk össze a felhasználó által, vagy olyan számítógép (mint például az utazási iroda számítógépe) által kezdeményezett műveleteket, amely nem a légitársaság felügyelete alá tartozik. Az adatbázison kívüli műveleteket is tartalmazó eseménysorok kezelésére más mechanizmust kell használnunk.

A sorbarendezhetőség biztosítása

A gyakorlati életben általában nem lehet megkövetelni, hogy az elvégzendő műveletek egymás után legyenek végrehajtva: egyszerűen túl sok van belőlük, ezért valamilyen párhuzamosítás szükséges. Így az adatbázis-kezelő rendszereknek valamilyen mechanizmusokkal biztosítaniuk kell a sorbarendezhetőséget: a felhasználó az eredményt úgy látja, mintha a műveletek végrehajtása sorban történt volna még akkor is, ha a végrehajtás valójában nem sorban történik.

Az egyik általánosan elterjedt megoldás az adatbázis-kezelők körében, hogy *zárolják* az elemeket azért, hogy két függvény ne férhessen hozzájuk egyidejűleg. A zárolást az 1.2.4. alfejezetben már megemlítettük. Kiterjedt módszerek léteznek arra, ahogy az adatbázisrendszerekben a zárolást megvalósítják. Ha például a 6.40. példa tranzakciójába beleírtuk volna a `Járatok` reláció zárolását, akkor az olyan tranzakciók, amelyek nem óhajtják a `Járatok` relációt használni, futhatnak párhuzamosan a helyfoglaló tranzakcióval, de olyanok nem, amelyek ugyanezt a relációt használnák.

6.41. példa. Tekintsünk egy másik jellemző helyzetet, amely egy bank számlaadatbázisának feldolgozása során fordulhatna elő. A helyzetet a

```
Számlák(számlaSzám, egyenleg)
```

relációval mutatjuk be.

Tegyük fel, hogy 100 $-t kell átutalni az 123 számláról a 456 számlára. Először ellenőriznünk kell, hogy van-e legalább 100 $ az 123 számlán, és ha igen, akkor a következő két lépést kell végrehajtanunk:

1. A 456 számla egyenlegét meg kell növelnünk 100-zal, ezt a következő SQL-utasítással érjük el:

```
UPDATE Számlák
SET egyenleg = egyenleg + 100
WHERE számlaSzám = 456;
```

2. Az 123 számla egyenlegét csökkentenünk kell 100-zal, ezt a következő SQL-utasítással érjük el:

```
UPDATE Számlák
SET egyenleg = egyenleg - 100
WHERE számlaSzám = 123;
```

Most tekintsük azt az esetet, amikor az 1. lépés után, de a 2. lépés előtt valamilyen hiba lép fel. Esetleg a számítógép hibásodik meg, vagy az adatbázissal való hálózati kapcsolatban történik átviteli hiba. Ekkor az adatbázis olyan állapotban marad, hogy az átutalt összeg megjelenik az arra kijelölt számlán, de az átutalt összeggel nem lesz megterhelve az a számla, amelyről a pénzt átutalták. A bank ekkor elvesztett annyi pénzt, amennyit át akartak utalni. □

A 6.41. példában bemutatott probléma elkerülhető lenne, ha a számlát terhelő, illetve a számlát jóváíró műveletek együttesét *atomian* hajthatnánk végre. Ez azt jelenti, hogy vagy mindkét műveletet végre kell hajtani, vagy egyik műveletet sem szabad végrehajtani. Az ilyen jellegű atomi végrehajtás megvalósítása általában úgy történik, hogy az adatbázist módosító műveleteket egy segédterületen hajtják végre, és eredményük csak azután lesz *véglegesítve* magában az adatbázisban, miután a teljes munka befejeződött, és akkor az összes változás beépül az adatbázisba, és láthatóvá válik más műveletek számára.

6.6.3. Tranzakciók

A 6.6.1. és 6.6.2. alfejezetekben megismert sorbarendezési és atomi műveletekkel kapcsolatos problémák megoldására vezették be az adatbázis-kezelőkben a *tranzakció* fogalmát. Egy tranzakció alatt adatbázis-elérési, illetve adatbázismódosító műveletek olyan csoportját értjük, amelyeket atomian kell végrehajtani: vagy a csoportba tartozó minden műveletet végre kell hajtani, vagy – ha ez valamilyen oknál fogva nem lehetséges – egyet sem szabad közülük végrehajtani. Ezenkívül az SQL megköveteli, hogy a tranzakciók alapértelmezés szerint sorba rendezhető módon valósuljanak meg. Az adatbázis-kezelő megengedheti, hogy a felhasználó egyes tranzakciók párhuzamos végrehajtására ennél gyengébb végrehajtási követelményeket határozzon meg. Ezen gyengítési és módosítási lehetőségekkel egy későbbi alfejezetben fogunk foglalkozni.

Amikor az általános *SQL-kezelőfelület*et használjuk, akkor általában minden utasítás önmagában egy tranzakciót alkot. Az SQL megengedi, hogy a programozó több utasítást is egy tranzakcióba csoportosítson. A `START TRANSACTION` SQL-utasítás használandó a tranzakció kezdetének kijelölésére. A tranzakció befejezésére két lehetőségünk van:

1. A `COMMIT` utasítás a tranzakció sikeres befejeződését eredményezi. Egy sikeresen befejeződött tranzakció a kezdete óta végrehajtott utasításainak módosításait tartósan rögzíti az adatbázisban, vagyis a módosítások *véglegesítődnek*. A `COMMIT` utasítás végrehajtása előtt a módosítások még átmenetiek, és a többi tranzakció nem is biztos, hogy látja azokat.

2. A `ROLLBACK` utasítás megszakítja a tranzakció végrehajtását, és annak sikertelen befejeződését eredményezi. Az így befejezett tranzakció SQL-utasításai által végrehajtott módosításokat a rendszer meg nem történtekké teszi (azaz *visszagörgeti*), a módosítások nem jelennek meg az adatbázisban.

Hogyan módosul az adatbázis egy tranzakció közben

A tranzakciók implementációja a különböző adatbázis-kezelő rendszerekben különbözőképpen történhet. Előfordulhat, hogy a tranzakciók már véglegesítésük előtt is módosítják az adatbázis tartalmát. Ha egy ilyen tranzakció sikertelenül fejeződik be, akkor (ha a programozó nem volt elővigyázatos) lehetséges, hogy a közbülső lépéseiben végzett adatmódosításokat más párhuzamosan futó tranzakciók felhasználták. Az adatbázis-kezelők ezt a problémát általában úgy oldják meg, hogy a tranzakciókban módosított adatelemeket zárolják egészen a `COMMIT` vagy `ROLLBACK` utasítás kiadásáig, és ezzel megakadályozzák, hogy más tranzakciók láthassák az átmeneti módosításokat. Ilyen zárakat vagy ezekkel ekvivalens eszközöket kell használnunk, ha a felhasználók sorbarendezhető módon kívánják tranzakcióik lefutását.

Amint azt a 6.6.4. alfejezetben látni fogjuk, az SQL számos más lehetőséget is kínál a tranzakciók közbülső lépéseiként elvégzett adatbázis-módosító műveletek eredményének kezelésére. Az is lehetséges, hogy a közbülső lépések során módosított adatelemek nem lesznek zárolva, és így láthatók lesznek annak ellenére, hogy később esetleg egy rollback eltünteti ezeket a módosításokat. A tranzakció tervezőjének a feladata annak eldöntése, hogy a közbülső lépések során megváltoztatott adatok láthatóságát meg kell-e akadályozni.

6.42. példa. Tegyük fel, hogy a 6.41. példában szereplő átutalást tranzakcióval szeretnénk megvalósítani. Ekkor az adatbázis elérése előtt végre kell hajtanunk egy `BEGIN TRANSACTION` utasítást. Ha úgy találjuk, hogy az átutaláshoz nincs elegendő pénz, akkor ki kell adnunk a `ROLLBACK` utasítást. Ha az átutaláshoz van elegendő pénz, akkor a két update (módosító) utasítás végrehajtása után végrehajtjuk a `COMMIT` utasítást. □

6.6.4. Csak olvasó tranzakciók

A 6.40. és 6.41. példákban olyan tranzakciókkal foglalkoztunk, amelyek olvasnak, majd esetleg módosítanak valamilyen adatokat az adatbázisban. Az ilyen tranzakciók esetében szükség van a sorbarendezhetőségi problémák vizsgálatára. A 6.40. példában azt láttuk, hogy mi történik akkor, ha két program ugyanakkor próbál meg lefoglalni egy helyet egy járaton; a 6.41. példában pedig azt, hogy mi történne, ha átutalás közben hiba lépne fel. Nagyobb viszont a szabadságunk a tranzakciók végrehajtásának párhuzamosítása során azoknál a tranzakcióknál, amelyek csak olvassák az adatbázis tartalmát, és nem módosítják azt.

Az alkalmazás, illetve a rendszer által kiadott rollback utasítások

A tranzakciók eddigi tárgyalása közben feltételeztük, hogy a commit és rollback közötti választás a tranzakciót futtató alkalmazás döntésétől függ. Vagyis, ahogyan a 6.44. és 6.42. példákban láthatjuk, egy tranzakció végrehajt számos adatbázis-műveletet, majd eldönti, hogy egy `COMMIT`-ot ad-e ki, ezzel véglegesítve a változtatásokat, vagy egy `ROLLBACK`-et, és ezzel visszatér a tranzakció előtti állapotba. Valójában az adatbázis-kezelő is kiadhat rollback utasítást egy tranzakcióra vonatkozóan, ha csak ezzel tudja biztosítani a tranzakciók atomosságát, vagy azt, hogy megfeleljenek a megadott elkülönítési szintnek. Ilyen rendszerutasítás kiadására valamilyen rendszerhiba vagy más tranzakciók párhuzamos futása adhat okot. Amennyiben a rendszer szakít meg egy tranzakciót, akkor a rendszer egyben egy speciális hibakódot, vagy egy úgynevezett kivételt generál. Ha egy alkalmazás garantálni szeretné, hogy a tranzakciói sikeresen lefussanak, akkor kezelnie kell az ilyen kivételeket, és a tranzakciót ismét futtatnia kell.

6.43. példa. Tegyük fel, hogy olyan programot írunk, mely a 6.40. példa `Légijáratok` relációjából olvas azért, hogy megállapítsa, hogy egy adott hely szabad-e. Egy ilyen programból egyszerre akárhány példányt elindíthatunk anélkül, hogy az adatbázis tartalmában bármilyen kárt okozhatnánk. Legrosszabb, ami ilyenkor történhet, hogy mialatt beolvassuk egy adott hely foglaltsági állapotát, addig valaki velünk párhuzamosan valamely más programmal éppen lefoglalja vagy felszabadítja azt. Az általunk visszakapott információ („szabad" vagy „foglalt") aktualitása akár ezredmásodperceken is múlhat, a válasz megfelel a lekérdezés időpontjában a valós állapotnak. □

Ha az SQL-adatbázis-kezelővel közöljük egy tranzakcióról, hogy az *csak olvasó* – azaz nem módosítja az adatbázis tartalmát –, akkor az adatbázis-kezelő felhasználhatja ezt az információt a tranzakciók optimálisabb ütemezésének megszervezésére. Általában az adatbázis-kezelő rendszer egyidejűleg több olyan csak olvasó tranzakciót is futtathat, amelyek ugyanazt az adatot olvassák, míg ugyanezek a tranzakciók nem futhatnának egy időben egy olyan másik tranzakcióval, amely az említett adatelemet módosítja.

Az alábbi SQL-utasítással közölhetjük az adatbázis-kezelővel, hogy a következő tranzakció *csak olvasó* tranzakció lesz:

```
SET TRANSACTION READ ONLY;
```

A fenti utasítást a tranzakció első művelete előtt kell végrehajtani. Azt is közölhetjük az adatbázis-kezelővel, hogy a következő tranzakció írhat és olvashat is. Ehhez az alábbi SQL-utasítást használjuk:

```
SET TRANSACTION READ WRITE;
```

Mindazonáltal ez az alapértelmezés szerinti viselkedés.

6.6.5. Piszkos adatok olvasása

*Piszkos adatok*nak nevezik a még nem véglegesített tranzakciók által módosított adatokat. *Piszkos adat olvasása* az az adatbeolvasó műveletet, amely egy piszkos adatot olvas be. A piszkos adatok beolvasásának az a veszélye, hogy az azt kiíró tranzakció lehet, hogy sikertelenül fejeződik be (abortál), és ilyenkor az illető tranzakció által kiírt piszkos adatok törlődnek az adatbázisból – mintha az időközben más tranzakciók által már beolvasott adatok sosem lettek volna az adatbázisban. Ha egy időközben visszavont piszkos adatot beolvasó tranzakció véglegesítve lesz, akkor az adatbázis tartalma egy olyan adatra épül, amelyet időközben már visszavontak.

Egyes alkalmakkor a piszkos adatbeolvasási műveletek nem jelentenek problémát, de van, amikor problémákat okozhatnak. Az is előfordulhat, hogy csekély veszéllyel járnak, és így megéri kockáztatni az esetleges piszkos olvasást. Ez mindig mérlegelés kérdése, amihez figyelembe kell venni az alábbi szempontokat:

1. Az adatbázis-kezelőnek sok időt és munkát jelent a piszkos olvasások megakadályozása.

2. A párhuzamosíthatóság lehetősége jelentősen csökken, mivel sok esetben várakoznia kell egyes tranzakcióknak a piszkos olvasások garantált elkerülése érdekében.

Tekintsünk át néhány példát a piszkos adatok beolvasásának lehetséges következményeiről.

6.44. példa. Nézzük meg ismét a 6.41. példabeli átutalási műveletet. Most azonban tételezzük fel, hogy az átutalást végző P program a következő lépések sorozatát hajtja végre:

1. Hozzáadja az átutalni kívánt összeget a 2. számla egyenlegéhez.

2. Ellenőrzi, hogy van-e elegendő pénz az 1. számlán.

   a) Ha az 1. számlán nincs meg a megfelelő fedezet, akkor levonja az előző lépésben a 2. számlára írt összeget és befejeződik.[7]

[7] Látnunk kell, hogy a P program itt olyan feladatokat végez el, amelyeket általában az adatbázis-kezelő szokott elvégezni. Például amikor P eldönti, hogy nem kell befejeznie a tranzakciót, akkor kiadhatna egy rollback utasítást, az adatbázis-kezelőre bízva az eddigi módosítások megsemmisítését.

b) Ha az 1. számlán van fedezet, akkor kivonja az átutalni kívánt összeget az 1. számla egyenlegéből és befejeződik.

Ha a fenti P program más tranzakciókhoz viszonyítva sorba rendezhető módon fut le, nem számít, hogy időlegesen pénzt írtunk a 2. számla egyenlegéhez. Utólag senki sem fog tudni erről, ha az átutalás fedezet hiánya miatt meghiúsulna.

Tegyük fel azonban, hogy megengedjük a piszkos adatok beolvasását. Tekintsünk három számlát: $A1$-et, $A2$-t és $A3$-at, egyenlegük rendre legyen 100 \$, 200 \$ és 300 \$. Tegyük fel, hogy egy T_1 nevű tranzakció végrehajtja a P programot, hogy átutaljon 150 \$-t $A1$-ről $A2$-re. Nagyjából ugyanekkor egy másik T_2 nevű tranzakció megkísérel átutalni szintén az említett P programmal 250 \$-t $A2$-ről $A3$-ra. Előfordulhat, hogy az egyes résztvevők az alábbi sorrendben hajtják végre lépéseiket:

1. A T_2 tranzakció végrehajtja az 1. lépést és hozzáad 250 \$-t $A3$ egyenlegéhez, amelyen így 550 \$ lesz.

2. A T_1 tranzakció végrehajtja az 1. lépést és hozzáad 150 \$-t $A2$ egyenlegéhez, amelyen így 350 \$ lesz.

3. T_2 végrehajtja a 2. lépés ellenőrzési részét, és látja, hogy $A2$ tartalmazza a 250 \$ átutalásához szükséges fedezetet (350 \$-t).

4. T_1 végrehajtja a 2. lépés ellenőrzési részét, és látja, hogy $A1$-en (100 \$) nincs fedezet arra, hogy 150 \$-t átutaljunk $A1$-ről $A2$-re.

5. T_2 végrehajtja a $2b$*)* lépést: levonja a 250 \$-t $A2$ egyenlegéből, amelyen most már csak 100 \$ van és befejeződik.

6. T_1 végrehajtja a $2a$*)* lépést: levonja a 150 \$-t $A2$ egyenlegéből, amelyen -50 \$ (mínusz) lesz, majd befejeződik.

A számlákon levő pénz összes mennyisége nem változott (az egyenlegek összege 600 \$). Minthogy T_2 piszkos adatot olvasott, nem tudtuk megakadályozni, hogy egy számlán negatív egyenleg képződjön, pedig ez volt a célja annak az ellenőrzésnek, amellyel az átutalás fedezetét vizsgáltuk. □

6.45. példa. Tekintsük a 6.40. példában megismert helyfoglalási feladatot, amelyet most a következő lépésekkel fogunk megoldani:

1. Megkeresünk egy szabad helyet és lefoglaljuk az `ülésStátusz` értékének `'foglalt'`-ra állításával. Ha nem találtunk szabad helyet, akkor befejezzük a tranzakciót.

2. Megkérdezzük a felhasználót, hogy megfelel-e a lefoglalt hely. Ha megfelel, akkor véglegesítjük a tranzakciót; ha nem felel meg, akkor felszabadítjuk az illető helyet az `ülésStátusz` értékét `'szabad'`-ra állítva, majd megismételjük az 1. lépést egy új helyet keresve.

Ha a fenti módon nagyjából ugyanabban az időben két tranzakció próbál meg helyet lefoglalni, akkor előfordulhat, hogy egyikük lefoglal egy S helyet, amelyet a felhasználó visszautasít. Ha a másik tranzakció akkor hajtja végre az 1. lépést, amikor még az adatbázis azt tartalmazza, hogy az S hely foglalt, akkor a program az illető helyet nem ajánlja fel a másik felhasználónak.

Hasonlóan a 6.44. példához, itt is a piszkos adatok olvasása okoz gondot: a második tranzakció olyan adatot olvasott, amelyet az első időlegesen kiírt, de később visszavont. □

Vajon mennyire fontos az a tény, hogy egy beolvasott adat piszkos? A 6.44. példában nagyon fontos volt, hiszen egy számla egyenlege negatív lett miatta, pedig ezt megpróbáltuk megelőzni. A 6.45. példában nem volt ilyen komoly a probléma, de ennek ellenére előfordulhat, hogy a második utas nem kapja meg kedvenc helyét, vagy esetleg alaptalanul mondják neki azt, hogy nincs hely a gépen. Ez a probléma könnyen feloldható, ha a tranzakciót újból lefuttatjuk (ekkor az S hely szabad volta kiderül, ha időközben más nem foglalta le). Látható, hogy ez utóbbi esetben igenis van értelme megengedni a piszkos adatok olvasását, mivel így csökkenthető az átlagos feldolgozási idő.

Az SQL lehetővé teszi, hogy egy-egy tranzakcióra vonatkozóan kijelöljük, megengedhető-e piszkos adatok olvasása. Erre a 6.6.4. alfejezetben már említett `SET TRANSACTION` SQL-utasítást használhatjuk. A 6.45. példában leírtakhoz hasonlóan működő tranzakció esetén a következő utasítást írhatjuk:

```
1)  SET TRANSACTION READ WRITE
2)      ISOLATION LEVEL READ UNCOMMITTED;
```

Az előbbi utasítás két dolgot ad meg:

1. Az 1. sor azt közli az adatbázis-kezelővel, hogy a tranzakció írhat és olvashat.

2. A 2. sor azt közli, hogy a tranzakció ún. piszkos adatok olvasását megengedő elkülönítési szinten futhat, azaz a tranzakció olvashat piszkos adatokat. A 6.6.6. alfejezetben foglalkozunk a 4 lehetséges elkülönítési szinttel, amelyek közül eddig kettőt ismertünk meg, a sorba rendezhetőt és a piszkos adatok olvasását megengedőt.

Figyeljük meg, hogy ha egy tranzakció nem csak olvasó (vagyis módosíthatja az adatbázist), és az elején megadjuk a `READ UNCOMMITTED` elkülönítési szintet, akkor ugyanitt meg kell adnunk a `READ WRITE` kulcsszavakat is. Emlékezzünk a 6.6.4. alfejezet alapján, hogy alapértelmezés szerint a tranzakciók író-olvasók, de az SQL esetében kivételt képeznek e szabály alól a piszkos adatok olvasását végző tranzakciók. Ezek alapértelmezés szerint csak olvasók, mivel az ilyen típusú író-olvasó tranzakciók végrehajtása túlságosan kockázatos. Ezért ha egy író-olvasó, piszkos adatok olvasására is képes tranzakciót szeretnénk megadni, akkor a `READ WRITE` kulcsszavakat explicit módon meg kell adnunk, ahogyan az a korábbi utasításban látható.

6.6.6. További elkülönítési szintek

Az SQL összesen négy *elkülönítési szint*et definiál, amelyek közül kettőt már megismerhettünk: a sorba rendezhetőt és a piszkos adatok olvasását megengedő szinteket. A másik két elkülönítési szint a *véglegesített olvasás*t és az *ismételhető olvasás*t biztosító szintek. Ezeket rendre a következő SQL-utasításokkal adhatjuk meg:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

illetve

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

Mindkét esetben a tranzakciók alapértelmezés szerint író-olvasók, így szükség esetén a fenti utasítások mögé kell írni a `READ ONLY` kulcsszavakat, ha csak olvasó tranzakciót szeretnénk megadni. Továbbá lehetőségünk van a sorba rendezhető végrehajtási módot választanunk a

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

SQL-utasítással, de mivel az SQL-ben ez az alapértelmezés, ezért ezt nem szükséges megadni.

A véglegesített olvasás elkülönítési szintjén – az elnevezésének megfelelően – nem megengedett a piszkos adatok olvasása, de a párhuzamosan működő véglegesített tranzakciók módosításai láthatóak lesznek. Ezért ennél a szintnél előfordulhat, hogy egy tranzakció ugyanazt a lekérdezést többször egymás után végrehajtja, és különböző eredményeket kap, ha a válaszok véglegesített tranzakciók eredményeként létrejött értékeket tükröznek.

6.46. példa. Tekintsük a 6.45. példában említett helyfoglaló programot, de tételezzük fel, hogy a tranzakciót a véglegesített olvasás elkülönítési szinten futtatjuk. Amikor a tranzakció egy helyet keres az 1. lépésben, akkor azokat a helyeket nem fogja foglaltnak látni, amelyeket egy másik tranzakció éppen lefoglal, de még nem commitált[8]. Azonban, ha az utas visszautasítja valamelyik felajánlott helyet, akkor egy következő próbálkozás során előfordulhat, hogy más szabad helyeket fog találni aszerint, ahogyan más tranzakciók sikeresen lefoglalnak egy-egy helyet, vagy éppen úgy döntenek, hogy a felajánlott hely nekik nem megfelelő, és felszabadítják azokat. □

Most tekintsük az ismételhető olvasás elkülönítési szintjét. Az elnevezés félrevezető, hiszen ha egy tranzakción belül egy lekérdezést többször végrehajtunk,

[8] Hogy ténylegesen mi történik, az most még kissé rejtélyes lehet, mivel nem ismertettük azokat az algoritmusokat, amelyek az elkülönítési szinteket biztosítják. Valószínűleg ha két tranzakció szabadnak látná ugyanazt a helyet, és megpróbálnák mindketten lefoglalni, akkor a rendszer az egyiküket megszakítaná és visszagörgetné, hogy elkerülje a holtpontot. (Lásd a 6.6.3. alfejezet „Az alkalmazás, illetve a rendszer által kiadott rollback utasítások" című bekeretezett részét.)

Különböző elkülönítési szinteken futó tranzakciók kölcsönhatása

Pontosan kell látnunk, hogy a tranzakciók elkülönítési szintje csak arra van hatással, hogy az *adott* tranzakció mely adatokat láthatja; az elkülönítési szint nem befolyásolja azt, hogy más tranzakciók mit láthatnak. Abban az esetben, ha a T tranzakció sorba rendezhető (serializable) elkülönítési szinten működik, akkor a T végrehajtása olyan, mintha teljes egészében más tranzakciók előtt vagy után történne. Ugyanakkor, ha a többi tranzakció közül néhány más elkülönítési szinten működik, akkor a piszkos adat olvasását megengedő (read-uncommitted) elkülönítési szinten futó tranzakciók a T által írott piszkos adatot azonnal láthatják, ahogy T kiírja. Még akkor is láthatják a T által kiírt adatot, ha T abortál, de ők piszkos adat olvasását megengedő elkülönítési szinten futnak.

akkor más-más választ kaphatunk az egyes végrehajtások alkalmával. Az ismételhető olvasás elkülönítési szintjén, ha egy lekérdezés egy adott sort visszaad az első végrehajtás alkalmával, akkor biztosak lehetünk abban, hogy az illető sort a lekérdezés másodszori végrehajtásakor is visszakapjuk. Ezen a szinten az is előfordulhat, hogy a lekérdezés egy következő végrehajtásakor ún. *fantomsor*okat is visszakapunk, amelyeket párhuzamosan működő tranzakciók illesztettek be az adatbázisba.

6.47. példa. Vizsgáljuk tovább a 6.45. és 6.46. példában megismert helyfoglalási problémát. Ha ezt a függvényt a megismételhető olvasás elkülönítési szintjén futtatjuk, akkor az első lekérdezés 1. lépése során visszakapott összes szabad helyet visszakaphatjuk a további lekérdezések során is.

Amennyiben eközben a `Légijáratok` táblába újabb járatokat vesznek fel, mert például a légitársaság egy adott járatán repülőgépet váltott, és ezért több az utasok rendelkezésére álló férőhely, vagy valaki lemondta a helyfoglalását, akkor a megismételhető olvasás elkülönítési szinten futó tranzakciók az újonnan bekerült üléseket is látni fogják a további helyfoglalási kísérleteik során. □

Az SQL elkülönítési szintjei közötti különbségeket a 6.17. ábra foglalja össze.

6.6.7. Feladatok

6.6.1. feladat. Az ebben és a következő feladatokban olyan programokat vizsgálunk, melyek az alábbi két reláción dolgoznak:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, ár)
```

A relációk a korábban már használt PC-adatbázisunkból származnak. Vázolja a következő, hagyományos programozási nyelven elkészített, de SQL-utasításokat

Elkülönítési szint	*Piszkos adat olvasása*	*Nem ismételhető olvasás*	*Fantomadatok*
Read Uncommitted	Megengedett	Megengedett	Megengedett
Read Committed	Nem megengedett	Megengedett	Megengedett
Repeatable Read	Nem megengedett	Nem megengedett	Megengedett
Serializable	Nem megengedett	Nem megengedett	Nem megengedett

6.17. ábra. Az SQL elkülönítési szintjeinek tulajdonságai

is tartalmazó programokat. Ne feledkezzen meg a `BEGIN TRANSACTION`, `COMMIT` és `ROLLBACK` utasítások megfelelő helyen való használatáról, és ha a tranzakció az adatbázisból csak olvas, akkor ezt jelezze az adatbázis-kezelőnek.

a) Adott egy sebességérték és egy RAM-mennyiség (a megvalósítandó függvény paramétereként). Keressük ki az adatbázisból a megadott sebességű és megadott mennyiségű RAM-memóriával rendelkező PC-k adatait, és nyomtassuk ki azok modellazonosítóit és az áraikat.

b) Adott modellazonosító esetén töröljük ki mindkét fenti táblából az e modellhez tartozó sorokat.

c) Csökkentsük 100 $-ral egy megadott modellazonosítóval rendelkező PC árát.

d) Adott egy számítógép gyártója, modellazonosítója, sebessége, RAM-mérete, merevlemezének mérete és ára. Ellenőrizzük, hogy az illető PC benne van-e már az adatbázisban. Ha már benne van, akkor írjunk ki egy figyelmeztető üzenetet; ha még nincs benne, akkor vegyük fel a fenti két táblába.

! **6.6.2. feladat.** Vizsgáljuk meg a 6.6.1. feladat programjainál esetlegesen felmerülő, atomossággal kapcsolatos problémákat, amelyek a program működése közben bekövetkező rendszerösszeomlás esetén bekövetkezhetnek.

! **6.6.3. feladat.** Tegyük fel, hogy a 6.6.1. feladat négy programjának valamelyikét végrehajtjuk egy T tranzakcióként, mialatt nagyjából ugyanabban az időben más tranzakciók a 6.6.1. feladat programjainak valamelyikét hajtják végre. Melyek azok a vizsgálandó esetek, amelyek akkor fordulhatnak elő, ha valamennyi tranzakció `READ UNCOMMITTED` elkülönítési szinten fut, és nem fordulhatnának elő, ha minden tranzakció `SERIALIZABLE` elkülönítési szinten futna? Gondoljuk végig külön-külön az egyes eseteket úgy, hogy a T tranzakció a 6.6.1. feladat *a)*–*d)* programjainak bármelyike lehet.

!! **6.6.4. feladat.** Tegyük fel, hogy van egy „állandóan" futó T tranzakciónk, mely óránként ellenőrzi, hogy van-e az adatbázisban egy 3.5-nél nem kisebb sebességű PC, amelynek ára 1000 $ alatt van. Ha talál egy ilyen PC-t, akkor kiírja a

jellemzőit, és befejeződik a futása. Ez alatt az idő alatt a 6.6.1. feladatban megismert programok futhatnak az adatbázisban. Vizsgáljuk meg mind a négy elkülönítési szinten, mi a hatása az említett T tranzakció futására annak a ténynek, hogy az az illető elkülönítési szinten fut.

6.7. Összefoglalás

- *SQL:* Az SQL a relációs adatbázis-kezelő rendszerek legfontosabb lekérdező nyelve A jelenlegi szabvány az SQL-99 vagy SQL3. A kereskedelmi rendszerek lényegében kielégítik ezt a szabványt.
- *Select-from-where lekérdezések:* Az SQL-lekérdezések legáltalánosabb alakja select-from-where alakú. Segítségével több reláció szorzatára (`FROM` záradék) alkalmazhatunk egy feltételt (`WHERE` záradék), és levetíthetjük a kívánt komponensekre (`SELECT` záradék).
- *Alkérdések:* A select-from-where lekérdezéseket alkérdésként is használhatjuk egy másik lekérdezés `WHERE` záradékában vagy `FROM` záradékában. Az `EXISTS`, `IN`, `ALL` és `ANY` operátorok logikai értékű feltételeket fejezhetnek ki a `WHERE` záradékban használt alkérdések eredményeként keletkezett relációkkal kapcsolatban.
- *Halmazműveletek relációkon:* Képezhetjük relációknak vagy relációkat definiáló lekérdezéseknek egyesítését, metszetét és különbségét a `UNION`, `INTERSECT` és `EXCEPT` kulcsszavak használatával.
- *Összekapcsolási kifejezések:* Az SQL-ben megtalálhatók olyan kifejezések, mint a `NATURAL JOIN`, amelyeket relációkra lekérdezésként vagy lekérdezésekben a `FROM` záradékban relációdefiniálásokra lehet alkalmazni.
- *Nullértékek:* Az SQL biztosít egy speciális értéket, a `NULL`-t, amely olyan komponensek helyén szerepel a relációkban, melyeknek nem tudjuk a konkrét értékét. A `NULL`-ra vonatkozó aritmetikai és logikai műveletek különböznek a szokásostól. Egy `NULL` érték összehasonlítása bármely értékkel, legyen az akár maga a `NULL` is, `ISMERETLEN` logikai értéket eredményez. Ez a logikai érték úgy viselkedik a logikai kifejezésekben, mintha az `IGAZ` és a `HAMIS` között félúton helyezkedne el.
- *Külső összekapcsolások:* Az `OUTER JOIN` operátor összekapcsolja a relációkat, de az eredmény tartalmazza azokat a sorokat is, amelyek nem kapcsolódnak össze a másik relációból egy sorral sem; ezeknek a soroknak az ismeretlen komponensei `NULL` értékekkel töltődnek ki.
- *A relációk multihalmazmodellje:* Az SQL a relációkat multihalmazoknak és nem halmazoknak tekinti. Az ismétlődéseket megszüntethetjük a `DISTINCT` kulcsszó segítségével, míg az `ALL` kulcsszó biztosítja, hogy az ismétlődések ne legyenek kitörölve olyan esetben, amikor az lenne az alapértelmezés.

- *Összesítések:* A reláció egy oszlopában megjelenő értékeket összesíthetjük a `SUM`, `AVG` (átlag), `MIN`, `MAX` és `COUNT` kulcsszavak segítségével. A sorokat az összesítés előtt csoportosíthatjuk a `GROUP BY` kulcsszavakkal. Bizonyos csoportokat kitörölhetünk az eredményből a `HAVING` kulcsszó segítségével.

- *Módosító utasítások:* Az SQL biztosítja a lehetőséget a sorok módosítására egy relációban. Beszúrhatunk sorokat az `INSERT`, törölhetünk a `DELETE` és módosíthatunk az `UPDATE` kulcsszavak segítségével.

- *Tranzakciók:* Az SQL lehetővé teszi utasítások tranzakcióba szervezését. Egy tranzakció véglegesíthető vagy hatása visszavonható (abortálható). A visszavonást kezdeményezheti az alkalmazás (ekkor a célja a módosítások visszavonása), vagy maga a rendszer abból a célból, hogy az atomosságot és az elkülönítést biztosítani tudja.

- *Elkülönítési szintek:* Az SQL a tranzakciók futtatásához négy különböző elkülönítési szintet biztosít, amelyek a legszigorúbbtól a leggyengébbig rendre a következők. Sorba rendezhető (egy ilyen szinten futó tranzakció úgy viselkedik, mintha vele egy időben nem futna másik tranzakció), ismételhető olvasást biztosító szint (egy ezen a szinten futó tranzakcióban egy lekérdezés során beolvasott összes sort újból megkapjuk a lekérdezés megismételt végrehajtásakor), véglegesített olvasás szintje (ezen a szinten a párhuzamosan futó módosító tranzakcióknak először le kell futniuk, és a véglegesített módosításokat látja a tranzakció). A negyedik, leggyengébb konzisztenciát biztosító szint nem támaszt megszorításokat azzal kapcsolatban, hogy a tranzakció más tranzakciók által elvégzett milyen módosításokat láthat.

6.8. Irodalomjegyzék

Az SQL-programozással kapcsolatban sok könyv található. Népszerűek a [3], [5] és [7]. A [6] az SQL-99 szabvány egy korai megjelenése.

Az SQL-t először [4]-ben definiálták és a „System R" [1] rendszer (az első generációs relációs adatbázisok egyike) részeként valósították meg [1].

A tranzakciók és kurzorok szabványával kapcsolatos problémák áttekintése [2]-ben található meg.

[1] M. M. Astrahan et al., „System R: a relational approach to data management," *ACM Transactions on Database Systems* **1**:2, pp. 97–137, 1976.

[2] H. Berenson, P. A. Bernstein, J. N. Gray, J. Melton, E. O'Neil, P. O'Neil, „A critique of ANSI SQL isolation levels," *Proceedings of ACM SIGMOD Intl. Conf. on Management of Data*, pp. 1–10, 1995.

[3] J. Celko, *SQL for Smarties*, Morgan-Kaufmann, San Francisco, 2005.

[4] D. D. Chamberlin et al., „SEQUEL 2: a unified approach to data definition, manipulation, and control," *IBM Journal of Research and Development* **20**:6, pp. 560–575, 1976.

[5] C. J. Date, H. Darwen, *A Guide to the SQL Standard*, Addison-Wesley, Reading, MA, 1997.

[6] P. Gulutzan, T. Pelzer, *SQL-99 Complete, Really*, R&D Books, Lawrence, KA, 1999.

[7] J. Melton, A. R. Simon, *Understanding the New SQL: A Complete Guide*, Morgan-Kaufmann, San Francisco, 2006.

7. fejezet

Megszorítások és triggerek

Ebben a fejezetben az SQL-nek azokat a sajátosságait tekintjük át, amelyek az *aktív* elemek létrehozásával kapcsolatosak. Egy aktív elem olyan kifejezés vagy utasítás, amelyet egyszer megírunk, eltárolunk az adatbázisban, és azt várjuk el tőle, hogy a megfelelő időpillanatokban lefusson. Ez az időpont lehet egy esemény bekövetkezése, mint például egy adott relációba való beszúrás, vagy lehet az adatbázisnak olyan megváltozása, amikor egy logikai értékű feltétel igazzá válik.

Az egyik komoly nehézség, amivel az alkalmazásfejlesztők szembe találják magukat, hogy az adatbázis módosításakor az új információ nagyon sokféleképpen lehet hibás. Kézzel bevitt adatok esetén például gyakran fordul elő elírás vagy másolási hiba. Az alkalmazásokat megírhatjuk úgy is, hogy minden beszúrást, törlést és módosító utasítást a szükséges, helyességet biztosító ellenőrzésekhez rendelünk hozzá. A legjobb ezeket az ellenőrzéseket az adatbázisban tárolni, és az ABKR-re bízni az ellenőrzést. Ekkor ugyanis biztosan nem felejtődik el egyetlen ellenőrzés sem, sőt így a felesleges munkát is elkerülhetjük.

Az SQL számos lehetőséget kínál arra, hogy az *épségi megszorítások*at az adatbázisséma részeként adjuk meg. Ebben a fejezetben a legfontosabb módszereket fogjuk áttekinteni. Korábban már foglalkoztunk a kulcsmegszorításokkal, amelyeknél egy attribútum vagy egy attribútumhalmaz a reláció kulcsaként van megadva. Az SQL lehetővé teszi az attribútumokra, a sorokra vonatkozó megszorításokat és a több relációt érintő megszorításokat is. Ez utóbbiakat önálló megszorításoknak nevezzük. Végül pedig a fejezet végén a „triggereket" fogjuk bemutatni. Ezek olyan aktív elemek, amelyek bizonyos események hatására jönnek működésbe. Ilyen esemény lehet például egy adott relációba való beszúrás.

7.1. Kulcsok és idegen kulcsok

Emlékezzünk vissza a 2.3.6. alfejezetre, amelyben láthattuk, hogy az SQL lehetővé teszi a `PRIMARY KEY`, illetve `UNIQUE` kulcsszó használatával azt, hogy egy vagy több attribútum a reláció kulcsa legyen. Az SQL nyelv a „kulcs" kifejezést

használja bizonyos hivatkozásiépség-megszorítások esetén is. Ezek a megszorítások, amelyeket idegenkulcs-megszorításoknak hívunk, azt mondják ki, hogy egy relációban előforduló értéknek szerepelnie kell egy másik reláció elsődleges kulcsoszlopában vagy oszlopaiban is.

7.1.1. Idegen kulcsok megadása

Az idegenkulcs-megszorítás bizonyos attribútumok értékeinek jelentésére előírt követelmény. Vegyük például a 2.21. példát, amelyben azt néztük meg, hogyan fejezhető ki a relációs algebrában az a megszorítás, hogy a filmeknek a producer „azonosító szám" értéke néhány gyártásirányítónak is azonosító száma lesz a `GyártásIrányító` relációban.

Az SQL-ben egy reláció azon attribútumát vagy attribútumait *idegen kulcs*nak deklarálhatjuk, amelyek egy másik reláció (ez lehet akár ugyanaz a reláció is) bizonyos attribútumaira hivatkoznak. Ez a deklaráció két dolgot jelent egyszerre.

1. A másik reláció azon attribútumait, amelyekre hivatkozunk, elsődleges kulcsként vagy `UNIQUE` kulcsként kell deklarálni abban a relációban. Enélkül nem adhatjuk meg az idegenkulcs-deklarációt.

2. Az idegen kulcs értékeinek, amelyek előfordulnak az első relációban, elő kell fordulniuk a hivatkozott attribútumokban is a másik reláció valamelyik sorában. Még pontosabban kifejezve, legyen F egy idegen kulcs, amely egy másik reláció G attribútumhalmazára hivatkozik. Tegyük fel, hogy az első relációnak egy t sora az F attribútumaiban csupa nem `NULL` értékkel rendelkezik. Jelöljük t-nek ezen attribútumokon felvett értékeit $t[F]$-fel. Ekkor a hivatkozott relációnak kell hogy legyen olyan s sora, amelyik a G attribútumain megegyezik $t[F]$-fel, vagyis $s[G] = t[F]$.

Az idegen kulcsot is kétféleképpen deklarálhatjuk ugyanúgy, ahogyan azt az elsődleges kulcsok esetében láttuk.

a) Ha az idegen kulcs egyetlen attribútum, akkor az attribútum neve és típusa után adhatjuk meg, hogy az egy másik tábla egy attribútumára hivatkozik. (Annak az attribútumnak ott elsődleges kulcsnak vagy unique-nak kell lennie.) A deklaráció formája a következő:

`REFERENCES` <tábla>(<attribútum>)

b) A másik lehetőség, hogy a `CREATE TABLE` utasításban az attribútumok listája után egy külön deklarációban adjuk meg, hogy bizonyos attribútumok idegen kulcsot alkotnak. Itt kell megadnunk azt a táblát és azokat az attribútumokat, amelyekre az idegen kulcs hivatkozik. (Ezeknek az attribútumoknak ez esetben is elsődleges vagy unique kulcsnak kell lenniük.) A deklaráció formája ekkor a következő:

```
FOREIGN KEY (<attribútumok>) REFERENCES
        <tábla>(<attribútumok>)
```

7.1. példa. Tegyük fel, hogy a következő relációt szeretnénk deklarálni:

```
Stúdió(név, cím, elnökAzon)
```

Ennek az elsődleges kulcsa a `név`, az `elnökAzon` attribútuma pedig idegen kulcs, amelyik a

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

reláció `azonosító` attribútumára hivatkozik.

Az `elnökAzon` közvetlenül deklarálható az `azonosító` attribútumra történő hivatkozással a következő módon:

```
CREATE TABLE Stúdió (
    név CHAR(30) PRIMARY KEY,
    cím VARCHAR(255),
    elnökAzon INT REFERENCES GyártásIrányító(azonosító)
);
```

Vagy megadhatjuk úgy is, hogy az idegen kulcsot külön deklaráljuk:

```
CREATE TABLE Stúdió (
    név CHAR(30) PRIMARY KEY,
    cím VARCHAR(255),
    elnökAzon INT,
    FOREIGN KEY elnökAzon REFERENCES
                GyártásIrányító(azonosító)
);
```

Figyeljük meg, hogy a hivatkozott `azonosító` attribútum a `GyártásIrányító` táblának valóban kulcsa, ahogyan az szükséges. Mindkét fenti deklarációnak az a jelentése, hogy ha egy érték szerepel a `Stúdió` tábla egy sorának `elnökAzon` oszlopában, akkor ennek az értéknek szerepelnie kell a `GyártásIrányító` tábla valamely sorának azonosító oszlopában is. Kivétel az az eset, ha a `Stúdió` tábla egy sorában az `elnökAzon` oszlopban `NULL` érték szerepel. Ilyenkor nem követelmény, hogy a `NULL` érték szerepeljen a másik tábla azonosító oszlopában. Valójában az `azonosító` elsődleges kulcs, és így nem is szerepelhet benne `NULL` érték. □

7.1.2. Hivatkozási épség fenntartása

A séma tervezője három lehetőség közül választhat az idegenkulcs-megszorítás kikényszerítésére. Az általános megközelítéseket megfigyelhettük a 7.1. példa vizsgálatánál, amelyben elvárás volt, hogy a `Stúdió` reláció `elnökAzon` értéke egyben a `GyártásIrányító` reláció azonosító értéke is legyen. A következő műveleteket az ABKR visszautasítja, vagyis egy futási hibát generál:

a) Ha megpróbálunk egy olyan sort beszúrni a `Stúdió` táblába, amelyben az `elnökAzon` értéke nem `NULL`, és nem egyezik meg a `GyártásIrányító` táblabeli `azonosító`-hoz tartozó egyik értékével sem.

b) Ha megpróbáljuk módosítani a `Stúdió` tábla egy sorát, és az `elnökAzon` attribútumot egy olyan nem `NULL` értékre változtatjuk, amelyik nem egyezik meg a `GyártásIrányító` táblabeli egyik `azonosító` értékkel sem.

c) Ha megpróbálunk kitörölni egy sort a `GyártásIrányító` táblából, amelyben az `azonosító` értéke nem `NULL` és szerepel a `Stúdió` tábla `elnökAzon` oszlopában.

d) Ha megpróbáljuk módosítani a `GyártásIrányító` tábla egy sorát oly módon, hogy az `azonosító` értékét is megváltoztatjuk és a régi `azonosító` értéke szerepel a `Stúdió` tábla `elnökAzon` oszlopában.

Az első két módosítás esetén, amelyek az idegenkulcs-megszorítással deklarált relációt akarták megváltoztatni, nincs választási lehetőségünk. A rendszer egyszerűen visszautasítja a megszorítást sértő módosítást. A hivatkozott relációra vonatkozó módosításoknál (vagyis az utolsó kettőnél) viszont a tervező három lehetőség közül választhat:

1. *Alapértelmezés szerinti eljárás (Megszorítást sértő módosítások visszautasítása).* Az SQL-ben az alapértelmezés szerinti eljárás minden olyan módosítást, amely megsérti a hivatkozásiépség-megszorítást, visszautasít.

2. *Továbbgyűrűző eljárás.* Ezzel az eljárással az idegen kulcsok hivatkozott attribútuma(i) is megváltoznak. Ha például a továbbgyűrűző eljárással egy stúdió elnökéhez tartozó `GyártásIrányító` sort törlünk, akkor a rendszer a hivatkozási épség megőrzéséhez a `Stúdió` reláció hivatkozott sorait is törli. Ha c_1-ről c_2-re változtatjuk egy gyártásirányító azonosítóját, és van olyan sora a `Stúdió` táblának, amelyben az `elnökAzon` értéke c_1, akkor a rendszer ezt az értéket is c_2-re módosítja.

3. *A* `NULL` *értékre állítás módszere.* Itt a hivatkozott reláció egy idegen kulcsának értékét érintő módosításánál a hivatkozott reláció megfelelő értékeit `NULL` értékre változtatjuk. Ha például a `GyártásIrányító` reláció az egyik stúdió elnökéhez tartozó sorát töröljük, akkor a rendszer a szóban forgó stúdió `elnökAzon` értékeit `NULL` értékre kell állítsa. Ha ennek az elnöknek az azonosító számát a `GyártásIrányító` relációban megváltoztatjuk, akkor a `Stúdió` reláció `elnökAzon` értékét `NULL` értékre kell változtassuk.

A fenti módszerek közül az alkalmazni kívántat a törlés és módosítás esetére külön-külön, egymástól függetlenül megadhatjuk, amikor az idegen kulcsot deklaráljuk. A megadásuk úgy történik, hogy a törlés esetén használandó módszert az `ON DELETE`, a módosítás esetén alkalmazandót pedig az `ON UPDATE` kulcsszó után adjuk meg. A módszerekre vonatkozó kulcsszavak pedig `NULL` értékre állítás esetén `SET NULL`, továbbgyűrűző módszer esetén pedig `CASCADE`.

7.2. példa. Nézzük meg, hogyan kell módosítanunk a

```
Stúdió(név, cím, elnökAzon)
```

reláció 7.1. példában szereplő deklarációját, hogy a

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

relációra vonatkozó törlések és módosítások esetén az általunk kívánt módszert alkalmazza a rendszer. A 7.1. ábrán az említett példában szereplő első `CREATE TABLE` utasítást láthatjuk az `ON DELETE` és `ON UPDATE` záradékkal kiegészítve. Az 5. sor azt adja meg, hogy ha a `GyártásIrányító` táblából kitörlünk egy sort, akkor az összes `Stúdió`-beli sorban, amelyekben az éppen törölt gyártásirányító volt az elnök, az `elnökAzon` értékét `NULL`-ra kell változtatni. A 6. sor azt mondja meg, hogy ha a `GyártásIrányító` tábla egy sorában megváltoztatjuk az `azonosító`-t, akkor a `Stúdió` tábla megfelelő soraiban ugyanarra az értékre kell változtatni az `elnökAzon` értékét.

```
1)  CREATE TABLE Stúdió (
2)      név CHAR(30) PRIMARY KEY,
3)      cím VARCHAR(255),
4)      elnökAzon INT REFERENCES GyártásIrányító(azonosító)
5)          ON DELETE SET NULL
6)          ON UPDATE CASCADE
    );
```

7.1. ábra. Módszer megadása a hivatkozási épség megőrzésére

Figyeljük meg, hogy a példában a törlés esetén a nullértékre állítás tűnik értelmes megoldásnak, míg a módosítás esetén a továbbgyűrűző módszer a logikus. Ha ugyanis egy stúdió elnöke visszavonul, attól még a stúdió tovább működik, egy darabig esetleg elnök nélkül. Ha azonban egy stúdió elnökének megváltozik az azonosítója, az valószínűleg csak valamilyen adminisztratív változást jelent. Valószínűleg az illető továbbra is elnöke marad a stúdiónak, és ezért ezt a változást abban a relációban is követni kellene. □

7.1.3. Megszorítások ellenőrzésének késleltetése

Vegyük a 7.1. példában említett szituációt, ahol a `Stúdió` reláció `elnökAzon` attribútuma idegen kulcs, amely a `GyártásIrányító` tábla `azonosító` oszlopára hivatkozik. Tegyük fel, hogy Arnold Schwarzenegger Kalifornia kormányzójaként visszavonul, és elhatározza, hogy egy filmstúdiót alapít, amelynek neve La Vista Studios lesz. A stúdió elnöke természetesen ő lesz. Ha végrehajtjuk a következő beszúrást, akkor gondban leszünk.

Lógó sorok és módosítási eljárások

Azokat a sorokat, amelyekben olyan idegen kulcs értéke szerepel, amely nincs benne a hivatkozott táblában, *lógó sorok*nak nevezzük. Emlékezzünk rá, hogy azokat a sorokat is lógó soroknak neveztük, amelyek egy összekapcsolás esetén kimaradtak az eredményből. A fenti két dolog szoros kapcsolatban áll egymással. Ha egy sorbeli idegen kulcs értéke hiányzik a hivatkozott táblából, akkor az adott sor nem lesz benne a relációnak a hivatkozott relációval vett összekapcsolásában. Ha az összekapcsolás az idegen kulcs, illetve a hivatkozott kulcs egyenlőségén alapul, akkor az összekapcsolást *idegenkulcs-összekapcsolás*nak nevezzük. A lógó sorok pontosan azok a sorok lesznek, amelyek megsértik az idegenkulcs-megszorítás hivatkozásiépség-elvárását.

```
INSERT INTO Stúdió
VALUES('La Vista', 'New York', 23456);
```

Az lesz a probléma, hogy nincs a `GyártásIrányító` táblának 23456 azonosítójú sora (feltételezzük, hogy ez lesz Arnold Schwarzenegger új azonosítója), és így nyilvánvaló módon megsértjük az idegenkulcs-megszorítást.

Az egyik lehetséges megoldás, hogy először szúrjuk be a La Vista stúdióra vonatkozó sort az elnök azonosítója nélkül:

```
INSERT INTO Stúdió(név, cím)
VALUES('La Vista', 'New York');
```

Ez a módosítás már nem sérti meg a megszorítást, mivel a La Vistára vonatkozó sorban `NULL` érték kerül az `elnökAzon` oszlopba, és ez nem követeli meg semmilyen érték meglétét a hivatkozott oszlopban. Mielőtt azonban módosítani tudnánk az értéket az alábbi utasítással, először be kell szúrnunk egy Arnold Schwarzeneggerre vonatkozó sort a `GyártásIrányító` táblába a megfelelő azonosítóval.

```
UPDATE Stúdió
SET elnökAzon = 23456
WHERE név = 'La Vista';
```

Ha nem szúrjuk be előzőleg a megfelelő sort a `GyártásIrányító` táblába, akkor a fenti módosítás is meg fogja sérteni az idegenkulcs-megszorítást.

A jelen példában, ha először beszúrjuk Arnold Schwarzeneggert és az ő azonosítóját a `GyártásIrányító` táblába, mielőtt a La Vistát is beszúrnánk, ezzel biztosan elkerülhetjük az idegen kulcs megsértését. Vannak azonban olyan esetek, amikor a *körkörös megszorítások* miatt még az adatbázis-módosítások sorrendjének megfontolt megválasztásával sem tudjuk orvosolni a problémát.

7.3. példa. Ha a gyártásirányítók csak stúdióelnökök lehetnének, akkor az azonosító oszlopot idegen kulcsként kellene deklarálnunk, ami a Stúdió(elnökAzon)-ra hivatkozik. Ez esetben az elnökAzon oszlopot UNIQUE-nak kellene deklarálnunk, aminek akkor van értelme, ha feltesszük, hogy egy ember nem lehet egyidejűleg két stúdiónak az elnöke.

Ebben az esetben lehetetlen új stúdiókat és elnököket felvinni. Nem tudunk új elnökAzon értékkel rendelkező sort beszúrni a Stúdió táblába, mert ez a sor megsértené azt az idegen kulcsot, ami az elnökAzon-ról a GyártásIrányító(azonosító)-ra mutat. Új azonosító értékkel rendelkező sort sem tudunk beszúrni a GyártásIrányító táblába, mert az pedig azt az idegen kulcsot sérti meg, amely az azonosító-ról a Stúdió(elnökAzon)-ra mutat. □

A 7.3. példában szereplő probléma másképp is megoldható:

1. Először is szükségünk van arra, hogy a két beszúrást (az egyiket a Stúdió, a másikat a GyártásIrányító táblába) egyetlen tranzakcióba foglaljunk.

2. Másrészt valahogy meg kell mondanunk az ABKR-nek, hogy a megszorításokat ne ellenőrizze, amíg az egész tranzakció be nem fejeződött. A tranzakciók nyelvén ezt a befejeződést jóváhagyásnak nevezzük.

Ahhoz hogy a 2. pontban az ABKR-t informáljuk, bármelyik megszorítás (kulcs, idegen kulcs vagy a fejezetben később előforduló megszorítástípusok) deklarációját egy DEFERRABLE (késleltethető) vagy egy NOT DEFERRABLE kulcsszó kell kövesse. Az utóbbi az alapértelmezés, és ez azt jelenti, hogy minden adatbázis-módosításkor a megszorítás közvetlenül utána ellenőrzésre kerül, ha a módosítás egyáltalán megsértheti az idegenkulcs-megszorítást. Ha azonban a megszorítást DEFERRABLE-ként deklaráljuk, akkor lehetőségünk van arra, hogy azt mondjuk a rendszernek, hogy a megszorítás ellenőrzésével várjon a tranzakció végéig.

A DEFERRABLE kulcsszót követhet egy INITIALLY DEFERRED vagy egy INITIALLY IMMEDIATE kiegészítés is. Az első esetben az ellenőrzés a tranzakció jóváhagyásáig késleltetve lesz. A második esetben az ellenőrzés minden egyes utasítás után azonnal megtörténik.

7.4. példa. A 7.2. ábra a Stúdió reláció módosított deklarációját mutatja. A módosítás az idegen kulcs késleltetését teszi lehetővé a tranzakció végéig. Az elnökAzon oszlopot UNIQUE-nak deklaráltuk, hogy más relációk idegenkulcs-megszorításai hivatkozhassanak rá.

Ha a 7.3. példában szereplő másik idegenkulcs-deklarációt is megadtuk volna a GyártásIrányító(azonosító)-ról a Stúdió(elnökAzon)-ra, akkor olyan tranzakciókat írhatnánk, amelyek két sort szúrnak be (mindegyik táblába), és a két idegen kulcs csak a két beszúrás után kerül ellenőrzésre. Így ha egy új stúdiót és az új elnököt is beszúrjuk ugyanazzal az azonosítóval, akkor nem sérül meg egyik megszorítás sem. □

```
CREATE TABLE Stúdió (
    név CHAR(30) PRIMARY KEY,
    cím VARCHAR(255),
    elnökAzon INT UNIQUE
        REFERENCES GyártásIrányító(azonosító)
        DEFERRABLE INITIALLY DEFERRED
);
```

7.2. ábra. Az `elnökAzon` egyedisége és a megszorítás késleltetése

Két további dolgot kell megemlítenünk a megszorítások késleltetésével kapcsolatban:

- Bármelyikfajta megszorításnak nevet adhatunk. Hogy ezt hogyan tehetjük, arról majd a 7.3.1. alfejezetben szólunk bővebben.
- Ha egy megszorításnak nevet adtunk (például `saját`), akkor egy késleltethető megszorítást azonnaliról késleltetettre változtathatunk a következő SQL-utasítással:

  ```
  SET CONSTRAINT saját DEFERRED;
  ```

  Ennek a fordítottját is megtehetjük, ha a `DEFERRED` helyett az `IMMEDIATE` kulcsszót használjuk.

7.1.4. Feladatok

7.1.1. feladat. A 2.2.8. alfejezetben bevezetett filmes adatbázisban minden relációhoz adtunk meg kulcsot.

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

Adjuk meg a következő hivatkozásiépség-megszorítások deklarációit a filmadatbázisra:

a) Egy film producerének szerepelnie kell a gyártásirányítók között. A `GyártásIrányító` tábla olyan módosításait, amelyek ezt a megszorítást megsértenék, utasítsa vissza a rendszer.

b) A feltétel legyen ugyanaz, mint az előbb, de a megszorítást megsértő módosítások esetén a `Filmek` reláció `producerAzon` oszlopát változtassa a rendszer `NULL`-ra.

c) A feltétel legyen ugyanaz, mint *a)*-ban, de a megszorítást megsértő törlések vagy módosítások esetén törölje vagy módosítsa a rendszer a `Filmek` reláció megfelelő sorait is.

d) A `SzerepelBenne` relációban lévő filmeknek benne kell lenniük a `Filmek` táblában is. A megszorítást megsértő utasításokat a rendszer utasítsa vissza.

e) A `SzerepelBenne` relációban levő színészeknek benne kell lenniük a `FilmSzínész` táblában is. A megszorítást megsértő utasításokat a rendszer kezelje le a megfelelő sorok törlésével.

! **7.1.2. feladat.** Azt a megszorítást szeretnénk megadni, hogy minden filmnek, amely a `Filmek` relációban szerepel, legalább egy színésszel benne kell lennie a `SzerepelBenne` relációban is. Megadhatjuk-e ezt egy idegen kulcs segítségével? Indokoljuk meg a választ!

7.1.3. feladat. Javasoljunk megfelelő kulcsokat és idegen kulcsokat a 2.4.1. feladat PC-adatbázisának relációihoz. Módosítsuk a 2.3.1. feladatban szereplő SQL-sémát úgy, hogy az tartalmazza ezeknek a kulcsoknak a deklarációját.

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

7.1.4. feladat. Javasoljunk megfelelő kulcsokat a 2.4.3. feladatban szereplő `Csatahajók` adatbázis relációihoz. Módosítsuk a 2.3.2. feladatban szereplő SQL-sémát úgy, hogy az tartalmazza ezeknek a kulcsoknak a deklarációját.

```
Hajóosztályok(hajóosztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, hajóosztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

7.1.5. feladat. Adjuk meg a következő hivatkozásiépség-megszorításokat az előző feladatban szereplő adatbázissémára vonatkozóan. Az ott általunk megadott kulcsokat használjuk, és a megszorításokat megsértő utasításokat kezeljük le úgy, hogy a megfelelő értékeket `NULL`-ra változtatjuk.

a) Minden `Hajók` táblabeli hajóosztálynak szerepelnie kell a `Hajóosztályok` táblában is.

b) Minden `Kimenetelek` táblabeli csatának szerepelnie kell a `Csaták` táblában is.

c) Minden `Kimenetelek` táblabeli hajónak szerepelnie kell a `Hajók` táblában is.

7.2. Attribútumokra és sorokra vonatkozó megszorítások

Egy SQL-beli `CREATE TABLE` utasításon belül kétféle megszorítást fogalmazhatunk meg:

1. Egy attribútumra vonatkozó megszorítást.
2. Egy sor egészére vonatkozó megszorítást.

A 7.2.1. alfejezetben bevezetünk egy egyszerű megszorítástípust az attribútum értékére vonatkozóan, amely azt mondja ki, hogy az attribútum nem veheti fel a `NULL` értéket. A 7.2.2. alfejezetben áttekintjük a fenti 1. módon kifejezett megszorítások alapvető formáját. Ezeket *attribútumra vonatkozó* `CHECK` *feltételek*nek fogjuk hívni. A 2. módon kifejezett sorokra vonatkozó megszorításokat a 7.2.3. alfejezetben tárgyaljuk.

Vannak további, még általánosabbfajta megszorítások, ezekről a 7.4. és 7.5. alfejezetben lesz szó. Ez utóbbiak segítségével korlátozhatjuk a teljes relációra vonatkozó módosításokat, vagy akár több relációra vonatkozó módosításokat is, de használhatjuk azokat egyetlen attribútum vagy sor értékének korlátozására is.

7.2.1. `NOT NULL` feltételek

Egy egyszerű megszorítás, amely egy attribútumhoz hozzárendelhető, a `NOT NULL` feltétel. Ez nem engedi meg olyan sorok előfordulását, amelyekben az adott attribútum értéke `NULL`. A megszorítást úgy deklarálhatjuk, hogy a `CREATE TABLE` utasításban az attribútum megadása után a `NOT NULL` kulcsszót szerepeltetjük.

7.5. példa. Tegyük fel, hogy a `Stúdió` relációban az `elnökAzon` értéke nem lehet `NULL`. Ezt megadhatjuk úgy, hogy a 7.1. ábra 4. sorát a következőre változtatjuk:

```
4) elnökAzon INT REFERENCES GyártásIrányító(azonosító) NOT NULL
```

Ennek a változtatásnak a következő következményei lesznek:

- Nem szúrhatunk be egy olyan sort a `Stúdió` relációba, amelyre csak a nevet és a címet adtuk meg, mert ekkor az új sorban az `elnökAzon` értéke `NULL` lenne.
- Nem alkalmazhatjuk a `NULL` értékre állítás módszerét a 7.1. ábra 5. sorához hasonló módon, ahol úgy oldaná meg a rendszer az idegenkulcs-megszorítások megsértését, hogy az `elnökAzon` értékét `NULL`-ra változtatná.

□

7.2.2. Attribútumra vonatkozó CHECK feltételek

Bonyolultabb megszorítások rendelhetők hozzá egy attribútumhoz, ha a deklarációban a CHECK kulcsszót használjuk. A kulcsszót egy zárójelbe tett feltétel követi, amelyet az attribútum minden értékének ki kell elégítenie. A gyakorlatban egy attribútumra vonatkozó CHECK feltétel olyan, mint egy egyszerű korlátozás az attribútum értékeire vonatkozóan. Ilyen lehet például a megengedett értékek felsorolása vagy egy aritmetikai egyenlőtlenség. Elvben azonban a feltétel bármi lehet, ami egy SQL-lekérdezésben a WHERE után állhat. Hivatkozhat például az éppen korlátozni kívánt attribútumra oly módon, hogy a feltételben ennek az attribútumnak a nevét szerepeltetjük. Ha azonban a feltétel másik relációra vagy más attribútumokra hivatkozik, akkor a megfelelő relációkat egy alkérdés FROM záradékában kell megadnunk. Ez magára a korlátozni kívánt attribútum relációjára is vonatkozik.

Egy attribútumra vonatkozó CHECK feltételt akkor ellenőriz a rendszer, amikor valamelyik sorban az adott attribútum új értéket kap. Ez történhet egy módosítás útján vagy egy beszúrás alkalmával. Ha az új érték megsérti a feltételt, akkor a rendszer az adatmódosítást visszautasítja. A módosítások esetén a megszorítás ellenőrzését az új értékre végezzük, nem pedig a régire. Ha az új érték megsérti a megszorítási feltételt, akkor a rendszer visszautasítja a módosítási kísérletet.

Fontos megérteni, hogy az attribútum alapú CHECK megszorítást nem ellenőrzi le a rendszer, amennyiben a módosítás nem érinti a megszorított attribútum értékét. Ez a korlát viszont vezethet a megszorítás megsérüléséhez, ha a megszorításban érintett egyéb értékek megváltoznak. Először nézzünk meg egy egyszerű példát az attribútum alapú ellenőrzésre. Majd vizsgáljunk meg egy alkérdést tartalmazó megszorítást, illetve annak a következményeit is, hogy a megszorítás csak akkor kerül ellenőrzésre, ha az attribútumát módosítjuk.

7.6. példa. Tegyük fel, hogy azt követeljük meg, hogy az azonosító számok legalább hatjegyűek legyenek. Ehhez a 7.1. ábra 4. sorát, amelyik a

```
Stúdió(név, cím, elnökAzon)
```

reláció sémájának egy részét deklarálja, a következőre módosíthatjuk:

```
4)  elnökAzon INT REFERENCES GyártásIrányító(azonosító)
                CHECK (elnökAzon >= 100000)
```

Vegyünk egy másik példát. A

```
FilmSzínész(név, cím, nem, születésiDátum)
```

reláció nem attribútumát a 2.8. ábrán CHAR(1) típusúnak deklaráltuk, ami egyetlen karaktert jelent. Valójában azonban azt szeretnénk, ha ez az egy karakter csak 'F' vagy 'N' lehetne. Ezt elérhetjük, ha a 2.8. ábra 4. sorát a következővel helyettesítjük:

```
4) nem CHAR(1) CHECK (nem IN ('F', 'N'))
```

Megjegyezzük, hogy az ('F', 'N') kifejezés egy egyoszlopos reláció két sorát jelenti. A megszorítás tehát azt írja elő, hogy a nem komponens értéke ennek a halmaznak az eleme kell legyen. □

7.7. példa. Azt gondolhatnánk, hogy egy idegenkulcs-megszorítást szimulálni tudunk egy attribútumra vonatkozó CHECK feltétellel, amely megköveteli a hivatkozott érték létezését. A következő egy *hibás* kísérlet annak megkövetelésére, hogy a

```
Stúdió(név, cím, elnökAzon)
```

reláció soraiban szereplő elnökAzon értékeknek elő kell fordulniuk a

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

reláció azonosító attribútumában.

Tegyük fel, hogy a 7.1. ábra 4. sorát a következővel helyettesítjük:

```
4) elnökAzon INT CHECK
       (elnökAzon IN (SELECT azonosító FROM GyártásIrányító))
```

Ez egy szabályos, attribútumra vonatkozó CHECK feltétel, de vizsgáljuk meg, hogy mit is eredményez pontosan. A Stúdió reláció a GyártásIrányító táblában nem szereplő elnökAzon értékkel történő módosításait a rendszer meg fogja tagadni. Ez majdnem ugyanolyan viselkedés, mint amit az idegen kulcsnál tapasztaltunk, azzal a kivétellel, hogy az attribútumalapú megszorítás az elnökAzon NULL értékeit is visszautasítja, ha nincs NULL értékű azonosító. Ennél fontosabb különbség, hogy ha megváltoztatjuk a GyártásIrányító relációt, például töröljük az egyik stúdió elnökének a sorát, akkor a CHECK megszorítás előtt ez a módosítás rejtve fog maradni. Ezért az ilyen esetekben a törlés még akkor is megengedett lesz, amikor az elnökAzon attribútum alapú CHECK megszorítása megsérül. □

7.2.3. Sorra vonatkozó CHECK feltételek

Egy R tábla soraira vonatkozó megszorítást megadhatunk úgy, hogy a CREATE TABLE utasításban az attribútumok, a kulcsok és az idegen kulcsok deklarációja után megadjuk a CHECK kulcsszót, majd zárójelek között egy feltételt. Ez a feltétel bármi lehet, ami a WHERE után szerepelhet. A rendszer a megadott feltételt az R egy sorára vonatkozó feltételként értelmezi, amelyben R attribútumaira a nevükkel hivatkozhatunk. Hasonlóan azonban az attribútumra vonatkozó CHECK feltételek esetéhez, a feltétel itt is hivatkozhat más relációkra vagy az R reláció más soraira. Ezek a hivatkozások most is alkérdésben szerepelhetnek.

A sorra vonatkozó CHECK feltételeket a rendszer akkor ellenőrzi, amikor egy új sort szúrunk be R-be, vagy amikor R egy sorát módosítjuk. A feltétel minden

A nem teljes körű ellenőrzés hiba vagy lehetőség?

Talán csodálkozunk azon, hogy az attribútumra és a sorra vonatkozó CHECK feltételek esetén miért engedi meg a rendszer azok megsértését, ha azok másik relációra vagy az adott reláció másik soraira hivatkoznak. Ennek az az oka, hogy ezek a megszorítások hatékonyabban valósíthatók meg, mint az általánosabb megszorítások. Az attribútumra vagy a sorra vonatkozó CHECK feltételek esetén a megszorítást csak a beszúrandó vagy módosítandó sorra kell kiértékelnie a rendszernek, míg az önálló megszorításokat minden esetben ki kell értékelni, ha a bennük szereplő bármelyik reláció megváltozik. A megfontolt adatbázis-tervező csak attribútumra vagy sorra vonatkozó feltételeket használ olyan esetben, amikor ezek megsérülésének nem áll fenn a veszélye, egyéb esetekben pedig más módszereket alkalmaz, például az önálló megszorításokat (lásd 7.4. alfejezet) vagy a triggereket (lásd 7.5. alfejezet).

új, illetve módosított sorra kiértékelődik. Ha az eredmény hamis lesz, akkor a feltételt megsértő sorra vonatkozó beszúrás- vagy módosításutasítást a rendszer visszautasítja. Ha azonban a feltétel egy másik relációra hivatkozik egy alkérdésben (ez lehet akár maga az R is) és ennek a másik relációnak a módosítása okozza azt, hogy R egy sorára a feltétel hamis lesz, akkor a rendszer emiatt nem utasítja vissza azt az utasítást. Vagyis az attribútumra vonatkozó CHECK feltételekhez hasonlóan a sorra vonatkozó CHECK feltételek is láthatatlanok más relációk számára. Így még egy R-ből történő törlés is vezethet a feltétel hamissá válásához, ha az R szerepelt valamilyen alkérdésben.

Másrészt, ha egy sorra vonatkozó CHECK feltétel csak az ellenőrizendő sor attribútumait tartalmazza, és nincs benne alkérdés, akkor a feltétel mindig igaz marad. Nézzünk egy egyszerű, sorra vonatkozó CHECK feltételt, amelyik egy sor több attribútumát is tartalmazza.

7.8. példa. Idézzük fel a 2.3. példában szereplő `FilmSzínész` tábla sémadeklarációját. A 7.3. ábra megismétli a `CREATE TABLE` utasítást és kiegészíti azt egy elsődleges kulcs deklarációval, valamint egy további megszorítással, amelyik egy lehetséges „következetességi feltétel", amelyet szeretnénk ellenőrizni. Ez a feltétel azt mondja ki, hogy ha egy színész neme férfi, akkor a neve nem kezdődhet `'Ms.'`-szel.

A 2. sorban a `név` attribútumot a reláció elsődleges kulcsaként adtuk meg, a 6. sor pedig a megszorítást deklarálja. A megszorítás feltétele igaz minden nőnemű filmszínészre és minden olyan színészre, akinek a neve nem `'Ms.'`-szel kezdődik. A feltétel azokra a sorokra lesz hamis, amelyekben a `nem` oszlop értéke férfi és a név `'Ms.'`-szel kezdődik. Éppen ezeket a sorokat szerettük volna kizárni a relációból. □

A megszorítások helyes megfogalmazása

Sokszor van szükségünk olyan megszorításra, amely hasonlít a 7.8. példában szereplőhöz, vagyis ahol olyan sorok előfordulását szeretnénk kizárni, amelyek két vagy több feltételnek is eleget tesznek. Ilyenkor a CHECK kulcsszót a feltételek tagadásainak kell követnie, közöttük az OR kulcsszót megadva. Ez az átalakítás egyike a „De Morgan-szabályoknak": a konjunkció tagadása azonos az állítások tagadásainak diszjunkciójával. A 7.8. példában az első feltétel az lenne, hogy a színész férfi legyen. Ezért használtuk a nem = 'N' feltételt, mint ennek tagadását. (Használhattuk volna a nem <> 'F' feltételt is.) A második feltétel, hogy a név 'Ms.'-szel kezdődjön, aminek a tagadására a NOT LIKE összehasonlítást használtuk. Ez az összehasonlítás magát a feltételt tagadja, ami eredetileg az SQL-ben név LIKE 'Ms.%' lett volna.

```
1)  CREATE TABLE FilmSzínész (
2)     név CHAR(30) PRIMARY KEY,
3)     cím VARCHAR(255),
4)     nem CHAR(1),
5)     születésiDátum DATE,
6)     CHECK (nem = 'N' OR név NOT LIKE 'Ms.%')
    );
```

7.3. ábra. Egy megszorítás a FilmSzínész táblára

7.2.4. A sor- illetve attribútum-alapú megszorítások összehasonlítása

Ha egy sorra vonatkozó megszorítás a sor több attribútumát is tartalmazza, akkor soralapú megszorításként kell megfogalmazni. Amennyiben csak a sor egyetlen attribútumára vonatkozik, akkor megírható akár sor-, akár attribútumalapú formában is. Egyik esetben sem számoljuk az alkérdésben szereplő attribútumokat, azaz még az attribútumalapú megszorítás esetén is szerepelhet ugyanazon relációnak több eltérő attribútuma az alkérdésben.

Ha csak a sor egy attribútumát érinti (nem számolva az alkérdéseket) az elvárás, akkor a feltétel ellenőrzése ugyanúgy zajlik, függetlenül attól, hogy sor- vagy attribútumalapú megszorítást írtunk. Noha a soralapú megszorítás gyakrabban kerül ellenőrzésre, mint az attribútumalapú. Ugyanis a soralapú megszorítás esetén a feltétel a sor bármely megfelelő attribútumértékének a megváltozásakor ellenőrzésre kerül, míg az attribútumalapúnál csak az adott attribútum értékének megváltozásakor.

7.2.5. Feladatok

7.2.1. feladat. Adjuk meg a következő megszorításokat a

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

reláció attribútumaira vonatkozóan:

a) Az év nem lehet 1915-nél korábbi.

b) A hossz nem lehet 60-nál kisebb és 250-nél nagyobb.

c) A stúdió neve csak Disney, Fox, MGM vagy Paramount lehet.

7.2.2. feladat. Adjuk meg a következő megszorításokat az 2.4.1. példában szereplő séma attribútumaira vonatkozóan:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, szín, típus, ár)
```

a) Egy laptop sebessége nem lehet kisebb 2.0-nál.

b) A nyomtatók típusa csak lézer, tintasugaras vagy buborék lehet.

c) A termékek típusa csak PC, laptop vagy nyomtató lehet.

! *d)* Egy termék modellje egy PC, egy laptop vagy egy nyomtató modellje kell hogy legyen.

7.2.3. feladat. Írjuk meg a következő megszorításokat sorra vonatkozó `CHECK` feltételek formájában az alábbi relációkra vonatkozóan:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

Ha a megszorítás két relációt is érint, akkor írjunk mindkét relációhoz megfelelő megszorítást, hogy bármelyik relációra vonatkozó beszúrás vagy módosítás esetén a rendszer ellenőrizze a feltételt. Tegyük fel, hogy törlések nem fordulhatnak elő. A sorra vonatkozó `CHECK` feltételekkel megadott megszorítások teljesülését törlések esetén nem lehet biztosítani.

a) Egy filmszínész nem szerepelhet olyan filmben, amelyet a születése előtt készítettek.

! *b)* Két stúdiónak nem lehet azonos a címe.

! *c)* Egy név, amelyik benne van a `FilmSzínész` táblában, nem szerepelhet a `GyártásIrányító` táblában is.

! *d)* A `Stúdió` táblában szereplő stúdióneveknek a `Filmek` tábla legalább egy sorában szerepelniük kell.

!! *e)* Ha egy film producere egyben egy stúdió elnöke is, akkor neki kell lennie a filmet készítő stúdió elnökének is.

7.2.4. feladat. Adjuk meg a következő megszorításokat sorra vonatkozó `CHECK` feltételek formájában PC-k sémájára vonatkozóan:

a) Egy 2.0 GHz-nél kisebb sebességű PC ára nem lehet több 600 dollárnál.

b) Egy olyan laptopnak, amelynek képernyője kisebb, mint 15 hüvelyk, vagy legalább 40 gigabájtos merevlemezzel kell rendelkeznie, vagy 1000 dollárnál kevesebbe kell kerülnie.

7.2.5. feladat. Adjuk meg a következő megszorításokat sorra vonatkozó `CHECK` feltételek formájában a csatahajó sémájára:

```
Hajóosztályok(hajóosztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, hajóosztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

a) Egyik hajóosztálynak sem lehetnek 16 hüvelyknél nagyobb kaliberű ágyúi.

b) Ha egy hajóosztályban az ágyúk száma több mint 9, akkor azok kalibere nem lehet nagyobb 14 hüvelyknél.

! *c)* Egyik hajó sem csatázhat a felavatása előtt.

! **7.2.6. feladat.** A 7.6., illetve 7.8. példákban a `FilmSzínész` `nem` attribútumára tettünk megszorításokat. Milyen megkötéssel érhetnénk el, hogy mindkét megszorítást ki lehessen kényszeríteni, ha a `nem` értéke `NULL`?

7.3. Megszorítások módosítása

A megszorításokat bármikor módosíthatjuk, törölhetjük vagy újakat hozhatunk létre. Hogy ezeket a módosításokat pontosan hogyan kell megadnunk, az attól függ, hogy a megszorítás attribútumhoz, táblához vagy adatbázissémához van-e hozzárendelve. Ez utóbbi esetet majd a 7.4. alfejezetben tárgyaljuk.

7.3.1. Megszorítások elnevezése

Ahhoz, hogy módosítani vagy törölni tudjunk egy létező megszorítást, szükséges, hogy a megszorításnak neve legyen. Ezt úgy adhatjuk meg, hogy a megszorítás elé beírjuk a CONSTRAINT kulcsszót és a megszorítás nevét.

7.9. példa. A 2.9. ábra 2. sora helyett, ami azt mondja ki, hogy a név attribútum elsődleges kulcs, a következőt írhatjuk:

```
2)  név CHAR(30) CONSTRAINT NévKulcs PRIMARY KEY,
```

Hasonlóképpen nevezhetjük el a 7.6. példában szereplő attribútumra vonatkozó CHECK feltételt:

```
4)  nem CHAR(1) CONSTRAINT FérfiVagyNő
                    CHECK (nem IN ('F', 'N')),
```

Végül pedig a következő módon nevezhetjük el a 7.3. ábra 6. sorában szereplő, sorra vonatkozó CHECK feltételt:

```
6)  CONSTRAINT Titulus
        CHECK (nem  = 'N' OR név NOT LIKE 'Ms.%');
```

□

7.3.2. Táblákra vonatkozó megszorítások megváltoztatása

Említettük a 7.1.3. alfejezetben, hogy egy megszorítás ellenőrzését azonnaliról késleltetettre vagy késleltetettről azonnalira állíthatjuk a SET CONSTRAINT utasítással. További módosításokat is tehetünk a megszorításokkal kapcsolatban az ALTER TABLE utasítással. Korábban a 2.3.4. alfejezetben már volt szó az ALTER TABLE utasításról, ott attribútumok hozzáadására, illetve törlésére használtuk ezt az utasítást.

Az ALTER TABLE utasítások többféleképpen is megváltoztathatják a megszorításokat. Egy megszorítás törlése úgy történik, hogy az utasításon belül megadjuk a DROP kulcsszót és a megszorítás nevét. Egy újabb megszorítás megadásához az ADD kulcsszót kell megadnunk, majd magát a megszorítást. Megjegyezzük azonban, hogy a hozzáadott megszorítás olyan típusú kell legyen, amely sorokra vonatkozik, ilyen a soralapú, a kulcs- vagy az idegenkulcs-megszorítás. Sőt megjegyezzük azt is, hogy egy újabb megszorítást csak akkor adhatunk hozzá a táblához, ha a tábla aktuális előfordulása kielégíti a megszorítás feltételét.

7.10. példa. Nézzük meg, hogyan törölhetnénk, majd vihetnénk fel újra a 7.9. példában szereplő megszorításokat a FilmSzínész relációra vonatkozóan. A következő három utasítás törli azokat:

```
ALTER TABLE FilmSzínész DROP CONSTRAINT NévKulcs;
ALTER TABLE FilmSzínész DROP CONSTRAINT FérfiVagyNő;
ALTER TABLE FilmSzínész DROP CONSTRAINT Titulus;
```

Nevezzük el a megszorításokat

Jegyezzük meg, hogy célszerű a megszorításoknak nevet adni még akkor is, ha úgy gondoljuk, hogy soha nem fogunk azokra hivatkozni. Ha egyszer létrehoztuk a megszorítást név nélkül, később már nem fogunk tudni nevet adni neki akkor sem, ha változtatni szeretnénk rajta. Ha azonban egyszer mégis szükségünk lesz rá, hogy egy név nélküli megszorítást módosítsunk, azt fogjuk látni, hogy az adatbázis-kezelő lehetővé teszi, hogy megkeressük ezt a megszorítást az összes megszorítások listájából. Mindezt úgy teszi, hogy valójában ezeknek is ad egy belső, saját maga által kiadott nevet, amely nevet használhatunk, ha a megszorításra szeretnénk hivatkozni.

Ha szeretnénk ismét érvényre juttatni a fenti megszorításokat, ezt a `FilmSzínész` reláció sémájának megváltoztatásával a következőképpen tehetjük meg:

```
ALTER TABLE FilmSzínész ADD CONSTRAINT NévKulcs
    PRIMARY KEY (név);
ALTER TABLE FilmSzínész ADD CONSTRAINT FérfiVagyNő
    CHECK (nem IN ('F', 'N'));
ALTER TABLE FilmSzínész ADD CONSTRAINT Titulus
    CHECK (nem = 'N' OR név NOT LIKE 'Ms.%');
```

Ezek a most újból létrehozott megszorítások mind sorra vonatkozó `CHECK` feltételek. Attribútumra vonatkozó `CHECK` feltételek formájában nem is tudnánk ezeket újból létrehozni.

Az újból létrehozott megszorításokra a megszorítás nevének megadása nem kötelező. Az SQL azonban nem jegyzi meg, hogy korábban melyik megszorítás melyik névhez tartozott. Ezért amikor ismét létrehozunk egy korábbi megszorítást, akkor ismét meg kell adnunk a megszorítás teljes leírását, és nem hivatkozhatunk rá csupán csak a korábbi nevével. □

7.3.3. Feladatok

7.3.1. feladat. Mutassuk meg, hogyan kell a filmadatbázis relációsémáit módosítani a következő célok eléréséhez:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

a) Adjuk meg a `filmcím` és `év` attribútumokat a `Filmek` tábla kulcsaként.

b) Követeljük meg azt a hivatkozásiépség-megszorítást, hogy a filmek producereinek szerepelniük kell a `GyártásIrányító` táblában.

c) Írjuk elő, hogy egy film ne lehessen rövidebb 60 percnél és ne lehessen hosszabb 250 percnél.

! *d)* Zárjuk ki, hogy egy név egyidejűleg előfordulhasson a filmszínészek és a gyártásirányítók között.

! *e)* Ne engedjük meg, hogy két stúdiónak azonos legyen a címe.

7.3.2. feladat. Mutassuk meg, hogyan kell a csatahajók adatbázissémáit megváltoztatni, hogy abban szerepeljenek a következő sorra vonatkozó `CHECK` feltételek:

```
Hajóosztályok(hajóosztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, hajóosztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

a) A `hajóosztály` és `ország` legyen kulcsa a `Hajóosztályok` relációnak.

b) Követeljük meg azt a hivatkozásiépség-megszorítást, hogy a `Csaták` táblában szereplő hajók szerepeljenek a `Hajók` táblában is.

c) Írjuk elő azt a hivatkozásiépség-megszorítást, hogy a `Kimenetelek` táblában szereplő hajók szerepeljenek a `Hajók` táblában is.

d) Írjuk elő, hogy egy hajónak se lehessen 14-nél több ágyúja.

! *e)* Zárjuk ki, hogy egy hajó a felavatása előtt csatában vehessen részt.

7.4. Önálló megszorítások

Az SQL-beli aktív elemek közül a leghatékonyabbak nincsenek hozzárendelve sem sorokhoz, sem azok komponenseihez. Ezek a triggerek és az önálló megszorítások. Ezek részei az adatbázissémának és a táblákhoz kötődnek.

- Az önálló megszorítás egy logikai értékkel rendelkező SQL-kifejezés, amelynek mindig igaznak kell lennie.
- A trigger egy műveletsorozat, amely hozzá van rendelve bizonyos eseményekhez, mint amilyen például egy táblába való beszúrás, és ha az esemény bekövetkezik, akkor a műveletsorozat végrehajtásra kerül.

Az önálló megszorítások használata könnyebb a programozó számára, hiszen itt csak azt kell megmondania, hogy mi az a feltétel, amelynek igaznak kell lennie. Az adatbázis-kezelők mégis inkább a triggert kínálják általános célú aktív elemként. Ennek az az oka, hogy az önálló megszorításokat meglehetősen nehéz hatékonyan megvalósítani. Ez esetben ugyanis az adatbázis-kezelőnek kell kikövetkeztetnie, hogy melyek azok a módosítások, amelyek érinthetik az állítás igazságértékét. A triggerek ezzel szemben pontosan megmondják, hogy mikor kell a rendszernek foglalkoznia velük.

7.4.1. Önálló megszorítások létrehozása

Az SQL-szabvány egy egyszerű formáját javasolja az önálló megszorításoknak, amely bármilyen feltétel ellenőrzését – ami a `WHERE` után megengedett – lehetővé teszi számunkra. Hasonlóan a többi sémaelemhez, ezeket is egy `CREATE` utasítással hozzuk létre. Az utasítás formája a következő:

`CREATE ASSERTION` <az önálló megszorítás neve> `CHECK` (<feltétel>)

Egy önálló megszorításban megadott feltételnek mindig igaznak kell lennie, és bármilyen adatbázis-módosítást, ami a feltétel megsértését eredményezné, a rendszer visszautasít.[1] Emlékeztetünk rá, hogy az eddig tárgyalt `CHECK` feltételek bizonyos körülmények között megsérülhetnek, ha azok alkérdéseket is tartalmaznak.

7.4.2. Önálló megszorítások használata

Máshogyan kell megadnunk a sorra vonatkozó `CHECK` feltételeket és az önálló megszorításokat. A soralapú ellenőrzések direkt módon hivatkozhatnak annak a relációnak az attribútumaira, amelynek deklarációjában szerepelnek. Az önálló megszorítások esetén viszont nincs ilyen jogosultságunk. Ez utóbbiaknál minden, a feltételben szereplő attribútumra magában a megszorításban kell megadnunk, hogy melyik táblában van. Ez általában a táblának egy select-from-where kifejezésben való szerepeltetésével történik.

Mivel a feltételnek egy logikai értéket kell szolgáltatnia, ezért szokásos a feltétel eredményeit valamilyen módon kombinálni, hogy egyetlen igaz vagy hamis értéket kapjunk. Például megadhatjuk a feltételt úgy, hogy az egy relációt adó kifejezés legyen, és erre alkalmazzuk a `NOT EXISTS` műveletet. Vagyis a megszorítás azt jelenti, hogy ez a reláció mindig üres. Egy másik lehetőség, hogy valamilyen összesítő műveletet – például a `SUM` műveletet – alkalmazzuk egy reláció egy oszlopára, és az eredményt összehasonlítjuk egy konstanssal. Ilyen módon előírhatjuk például, hogy az összeg mindig kisebb legyen valamilyen korlátnál.

[1] Emlékezzünk azonban vissza a 7.1.3. alfejezetre, amelyben a megszorítás ellenőrzését elhalaszthattuk a hozzá tartozó tranzakció jóváhagyásáig. Ha egy önálló megszorításra is megtehetnénk ugyanezt, akkor röviden mondva a tranzakció végére a feltétele hamissá válhatna.

7.11. példa. Tegyük fel, hogy azt szeretnénk kikötni, hogy senki ne lehessen egy stúdió elnöke, ha a nettó bevétele nem éri el a 10 000 000 dollárt. Létrehozunk egy önálló megszorítást, amelyik azt mondja ki, hogy azon filmstúdiók halmaza, amelyekre az elnök nettó bevétele kisebb, mint 10 000 000 dollár, üres. Ez az önálló megszorítás az alábbi két relációt foglalja magába:

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

Az önálló megszorítást a 7.4. ábrán láthatjuk. □

```
CREATE ASSERTION GazdagElnök  CHECK
    (NOT EXISTS
         (SELECT *
          FROM Stúdió, GyártásIrányító
          WHERE elnökAzon = azonosító AND
                            nettóBevétel < 10000000
         )
    );
```

7.4. ábra. Önálló megszorítás, amelyik a stúdióelnökök gazdagságát írja elő

7.12. példa. Vegyük a következő önálló megszorítást, amelyik csak egy relációt, a

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

relációt érint. A megszorítás azt mondja ki, hogy egy stúdió által készített filmek összhosszúsága nem haladhatja meg a 10 000 percet.

```
CREATE ASSERTION ÖsszHossz CHECK (10000 >= ALL
    (SELECT SUM(hossz) FROM Filmek GROUP BY stúdióNév)
);
```

A fenti megszorítás csak a `Filmek` relációra mond ki feltételt. Az önálló megszorítás használata helyett ezt kifejezhettük volna egy sorra vonatkozó `CHECK` feltétellel is. Ehhez a következő sorokat kellett volna a `Filmek` reláció sémadeklarációjához hozzáadnunk:

```
CHECK (10000 >= ALL
    (SELECT SUM(hossz) FROM Filmek GROUP BY stúdióNév));
```

Figyeljük meg, hogy ez utóbbi feltétel tulajdonképpen a `Filmek` tábla minden egyes sorára vonatkozik. A fenti megfogalmazásban egyetlen attribútum neve sem szerepel konkrétan, hanem magát a feltételt az alkérdés biztosítja.

Megszorítások összehasonlítása

Az alábbi táblázatban összefoglaltuk a legfontosabb különbségeket az attribútumra vonatkozó CHECK feltételek, a sorra vonatkozó CHECK feltételek és az önálló megszorítások között.

Megszorítás típusa	*Hol deklaráljuk?*	*Mikor ellenőrzi a rendszer?*	*Minden esetben igaz marad?*
Attribútumra vonatkozó CHECK feltétel	Az attribútum megadásakor	A relációba való beszúráskor vagy az attribútum módosításakor	Alkérdés esetén nem
Sorra vonatkozó CHECK feltétel	A relációséma megadásakor	A relációba való beszúráskor vagy egy sor módosításakor	Alkérdés esetén nem
Önálló megszorítás	Az adatbázisséma megadásakor	Bármelyik, a feltételben szereplő reláció megváltozása esetén	Igen

Azt is fontos megjegyezni, hogy ha a fenti ellenőrzést sorra vonatkozó CHECK feltétel alkalmazásával valósítanánk meg, akkor a feltételt a rendszer nem ellenőrizné a `Filmek` táblából való törlés esetén. Ebben a konkrét esetben ez nem is okoz problémát, hiszen ha a megszorítás feltétele a törlés előtt teljesült, akkor biztosan teljesülni fog a törlés után is. Ha azonban a megszorítás a filmek összhosszúságára vonatkozóan nem felső korlátot adna meg, ahogyan az most a példában szerepelt, hanem alsó korlátot, akkor a feltétel megsérülhetne, ha azt sorra vonatkozó CHECK feltétel segítségével valósítanánk meg. Önálló megszorítás használata esetén ez nem fordulhat elő. □

Végül megjegyezzük, hogy az önálló megszorításokat el is dobhatjuk. Az e célra szolgáló utasítás az egyéb adatbázissémabeli elemeket eldobó utasítások mintájára adható meg a következőképpen:

DROP ASSERTION <önálló megszorítás neve>

7.4.3. Feladatok

7.4.1. feladat. Adjuk meg a következő önálló megszorításokat. Használjuk a 2.4.1. feladat PC-termékek adatbázissémáját:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, szín, típus, ár)
```

a) A PC-k gyártói laptopokat is készíthetnek.

b) A PC-k gyártói legalább akkora sebességű processzorral rendelkező laptopokat készíthetnek, mint a PC-ké.

c) Ha egy laptopnak több memóriája van, mint egy PC-nek, akkor az ára is legyen magasabb, mint a szóban forgó PC-nek.

d) Ha a `Termék` relációban szerepel egy modell és a modell típusa, akkor ez a modell elő kell forduljon a megfelelő típushoz tartozó relációban is.

7.4.2. feladat. Fogalmazzuk meg az alábbiakat önálló megszorításként. Használjuk a 2.4.3. feladat csatahajó-adatbázis sémáját:

```
Hajóosztályok(hajóosztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, hajóosztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

a) Egyik osztályban sem lehet 2-nél több hajó.

! *b)* Egyetlen országnak sem lehet csatahajója és csatacirkálója is egyszerre.

! *c)* Egyetlen 9-nél több ágyúval rendelkező hajó sem süllyedhet el 9-nél kevesebb ágyúval rendelkező hajóval folytatott csatában.

! *d)* Egyetlen hajót sem indíthatnak el az osztályának nevet adó, első hajó előtt.

! *e)* Minden osztályra van olyan hajó, amely az osztály nevét viseli.

! **7.4.3. feladat.** A 7.11. példában szereplő önálló megszorítás megfogalmazható két sorra vonatkozó megszorításként is. Mutassa meg, hogyan.

7.5. Triggerek

A *triggerek*, amelyeket szokás *esemény-feltétel-művelet szabályok*nak is nevezni, a korábban tárgyalt megszorításoktól három dologban térnek el.

1. A triggereket a rendszer csak akkor ellenőrzi, ha bizonyos, az adatbázis-programozó által megadott *események* bekövetkeznek. A megengedett események általában egy adott relációra vonatkozó beszúrás, törlés és módosítás. Egy másikfajta esemény, amit sok SQL-rendszer megenged, a tranzakció befejeződése.

2. Egy kiváltó esemény bekövetkezésekor a trigger egy *feltételt* vizsgál meg. Ha a feltétel nem teljesül, akkor a kiváltó eseményre válaszul a triggerrel összefüggésben semmi nem fog történni.

3. Ha a trigger feltétele teljesül, akkor a rendszer végrehajtja a triggerhez tartozó *műveletet*. Ez egy lehetséges művelet az esemény hatásának valamilyen módon történő megváltoztatásra. Ez akár az eseményhez tartozó tranzakció elvetése is lehet. A megadott művelet azonban adatbázis-műveleteknek egy tetszőleges sorozata lehet, beleértve az olyan műveleteket is, amelyeknek semmi köze sincs a kiváltó eseményhez.

7.5.1. Az SQL triggerei

Az SQL nyelv triggerutasítása számos különböző lehetőséget kínál a felhasználónak az eseményre, a feltételre és a műveletre vonatkozó részében. Az alábbiakban felsoroljuk a legfontosabbakat.

1. A *triggerfeltétel* ellenőrzésének és a *trigger* műveletének végrehajtása történhet akár a kiváltó esemény végrehajtása előtt létező *adatbázis-állapot*ban, azaz a relációk aktuális előfordulására nézve, vagy abban az állapotban, amely a kiváltó esemény után áll fenn.

2. A művelet és a feltétel hivatkozhat a műveletet kiváltó esemény által módosított sorok régi és/vagy új értékeire is.

3. Olyan módosítási esemény is megadható, amely csak egy adott attribútumra vagy attribútumhalmazra vonatkozik.

4. A programozó megadhatja, hogy a művelet végrehajtása a következő két lehetőség közül melyik módon történjen meg:

   a) minden módosított sorra egyszer *(sor szintű trigger)* vagy

   b) egy SQL-módosító utasítás által módosított összes sorra vonatkozóan egyszer (*utasítás szintű trigger*, emlékeztetve arra, hogy egy SQL módosító utasítás több sorra is hatással lehet).

Mielőtt a triggerek szintaktikai részleteire rátérnénk, nézzünk meg egy példát, amely rávilágít a legfontosabb szintaktikai és szemantikai szempontokra. Figyeljük meg a 7.5. ábrán szereplő példatrigger kulcsfontosságú elemeit, illetve ezeknek az előfordulási sorrendjét:

a) A `CREATE TRIGGER` utasítás (1. sor).

b) Egy záradék, amely megfogalmazza a kiváltó eseményt, illetve azt is, hogy a trigger a kiváltó esemény végrehajtása előtti vagy utáni adatbázis-állapotot fogja használni (2. sor).

c) Egy `REFERENCING` záradék, hogy lehetővé tegyük a trigger feltételének és műveletének a módosított sorokra történő hivatkozást (3–5. sorok). Egy ehhez hasonló módosítás esetén ez a záradék lehetővé teszi, hogy a módosítás előtti, illetve utáni soroknak nevet adhassunk.

d) Egy záradék, amely azt fejezi ki, hogy egy SQL-utasítás esetén a trigger minden módosított sorra egyszer vagy az összes módosított sorra egyszer hajtódjon-e végre (6. sor).

e) Egy feltétel a `WHEN` kulcsszó használatával, amely egy logikai kifejezést tartalmaz (7. sor).

f) Egy vagy több SQL-utasítást tartalmazó művelet (8–10. sorok).

Mindegyik fenti elemnek vannak opciói, amelyeket a példát követően ismertetni fogunk.

7.13. példa. A 7.13. ábrán megadunk egy SQL-beli triggert, amelyik a

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

táblára vonatkozik. A kiváltó esemény a `nettóBevétel` attribútum módosítása. A trigger hatása az lesz, hogy ha valaki megpróbálja csökkenteni a gyártásirányítók nettó bevételét, akkor a művelet ezt meghiúsítja, és a módosítás előtti állapotot hozza ismét létre.

Az 1. sor a `CREATE TRIGGER` kulcsszót és a trigger nevét tartalmazza. A 2. sor adja meg a kiváltó eseményt, amely most a `GyártásIrányító` tábla `nettóBevétel` attribútumának módosítása. A 3. sortól az 5. sorig terjedő rész lehetővé teszi a triggerbeli feltétel és művelet rész számára, hogy hivatkozni tudjanak a módosítás előtti régi, és a módosítás utáni új értékekre is. A módosítás előtti sorokra `RégiSor` néven, a módosítás utániakra `ÚjSor` néven lehet majd hivatkozni, ahogy azt a 4. és 5. sorban láthatjuk. A feltételben és a műveletben ezekre ugyanúgy hivatkozhatunk, mintha azok egy szokásos SQL-lekérdezés `FROM` záradékában megadott sorváltozók lennének.

A 6. sor `FOR EACH ROW` fejezi ki azt a követelményt, hogy a műveletek végrehajtása minden egyes sorra külön-külön történjen meg. A 7. sor a triggerfeltétel része. Azt mondja ki, hogy a műveletet csak akkor fogjuk végrehajtani, ha a

```
1)  CREATE TRIGGER NetBevétTrigger
2)  AFTER UPDATE OF nettóBevétel ON GyártásIrányító
3)  REFERENCING
4)      OLD ROW AS RégiSor,
5)      NEW ROW AS ÚjSor
6)  FOR EACH ROW
7)  WHEN (RégiSor.nettóBevételI > ÚjSor.nettóBevétel)
8)      UPDATE GyártásIrányító
9)      SET nettóBevétel = RégiSor.nettóBevétel
10)     WHERE azonosító = ÚjSor.azonosító ;
```

7.5. ábra. Egy SQL-beli trigger

régi nettó bevétel nagyobb, mint az új, vagyis az illető személy nettó bevétele csökkent.

A 8–10. sorok alkotják a triggerművelet részét. Ezek a sorok egy szabályos SQL-beli `UPDATE` utasítást alkotnak, amely utasításnak az a hatása, hogy az adott személy nettó bevételét visszaállítja a módosítás előtti értékre. Figyeljük meg, hogy az utasítás a `GyártásIrányító` tábla összes sorát érintené, de a 10. sorban szereplő `WHERE` záradék biztosítja, hogy csak az a korábbi módosításban szereplő sor legyen érintve, amelyik a megfelelő `azonosító`-val rendelkezik. □

7.5.2. A trigger szerkesztésének lehetőségei

Természetesen a 7.13. példa csak néhány sajátosságát mutatja be az SQL triggereinek. A következőkben azokat a lehetőségeket fogjuk felvázolni, amelyeket a triggerek kínálnak a számunkra, és megmutatjuk azt is, hogyan lehet ezeket a lehetőségeket kifejezni.

- A 7.5. ábra 2. sora azt mondja ki, hogy a szabályban szereplő feltételellenőrzést, illetve műveletet a kiváltó esemény után kell végrehajtani. Ezt az `AFTER` kulcsszó jelzi. Az `AFTER` helyett a `BEFORE` kulcsszót is használhatnánk, ez esetben a `WHEN` feltétel a kiváltó esemény előtt, vagyis a triggert kiváltó módosítás előtt kerülne ellenőrzésre. Ha a feltétel igaz, a trigger művelete(i) végrehajtódnak. Végül ezután hajtódna végre a kiváltó esemény függetlenül attól, hogy a feltétel igaz-e. Van egy harmadik lehetőség is, az `INSTEAD OF` használata, amelyet a 8.2.3. alfejezetben tárgyalunk a nézetek módosításával kapcsolatban.

- A módosítás (`UPDATE`) mellett további lehetséges kiváltó események még a beszúrás (`INSERT`) és a törlés (`DELETE`). A 7.5. ábra 2. sorában szereplő `OF nettóBevétel` záradék opcionálisan megadható, és a megadása azt jelenti, hogy csak az `OF` kulcsszó után felsorolt attribútumok módosítása (`UPDATE`) számít kiváltó eseménynek. Az `OF` záradék beszúrás (`INSERT`) és törlés

(DELETE) események esetén nem adható meg, hiszen ezek az események mindig teljes sorokra vonatkoznak.

- A WHEN megadása opcionális. Ha nem adjuk meg, a művelet minden esetben végrehajtódik, amikor a trigger aktivizálódik. Ha megadtuk, a művelet pontosan akkor hajtódik végre, amikor a WHEN feltétele teljesül.

- A példában csak egyetlen SQL-utasítás szerepelt a művelet részben, de megadható ott több ilyen utasítás is pontosvesszőkkel elválasztva, BEGIN és END közé téve.

- Ha egy sor szintű trigger kiváltó eseménye módosítás, akkor a módosítás előtti sort régi sornak, a módosítás utánit pedig új sornak tekinthetjük. Ezeknek a soroknak nevet adhatunk az OLD ROW AS, illetve a NEW ROW AS záradékok segítségével, ahogyan az a 7.5. ábra 4. és 5. soraiban látható. Ha a kiváltó esemény beszúrás, akkor a beszúrt sornak a NEW ROW AS záradék segítségével adhatunk nevet, az OLD ROW AS záradék viszont ilyenkor nem használható. Törlés esetén az OLD ROW AS záradék segítségével nevezhetjük el a törölt sort, ilyenkor viszont a NEW ROW AS záradék nem használható.

- Ha elhagyjuk vagy kicseréljük az alapértelmezett FOR EACH STATEMENT kulcsszóra a 6. sorban szereplő FOR EACH ROW megkötést, akkor egy sor szintű triggerből (mint amilyen például a 7.5. ábrán is szerepel) utasítás szintű trigger lesz. Egy utasítás szintű trigger egyszer hajtódik végre kiváltó utasításonként, függetlenül attól, hogy a kiváltó utasítás hány sort érint (nullát, egyet vagy többet). Ha például egy egész táblát módosítunk egy SQL-beli UPDATE utasítással, akkor egy utasítás szintű trigger egyszer fog végrehajtódni, míg egy sor szintű trigger minden módosított sorra végrehajtásra kerül.

- Egy utasítás szintű triggerben nem hivatkozhatunk közvetlenül a régi és az új sorokra, ahogy azt a 4. és 5. sorban láttuk. Bármely triggerben (utasítás és sor szintűben is) hivatkozhatunk azonban a *régi sorok* relációjára és az *új sorok* relációjára. Az előbbi a törölt sorokat, illetve a módosított sorok régi verzióit jelenti, az utóbbi pedig a beszúrt sorokat, illetve a módosított sorok új verzióit jelenti. A deklaráció formája ez esetben OLD TABLE AS RégiAdat és NEW TABLE AS ÚjAdat.

7.14. példa. Tegyük fel, hogy azt szeretnénk megakadályozni, hogy a gyártásirányítók nettó bevételének átlaga 500 000 dollár alá csökkenjen. Ezt a

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

táblára vonatkozó megszorítást egy beszúrás, egy törlés vagy a nettóBevétel oszlop módosítása sértheti meg. Vigyáznunk kell, hiszen egy beszúrás vagy módosítás sok sort érinthet. A módosítások végrehajtása közben az átlag átmenetileg 500 000 dollár alá csökken, majd az összes módosítás elvégzése után ismét

afölé megy. Csak azokat a módosításokat szeretnénk visszautasítani, amelyeknek a teljes lefutása után az átlag a megadott határ alá kerül.

A GyártásIrányító táblának mind a három típusú eseményére (beszúrás, módosítás, törlés) egy triggert kell írnunk. A 7.6. ábrán a módosításra vonatkozó triggert láthatjuk. A beszúrásra és a törlésre vonatkozó triggerek hasonlóak.

```
1)  CREATE TRIGGER ÁtlagNetBevétTrigger
2)  AFTER UPDATE OF nettóBevétel ON GyártásIrányító
3)  REFERENCING
4)      OLD TABLE AS RégiAdat,
5)      NEW TABLE AS ÚjAdat
6)  FOR EACH STATEMENT
7)  WHEN (500000 > (SELECT AVG(nettóBevétel) FROM
                                        GyártásIrányító))
8)  BEGIN
9)      DELETE FROM GyártásIrányító
10)     WHERE (név, cím, azonosító, nettóBevétel) IN ÚjAdat;
11)     INSERT INTO GyártásIrányító
12)         (SELECT * FROM OldStuff);
13) END;
```

7.6. ábra. Az átlagos nettó bevétel megszorítása

A 3–5. sorok azt adják meg, hogy annak a relációnak a neve, amelyik a régi sorokat tartalmazza: RégiAdat, annak a neve pedig, amelyik az új sorokat tartalmazza: ÚjAdat. Itt most az, hogy régi, illetve új, azzal az adatbázis-művelettel kapcsolatban értendő, amelyik kiváltotta a triggert. Egy adatbázis-művelet egy relációnak több sorát is módosíthatja, ezért egy ilyen művelet végrehajtása után a RégiAdat és az ÚjAdat tábláknak nagyon sok sora is lehet.

Ha a kiváltó művelet módosítás, akkor a RégiAdat tábla a sorok módosítás előtti régi verzióját, az ÚjAdat tábla pedig a sorok módosítás utáni új verzióját tartalmazza. Ha törlés műveletre írnánk egy hasonló triggert, abban az esetben a törölt sorok lennének a RégiAdat táblában, és az ÚjAdat táblára vonatkozó deklaráció nem szerepelne a trigger leírásában. Beszúrás műveletre megírt triggerben pedig a beszúrt sorok lennének az ÚjAdat táblában, és a RégiAdat táblára vonatkozó deklaráció maradna el.

A 6. sor azt mondja ki, hogy a trigger utasításonként egyszer hajtódjon végre, függetlenül a módosított sorok számától. A 7. sor a feltétel. Ez akkor válik igazzá, ha a módosítás után az átlagos nettó bevétel kisebb lesz 500 000 dollárnál.

A 8–13. sorok két utasítást tartalmaznak, amelyek a régi GyártásIrányító táblát állítják vissza, ha a WHEN feltétel igaz lesz, vagyis amikor az átlagos nettó bevétel túl alacsonnyá válik. A 9–10. sor kitörli az új sorokat, vagyis a módosított sorok új verzióit, a 11–12. sorok pedig ismét beszúrják a módosítás előtti sorokat. □

7.15. példa. A BEFORE triggerek egy fontos alkalmazása, amikor egy beszúrandó sort a beszúrás előtt valamilyen módon a megfelelő formára hoznak. Tegyük fel, hogy filmsorokat kívánunk beszúrni a

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

táblába, de nem mindig ismerjük a film kiadásának évét. Mivel az év a kulcsnak egy attribútuma, ezért értéke nem lehet NULL. Egy trigger segítségével viszont biztosíthatjuk, hogy az év értéke ne legyen NULL. A trigger a NULL helyett valamilyen megfelelő értéket ír be, természetesen ez akár egy bonyolult módon számított érték is lehet. A 7.7. ábrán szereplő trigger a NULL értéket egyszerűen az 1915-tel helyettesíti (ez lehetne akár egy alapértelmezett érték is, de most példának is megteszi).

A 2. sorban szerepel a beszúrás előtt lefutó feltétel és művelet. A 3–5. sorokban, a hivatkozási záradékban definiáljuk a beszúrandó új sor, illetve a – csak ezt a sort tartalmazó – tábla nevét is. Még akkor is, ha a trigger minden beszúrandó sorra csak egyszer fut le (mivel a 6. sor sor szintű triggerként határozza meg a triggert), a 7. sorban lévő feltételnek tudnia kell hivatkozni a beszúrandó sor egy attribútumára, emellett a 8. sorban lévő műveletnek pedig egy táblára kell tudnia hivatkozni a módosítás leírásához. □

```
1)  CREATE TRIGGER ÉvJavítóTrigger
2)  BEFORE INSERT ON Filmek
3)  REFERENCING
4)      NEW ROW AS újSor
5)      NEW TABLE AS újÉrték
6)  FOR EACH ROW
7)  WHEN újSor.év IS NULL
8)  UPDATE újÉrték SET év = 1915;
```

7.7. ábra. A beszúrt sorok NULL értékeinek helyettesítése

7.5.3. Feladatok

7.5.1. feladat. Írjuk meg a 7.6. ábra triggeréhez hasonló SQL-beli triggereket a GyártásIrányító tábla beszúrás és törlés műveletére.

7.5.2. feladat. Írjuk meg a következőket, mint triggereket. Egyik esetben se engedjük meg a módosítás végrehajtását, ha az nem elégíti ki a megadott megszorítást. Az adatbázisséma a 2.4.1. feladat PC adatbázisának megfelelő.

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, szín, típus, ár)
```

a) Amikor egy PC árát módosítjuk, ellenőrizzük, hogy nincs ugyanolyan sebességű, de kisebb árú PC.

b) Amikor egy új nyomtatót viszünk fel, ellenőrizzük, hogy az adott modellszám benne van a `Termék` táblában.

! *c)* A `Laptop` reláció bármilyen módosítása esetén ellenőrizzük, hogy a laptopok átlagos ára gyártónként legalább 1500 dollár legyen.

! *d)* Amikor egy PC memóriáját vagy merevlemezét módosítjuk, ellenőrizzük, hogy a módosított PC-nek legalább 100-szor akkora a merevlemeze, mint a memóriája.

! *e)* Amikor egy új PC-t, laptopot vagy nyomtatót viszünk fel, ellenőrizzük, hogy az adott modellszám még nem szerepel a `PC`, `Laptop` vagy `Nyomtató` táblában.

7.5.3. feladat. Írjuk meg a következő feladatokat triggerek formájában. Egyik esetben se engedjük meg a módosítás végrehajtását, ha az nem elégíti ki a megadott megszorítást. Az adatbázisséma a 2.4.3. feladat csatahajó adatbázisának megfelelő.

```
Hajóosztályok(hajóosztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, hajóosztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

a) Ha egy új hajóosztályt viszünk fel a `Hajóosztályok` táblába, vigyünk fel egy hajót is, amelyiknek ugyanaz a neve, mint a hajóosztálynak, és a felavatási dátuma legyen `NULL` érték.

b) Ha egy új hajóosztályt viszünk fel, amelynek a vízkiszorítása nagyobb 35 000 tonnánál, akkor engedjük meg a beszúrást, de a vízkiszorítást változtassuk 35 000 tonnára.

! *c)* Ha egy új sort viszünk fel a `Kimenetelek` táblába, ellenőrizzük, hogy a hajó és a csata benne van-e a `Hajók`, illetve a `Csaták` táblában, és ha nincs, akkor szúrjuk be a megfelelő sort egyik vagy mindkét táblába, a hiányzó oszlopokba `NULL` értéket téve.

! *d)* Ha új sorokat viszünk fel a `Hajók` táblába, vagy a tábla `hajóosztály` attribútumát módosítjuk, ellenőrizzük, hogy egy országnak se lehessen több mint 20 hajója.

!! *e)* Ellenőrizzük minden lehetséges esetben, ami a feltételt megsértheti, hogy egy hajó nem vehet részt egy csatában, ha egy korábbi dátumú csatában elsüllyesztették.

! **7.5.4. feladat.** Írjuk meg a következő feladatokat triggerek vagy önálló megszorítások formájában. Egyik esetben se engedjük meg a módosítás végrehajtását, ha az nem elégíti ki a megadott megszorítást. A feladatok a filmadatbázis tábláira vonatkoznak.

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

Feltételezhetjük, hogy a kívánt feltétel minden esetben teljesül, mielőtt megpróbáljuk módosítani az adatbázist. Ha lehetséges, akkor az utasítás visszautasítása helyett inkább módosítsuk az adatbázist, még akkor is, ha ez `NULL` értékek vagy alapértelmezés szerinti értékek felvitelével jár.

a) Biztosítsuk azt a feltételt, hogy a `SzerepelBenne` relációban lévő színészek benne vannak a `FilmSzínész` relációban is.

b) Biztosítsuk azt a feltételt, hogy minden gyártásirányító egyben vagy egy stúdió elnöke, vagy egy film producere, vagy mindkettő egyidejűleg.

c) Biztosítsuk azt a feltételt, hogy minden filmnek legalább egy férfi és egy női szereplője is van.

d) Biztosítsuk azt a feltételt, hogy egy stúdió egy évben nem készíthet száznál több filmet.

e) Biztosítsuk azt a feltételt, hogy az ugyanabban az évben készült filmek hosszának átlaga nem lehet több mint 120 perc.

7.6. Összefoglalás

- *Hivatkozásiépség-megszorítások:* Előírhatjuk, hogy egy attribútum vagy egy attribútumhalmaz értékeinek elő kell fordulnia a reláció vagy egy másik reláció valamelyik sorának megfelelő attribútumában vagy attribútumaiban. Ezt a relációséma megadásakor a `REFERENCES` vagy a `FOREIGN KEY` kulcsszóval adhatjuk meg.

- *Attribútumra vonatkozó* `CHECK` *feltételek:* Egy attribútum értékeire vonatkozó megszorítást előírhatunk úgy, hogy a `CHECK` kulcsszót és az ellenőrizendő feltételt megadjuk a relációsémában az attribútum deklarációja után.

- *Sorra vonatkozó* `CHECK` *feltételek:* Egy reláció soraira megadhatunk ellenőrizendő feltételeket úgy, hogy a reláció deklarációja után megadjuk a `CHECK` kulcsszót és a feltételt.

- *Megszorítások módosítása:* Sorra vonatkozó `CHECK` feltételt törölhetünk vagy hozzáadhatunk a táblához, a táblára vonatkozó `ALTER` utasítással.
- *Önálló megszorítások:* Önálló megszorítást az adatbázisséma részeként adhatjuk meg. A deklarációban adjuk meg az ellenőrizendő feltételt. A feltétel vonatkozhat az adatbázisséma egy vagy több relációjára, és hivatkozhat a reláció egészére – például összesítő függvények esetén – vagy az egyes sorokra.
- *Az ellenőrzések kezdeményezése:* Az önálló megszorításokat a rendszer akkor ellenőrzi, amikor a feltételben szereplő relációk valamelyike megváltozik. Az attribútumra és a sorra vonatkozó `CHECK` feltételeket a rendszer csak akkor ellenőrzi, ha az az attribútum vagy az a reláció, amire a feltétel vonatkozik, beszúrás- vagy módosításutasítás hatására változik meg. Így ez utóbbi megszorítások megsérülhetnek, ha alkérdéseket is tartalmaznak.
- *Triggerek:* Az SQL-szabvány a triggereket is tartalmazza, amelyek megadhatnak bizonyos eseményeket, amelyek életre hívják őket. Ilyen esemény lehet a beszúrás, a módosítás vagy a törlés. Amikor a trigger kiváltódik, a rendszer egy feltétel teljesülését ellenőrzi, amelynek igaz volta esetén egy megadott, SQL-utasításokból álló sorozat hajtódik végre.

7.7. Irodalomjegyzék

[5] és [4] áttekinti az adatbázisrendszerek aktív elemeinek minden lényeges kérdését. [1] a legújabb véleményeket tárgyalja az aktív elemek SQL-99-ben és a jövőbeni szabványokban elfoglalt helyéről. [2] és [3] egy HiPAC nevű korai prototípusrendszert mutat be, amely aktív adatbáziselemek használatát tette lehetővé.

[1] R. J. Cochrane, H. Pirahesh, N. Mattos, „Integrating triggers and declarative constraints in SQL database systems," *Intl. Conf. on Very Large Databases*, pp. 567–579, 1996.

[2] U. Dayal et al., „The HiPAC project: combining active databases and timing constraints," *SIGMOD Record* **17**:1, pp. 51–70, 1988.

[3] D. R. McCarthy, U. Dayal, „The architecture of an active database management system," *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pp. 215–224, 1989.

[4] N. W. Paton, O. Diaz, „Active database systems," *Computing Surveys* **31**:1 (March, 1999), pp. 63–103.

[5] J. Widom, S. Ceri, *Active Database Systems*, Morgan-Kaufmann, San Francisco, 1996.

8. fejezet

Nézetek és indexek

Ezt a fejezetet a nézettáblák ismertetésével kezdjük. A nézettábla olyan reláció, melyet más relációkra vonatkozó lekérdezésekkel definiálunk. A nézettáblák az adatbázisban nincsenek tárolva, de ugyanúgy lekérdezhetők, mintha létező táblák lennének. A nézettáblára vonatkozó lekérdezés végrehajtásakor a lekérdezésfeldolgozó a nézettábla definícióját felhasználva látszólagosan előállítja a nézettáblát.

A nézettáblák tárolhatók is abban az értelemben, hogy az adatbázisból periodikusan előállíthatók és tárolhatók is. A tárolt nézettáblákkal a lekérdezések végrehajtási sebessége növelhető. A tárolt nézettáblák nagyon fontos speciális típusa az index, ami olyan tárolt adatstruktúra, amelynek egyedüli célja a tárolt relációk meghatározott sorai elérésének felgyorsítása. E fejezetben be fogjuk mutatni az indexeket, és áttekintjük a tárolt táblákhoz a megfelelő index kiválasztásának legfontosabb kérdéseit.

8.1. Nézettáblák

A `CREATE TABLE` utasítással definiált táblák fizikailag léteznek az adatbázisban, azaz az adatbázisrendszer valamilyen fizikai struktúrában tárolja őket. Megtartják az állapotukat, azaz nem változnak addig, amíg valamilyen táblamódosító SQL-utasítás meg nem változtatja őket.

Létezik az SQL-relációknak egy másik típusa, amit *nézettábla*nak nevezünk, amelyek nem léteznek fizikailag az adatbázisban. A nézettáblákat a lekérdezéshez hasonló kifejezés segítségével definiáljuk. A nézettáblákat ugyanúgy lekérdezhetjük, mint a fizikailag létező táblákat, és egyes esetekben még módosíthatjuk is őket.

Relációk, táblák és nézettáblák

Az SQL-programozók gyakran használják a „tábla" szót a „reláció" szó helyett. Az ok abban rejlik, hogy fontos különbséget tenni a tárolt relációk, azaz a „táblák" és a virtuális relációk, a „nézettáblák" között. Most, hogy már ismerjük a különbséget a tábla és a nézettábla között, a „reláció" megnevezést csak ott fogjuk használni, ahol a táblát és a nézettáblát is használhatnánk. Amikor hangsúlyozni akarjuk, hogy egy reláció tárolt, akkor az „alapreláció" vagy „alaptábla" kifejezéseket fogjuk használni.

Létezik egy harmadik típusú reláció is, amely nem nézettábla és nincs fizikailag tárolva. Ezek az ideiglenes relációk, amelyek valamely lekérdezés eredményeképpen jöttek létre. Ezekről is „relációként" fogunk említést tenni.

8.1.1. Nézettáblák létrehozása

A legegyszerűbb mód egy nézettábla létrehozására a következő:

```
CREATE VIEW <név> AS <definíció>;
```

A nézettábla definíciója egy SQL-lekérdezés.

8.1. példa. Tételezzük fel, hogy egy olyan nézettáblát szeretnénk, amely a

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

reláció egy részét specifikálja, pontosabban a Paramount stúdió által gyártott filmek címét és gyártási évét. Ezt a nézettáblát a következőképpen definiálhatjuk:

```
1)  CREATE VIEW ParamountFilmek AS
2)     SELECT filmcím, év
3)     FROM Filmek
4)     WHERE stúdióNév = 'Paramount';
```

Az 1. sorban látható, hogy a nézettábla neve `ParamountFilmek`. A nézettábla attribútumait a 2. sor tartalmazza: `filmcím` és `év`. A nézettábla definíciója a 2–4. sorok között található meg. □

8.2. példa. Nézzünk egy példát bonyolultabb lekérdezéssel definiált nézettáblára. A célunk a `FilmProducer` nézettábla létrehozása, amely a filmek címét és a producereik nevét tartalmazza. Ezt a nézettáblát a következő két reláció felhasználásával tudjuk definiálni:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

A nézettábla definíciója:

```
CREATE VIEW FilmProducer AS
    SELECT Filmek.filmcím, név
    FROM Filmek, GyártásIrányító
    WHERE producerAzon = azonosító;
```

Fenti definíció két relációt kapcsol össze és az azonosítók egyezését követeli meg. A film címét és a producer nevét a megegyező azonosítójú sorpárokból állítja elő. □

8.1.2. Nézettáblák lekérdezése

A nézettáblák pontosan ugyanúgy kérdezhetők le, mint a ténylegesen tárolt alaptáblák. A FROM záradékban megadjuk a nézettábla nevét, és a nézettábla definícióját használva az adatbázisrendszerre bízzuk, hogy előállítsa a lekérdezéshez szükséges sorokat.

8.3. példa. Úgy kérdezhetjük le a ParamountFilmek relációt, mintha egy tárolt reláció lenne:

```
SELECT filmcím
FROM ParamountFilmek
WHERE év = 1979;
```

Ezzel a lekérdezéssel a Paramount által 1979-ben gyártott filmek címeit kapjuk meg. □

8.4. példa. Írhatunk olyan lekérdezéseket is, amelyek nézettáblákat és alaptáblákat is tartalmaznak. Ilyen például a következő:

```
SELECT DISTINCT színészNév
FROM ParamountFilmek, SzerepelBenne
WHERE filmcím = filmCím AND
          ParamountFilmek.év = SzerepelBenne.filmÉv;
```

Ez a lekérdezés a Paramount által gyártott filmekben szereplő összes színész nevét megadja. □

Azt, hogy mit is jelent a nézettáblák használata, a legegyszerűbben úgy tudjuk megmutatni, hogyha a FROM záradékban a nézettáblákat kicseréljük a nézettáblákat definiáló alkérdéssel. Az alkérdést követő sorváltozóval tudunk az alkérdéssel létrehozott tábla soraira hivatkozni. Például a 8.4. példa lekérdezésének eredménye megegyezik a 8.1. ábrán látható lekérdezés eredményével.

```
SELECT DISTINCT színészNév
FROM (SELECT filmcím, év
      FROM Filmek
      WHERE stúdióNév = 'Paramount'
     ) Pm, SzerepelBenne
WHERE Pm.filmcím = filmCím AND Pm.év = SzerepelBenne.filmÉv;
```

8.1. ábra. A nézettábla használatának alkérdéssel való megvalósítása

8.1.3. Attribútumok átnevezése

Néha szeretnénk más attribútumneveket használni a nézettábla definíciójában, mint amelyek a nézettáblát definiáló lekérdezésből adódnak. A nézettábla attribútumait megadhatjuk egy zárójelek közé írt lista formájában, amelyet a `CREATE VIEW` utasításban a nézettábla neve után kell írni. Például a 8.2. példa nézettáblájának definícióját a következőképpen lehetne átírni:

```
CREATE VIEW FilmProducer(filmCím, producerNév) AS
    SELECT Filmek.filmcím, név
    FROM Filmek, GyártásIrányító
    WHERE  producerAzon = azonosító;
```

A nézettábla ugyanaz lesz, attól eltekintve, hogy az oszlopnevek nem `filmcím` és `név`, hanem `filmCím` és `producerNév` lesznek.

8.1.4. Feladatok

8.1.1. feladat. A következő alaptáblákból kiindulva:

```
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

Építsük fel a következő nézettáblákat:

a) Egy `GazdagProducer` nézettáblát, amely a legalább 10 000 000 $ nettó bevételű gyártásirányítók nevét, címét, azonosító számát és bevételét adja meg.

b) Egy `StúdióElnök` nézettáblát, amely azon gyártásirányítók nevét, címét és azonosító számát tartalmazza, akik stúdióelnökök is.

c) Egy `ProdSzínész` nézettáblát, amely azon személyek nevét, címét, nemét, születési dátumát, azonosító számát és nettó bevételét tartalmazza, akik egyben gyártásirányítók és színészek is.

8.1.2. feladat. A 8.1.1. feladat nézettábláit használva (alaptáblák használata nélkül) adjuk meg a következő lekérdezéseket:

a) Keressük meg azoknak a nőknek a neveit, akik gyártásirányítók és színészek is.

b) Keressük meg azon gyártásirányítók neveit, akik egyben stúdióelnökök is és nettó bevételük legalább 10 000 000 $.

! *c)* Keressük meg azon stúdióelnökök nevét, akik egyben színészek is és nettó bevételük legalább 50 000 000 $.

8.2. Adatok módosítása nézettáblákon keresztül

Korlátozott módon lehetséges beszúrni, törölni vagy változtatni az adatokat egy nézettáblán. Első látásra teljesen értelmetlen ötletnek tűnik, mivel a nézettábla nem létezik abban a formában, amelyben egy alaptábla (tárolt reláció) létezik. Mit jelent például egy új sor beszúrása egy nézettáblába? Hol tároljuk, és honnan emlékezzen majd arra az adatbázis-kezelő, hogy a sor a nézettáblába került?

A legtöbb nézettábla esetén nem engedjük meg az ilyen beszúrást. Eléggé egyszerű nézettáblák esetén azonban a nézettábla módosítását átalakíthatjuk az alaptábla módosításává, amely ugyanolyan hatású lesz, mintha a nézettáblát módosítanánk. Az ilyen nézettáblákat *módosítható nézettáblák*nak nevezzük. Az „instead of" (a kiváltó esemény helyett működő) trigger is használható arra, hogy a nézettábla módosítását az alaptábla módosításával végezze el. Ezzel a programozó meg tudja határozni, hogy a nézettáblán keresztüli módosítás hogyan történjen.

8.2.1. Nézettábla megszüntetése

A nézettábla különleges módosításának tekinthető a nézettábla megszüntetése. Ezt a „változtatást" akkor is végre lehet hajtani, ha a nézettábla nem módosítható. A megszüntetési utasítás:

```
DROP VIEW ParamountFilmek;
```

Ez az utasítás kitörli a nézettábla definícióját, tehát utána már nem kérdezhetjük le és nem módosíthatjuk a nézettáblát. Ugyanakkor a nézettábla megszüntetése nincs kihatással az alaptábla soraira. Ellenben, a

```
DROP TABLE Filmek
```

nemcsak a `Filmek` táblát szünteti meg, hanem a `ParamountFilmek` nézettáblát is használhatatlanná teszi, mivel az egy már nem létező `Filmek` relációra hivatkozik.

8.2.2. Módosítható nézettáblák

Az SQL-szabvány formálisan leírja, mikor lehet egy nézettáblát módosítani és mikor nem. Az SQL-szabályok bonyolultak, de nagyjából azt engedik meg, hogy egy R reláción (amely szintén lehet módosítható nézettábla) definiált nézettáblát módosíthassunk, ha definíciójában a `SELECT` után `DISTINCT` nem szerepel. Két fontos kikötés van:

- A `WHERE` záradékban R nem szerepelhet egy alkérdésben sem.
- A `FROM` záradékban csak R szerepelhet, csak egyszer fordulhat itt elő és semmilyen más reláció nem szerepelhet (a `FROM` záradékban).
- A `SELECT` záradék listája elég attribútumot kell tartalmazzon ahhoz, hogy egy beszúrás esetén a többi attribútumot nullértékkel, vagy az alapértelmezett értékkel tölthessük fel az alaptáblában. Például kötelező az olyan attribútumérték megadása, mely `NOT NULL`-nak van deklarálva és nincs alapértelmezett értéke.

A nézettáblába való beszúrás közvetlenül a nézettábla R alaptáblájába történik. Az egyetlen finom különbség az, hogy csak azon attribútumoknak tudunk így értéket adni, amelyek a nézettáblát definiáló `SELECT` záradékban előfordulnak.

8.5. példa. Tételezzük fel, hogy 8.1. példában definiált `ParamountFilmek` nézettáblába szeretnénk egy sort beszúrni a következő módon:

```
INSERT INTO ParamountFilmek
VALUES('Csillagok háborúja', 1979);
```

A `ParamountFilmek` nézettábla megfelel az SQL módosíthatósági feltételeinek, mivel a következő alaptábla néhány attribútumára vonatkozik:

```
Filmek(filmcím, év, műfaj, hossz, stúdióNév, producerAzon)
```

A `ParamountFilmek` nézettáblába való beszúrás a `Filmek` táblába való következő beszúrásként kerül végrehajtásra:

```
INSERT INTO Filmek(filmcím, év)
VALUES('Csillagok háborúja', 1979);
```

Megjegyezzük, hogy a nézettábla definíciója miatt ebben a beszúrásban csak a `filmcím` és `év` attribútumok kaphattak értéket, a `Filmek` tábla többi attribútumainak nem tudtunk értéket adni.

A `Filmek` táblába beszúrandó sor attribútumértékei tehát: a `filmcím` attribútumértéke `'Csillagok háborúja'`, az `év` attribútumértéke `1979`, az összes többi attribútumértéke `NULL`. Minthogy a beszúrt sor `stúdióNév` attribútumának értéke `NULL`, és ez a `ParamountFilmek` nézettábla válogatási feltételének nem felel meg, így a beszúrt sornak a nézettáblára nincs hatása. Így például

a 8.3. példa lekérdezése a most beszúrt sort (`'Csillagok háborúja', 1979`) nem fogja figyelembe venni.

Ezen anomália elkerülése végett a nézettábla definiálásakor a `SELECT` záradékban `stúdióNév` attribútumot is szerepeltetnünk kell:

```
CREATE VIEW ParamountFilmek AS
   SELECT stúdióNév, filmcím, év
   FROM Filmek
   WHERE stúdióNév = 'Paramount';
```

Ezután a `ParamountFilmek` nézettáblába beszúrjuk a kívánt sort:

```
INSERT INTO ParamountFilmek
VALUES('Paramount', 'Csillagok háborúja', 1979);
```

Ez a beszúrás a `Filmek` táblára ugyanúgy hat, mint a következő:

```
INSERT INTO Filmek(stúdióNév, filmcím, év)
VALUES('Paramount', 'Csillagok háborúja', 1979);
```

Megjegyezzük, hogy bár a létrehozott sor a nem említett attribútumokon `NULL` értéket kap, de a `ParamountFilmek` nézettáblában éppen a beszúrás által megkívánt sort fogja eredményezni. □

A módosítható nézettáblákból törölhetünk is. A törlés, a beszúráshoz hasonlóan, az alaptáblában történik. Azért, hogy biztosítsuk azt, hogy csak azok a sorok törlődjenek, amelyek a nézettáblában láthatóak, a törlő utasítás `WHERE` záradékához „és" (`AND`) logikai művelettel hozzá kell kapcsolnunk a nézettáblát definiáló utasítás `WHERE` záradékában szereplő feltételt.

8.6. példa. Tételezzük fel, hogy a módosítható `ParamountFilmek` nézettáblából szeretnénk kitörölni az összes olyan filmet, amelyek címe tartalmazza a „háború" szót. Ezt a következő utasítás segítségével érhetjük el:

```
DELETE FROM ParamountFilmek
WHERE filmcím LIKE '%háború%';
```

A törlés átalakul egy, a `Filmek` táblára vonatkozó törlésre azzal a különbséggel, hogy a `ParamountFilmek` nézettáblát definiáló lekérdezés `WHERE` feltétele hozzáadódik a törlés `WHERE` feltételéhez:

```
DELETE FROM Filmek
WHERE filmcím LIKE '%háború%' AND stúdióNév = 'Paramount';
```

a művelet eredménye. □

Hasonlóképpen, a módosítások is az alaptáblán keresztül történnek. A nézettábla módosításainak tehát az a hatása, hogy az alaptábla azon sorait módosítja, amelyek a nézettáblában sort eredményeznek.

Miért nem módosíthatók egyes nézettáblák?

Tekintsük a 8.2. példában előforduló `FilmProducer` nézettáblát, amely a filmcímeket párosítja a gyártásirányítókkal. Ez a nézettábla az SQL definíciója szerint nem módosítható, mivel két relációra vonatkozik: a `Filmek` és a `GyártásIrányító` relációkra. Tételezzük fel, hogy szeretnénk beszúrni a következő sort:

```
('Indiana Jones és a haláltemplom', 'George Lucas')
```

A `Filmek` és a `GyártásIrányító` relációkba is be kellene szúrni egy-egy sort. Az olyan attribútumokra, mint például a `hossz` vagy a `filmcím`, használhatjuk az alapértelmezett értéket, de mit csináljunk a két egyenlővé tett attribútummal, a `producerAzon` és az `azonosító` attribútumokkal? Használhatnánk a `NULL` értéket számukra. Így azonban nem tudnánk összekapcsolni a két relációt, mert az SQL két `NULL` értéket nem tekint egyenlőnek (lásd 6.1.6. alfejezet). Tehát az `'Indiana Jones és a haláltemplom'` és a `'George Lucas'` nem kapcsolódna össze a `FilmProducer` nézettáblában, tehát a beszúrás nem lenne helyesen végrehajtva.

8.7. példa. A következő, nézettáblában történő módosítás:

```
UPDATE ParamountFilmek
SET év = 1979
WHERE filmcím = 'Csillagok háborúja film';
```

az alaptáblában ilyen alakban megy végre:

```
UPDATE Filmek
SET év = 1979
WHERE filmcím = 'Csillagok háborúja film' AND
    stúdióNév = 'Paramount';
```

□

8.2.3. Nézettáblákra vonatkozó „helyette" (instead-of) típusú triggerek

Ha nézettáblára vonatkozó triggert definiálunk, akkor a `BEFORE` és `AFTER` helyett az `INSTEAD OF`-ot kell használnunk. Ha így járunk el, akkor ha egy esemény kiváltja a trigger működését, akkor a triggerben megadott műveletek futnak le a trigger működését kiváltó művelet helyett. Így az instead-of típusú trigger elfogadja a nézettábla módosítási kísérletet, és helyette az adatbázis tervezője által helyesnek gondolt műveletet hajtja végre. A következő egy tipikus példa erre.

8.8. példa. Idézzük fel a Paramount által készített filmek nézettáblájának 8.1. példában szereplő definícióját:

```
CREATE VIEW ParamountFilmek AS
   SELECT filmcím, év
   FROM Filmek
   WHERE stúdióNév = 'Paramount';
```

Amint a 8.5. példában megmutattuk, ez a nézettábla módosítható, de nem kívánt következménnyel jár, ha ugyanis a `ParamountFilmek` nézettáblába beszúrunk egy sort, akkor a rendszer nem tudja kikövetkeztetni, hogy az alaptáblába történő beszúráskor a `stúdióNév` attribútum helyes értéke `Paramount` lesz-e, így a `stúdióNév` `NULL` értéket kap.

Jobb megoldást érhetünk el, ha egy erre a táblára vonatkozó, a 8.2. ábrán látható, „instead-of" típusú triggert hozunk létre. A triggerben nincs semmi meglepő. A 2. sorban látjuk az `INSTEAD OF` kulcsszót, amellyel azt érjük el, hogy a `ParamountFilmek` nézettáblába való beszúrás nem jön létre, helyette ez a trigger fog működni.

```
1)  CREATE TRIGGER ParamountInsert
2)  INSTEAD OF INSERT ON ParamountFilmek
3)  REFERENCING NEW ROW AS ÚjSor
4)  FOR EACH ROW
5)  INSERT INTO Filmek(filmcím, év, stúdióNév)
6)  VALUES(ÚjSor.filmcím, ÚjSor.év, 'Paramount');
```

8.2. ábra. A nézettáblába való beszúrást az alaptáblán történő beszúrásra cserélő trigger

A végrehajtani kívánt beszúrás helyett az 5. és 6. sorokban megadott akció következik be. Ez a `Filmek` táblába történő beszúrás, melyben a három ismert attribútumot adjuk meg. A `filmcím` és az `év` attribútumok a nézettáblába beszúrni kívánt sorból származnak; ezen értékekre az `ÚjSor` sorváltozó névvel hivatkoztunk, melyet a 3. sorban deklaráltunk azért, hogy meg tudjuk nevezni azt a sort, amelyet a nézettáblába kíséreltünk meg beszúrni. A `stúdióNév` attribútum értéke a `'Paramount'` konstans. Ez nem a nézettáblába beszúrni kísérelt sor része, hanem abból a feltételezésünkből következik, hogy a beszúrás a `ParamountFilmek` nézeten keresztül érkezett. □

8.2.4. Feladatok

8.2.1. feladat. Melyek módosíthatók a 8.1.1. feladat nézettáblái közül?

8.2.2. feladat. Tegyük fel, hogy megkonstruáltuk a következő nézettáblát:

```
CREATE VIEW DisneyVígjátékok AS
    SELECT filmcím, év, hossz FROM Filmek
    WHERE stúdióNév = 'Disney' AND műfaj = 'vígjáték';
```

a) Módosítható-e ez a nézettábla?

b) Készítse el az ezen nézettáblába való beszúrást kezelő instead-of típusú triggert.

c) Készítsen instead-of típusú triggert, amely ezen nézettáblán keresztül a (címmel és évvel megadott) film hosszának módosítását oldja meg.

8.2.3. feladat. A következő alaptáblákra vonatkozóan:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, ár)
```

megkonstruáltuk a következő nézettáblát:

```
CREATE VIEW ÚjPC  AS
SELECT gyártó, modell, sebesség, memória, merevlemez, ár
FROM Termék, PC
WHERE Termék.modell = PC.modell AND típus = 'pc';
```

Megjegyezzük, hogy konzisztencia-ellenőrzést is használunk: a modellszám nemcsak a PC-relációban fordul elő, hanem a `Termék` reláció `típus` attribútuma jelzi, hogy a szóban forgó termék PC.

a) Módosítható-e ez a nézettábla?

b) Készítse el az ezen nézettáblába való beszúrást kezelő instead-of típusú triggert.

c) Készítsünk instead-of típusú triggert, amely ezen nézettáblán keresztül az ár módosítását oldja meg.

d) Készítsen instead-of típusú triggert, amely ezen nézettáblán keresztül adott sor törlését oldja meg.

8.3. Indexek az SQL-ben

A reláció A attribútumára épített index egy olyan adatstruktúra, amely a reláció olyan sorainak megkeresését teszi hatékonyabbá, melyekben az A attribútum értéke valamilyen meghatározott érték. Az indexre úgy is gondolhatunk, mint (kulcs, érték) párokra vonatkozó bináris keresőfára, ahol az a kulcs (az A attribútum egy lehetséges értéke) van társítva az „értékkel", amely az olyan sorok helyének halmaza. mely sorokban az A attribútum értéke a. Az ilyen index az

olyan kérdések kiértékelésében segít, melyekben az A értékét konstanssal hasonlítjuk össze, például $A = 3$ vagy $A \leq 3$. Az index kulcsa egy attribútum vagy az attribútumok halmaza lehet, de nem szükséges, hogy annak a relációnak kulcsa legyen, amelyre az indexet építjük. Az index attribútumaira, ha a megkülönböztetés fontos, akkor „indexkulcsként" hivatkozunk.

A nagy relációkra épített indexek megvalósításának technológiája az adatbázisrendszerek megvalósításának központi fontosságú kérdése. A tipikus adatbázisrendszerekben használt legfontosabb adatstruktúra a „B-fa", amely a kiegyensúlyozott bináris fa általánosítása. Amikor az adatbázis megvalósításáról beszélünk, akkor B-fákat feltételezünk, de most elegendő, ha az indexekre, mint bináris keresőfákra gondolunk.

8.3.1. Az index használatának indoka

Ha a reláció nagyon nagy, akkor az összes sorainak átvizsgálása – hogy a kívánt tulajdonságú (esetleg nagyon kevés) sorokat megtaláljuk –, nagyon drága. Tegyük fel például, hogy az első megvizsgált lekérdezés a 6.1. példa lekérdezése:

```
SELECT *
FROM Filmek
WHERE stúdióNév = 'Disney' AND év = 1990;
```

A `Filmek` táblában lehet 10 000 sor, és ebből csak 200 vonatkozik 1990-ben készült filmre.

A lekérdezés naiv megvalósítása beolvassa mind a 10 000 sort és mindegyikre ellenőrzi a `WHERE` záradékban megadott feltétel teljesülését. Sokkal hatékonyabb lenne, ha valamilyen módon csak az 1990-ben gyártott filmek 200 sorát olvasnánk be és ezeken vizsgálnánk meg, hogy a Disney stúdió gyártotta-e őket. Még sokkal hatékonyabb lenne, ha eleve csak azt a 10 sort kaphatnánk meg, amelyek a `WHERE` záradék minden feltételének – azaz a stúdió Disney és az év 1990 – megfelelnének. Lásd a 8.3.2. alfejezet többattribútumos indexekre vonatkozó meggondolásait.

Az indexek a táblák összekapcsolását (join) használó lekérdezéseknél is hasznosak. Ezt szemlélteti a következő példa.

8.9. példa. Idézzük fel a 6.12. példa lekérdezését:

```
SELECT név
FROM Filmek, GyártásIrányító
WHERE filmcím = 'Csillagok háborúja'
                  AND producerAzon = azonosító;
```

Ez a lekérdezés a Csillagok háborúja című film producerének a nevét adja meg. Ha a `Filmek` reláció `filmcím` attribútumára van index, akkor ezt az indexet használhatjuk a Csillagok háborúja sorának megszerzéséhez. Ebből a sorból kinyerhetjük a producer azonosítóját (`producerAzon`).

Tegyük fel most, hogy a GyártásIrányító tábla azonosító attribútumára is van index. Akkor a producerAzon-t használhatjuk ezzel az indexszel a GyártásIrányító tábla azon sorának megtalálására, amely a Csillagok háborúja film producerére vonatkozik. Ebből a sorból pedig kinyerhetjük a producer nevét. Megjegyezzük, hogy ezzel a két indexszel a két táblából csak a kérdés megválaszolásához szükséges egy-egy sort olvastuk be. Indexek nélkül mindkét tábla összes sorát vizsgálnunk kell. □

8.3.2. Az index megadása SQL-ben

Jóllehet az indexek létrehozása – az SQL-99 szabványt is beleértve – egyetlen SQL-szabványnak sem része, a legtöbb kereskedelmi rendszer megadja a lehetőséget arra, hogy adott táblák adott attribútumára vonatkozó indexet definiáljunk. Tegyük fel, hogy a Filmek reláció év attribútumára kívánunk indexet építeni, erre a tipikus utasítás:

```
CREATE INDEX ÉvIndex ON Filmek(év);
```

Az eredmény a Filmek tábla év attribútumára épített index lesz, amelynek a neve ÉvIndex. Innentől kezdve az olyan lekérdezések kiértékelésében, amelyben az év attribútum is előfordul, az SQL-lekérdezés feldolgozója a Filmek táblának már csak a megadott évre vonatkozó sorait kell hogy vizsgálja; ennek eredményeként a kérdés megválaszolásához szükséges idő csökken.

Az adatbázisrendszerek gyakran lehetővé teszik, hogy többattribútumos egyszerű indexet építsünk. Az ilyen index több attribútum értékét tartalmazza, és hatékonyan találja meg azokat a sorokat, amelyekre ezen attribútumok értékei a megadottak.

8.10. példa. Minthogy a filmcím és az év képezik a Filmek tábla kulcsát, feltételezhetjük, hogy e két attribútum értékét vagy együtt használjuk, vagy egyiket sem. E két attribútumra vonatkozó index tipikus deklarációja a következő:

```
CREATE INDEX KulcsIndex ON Filmek(filmcím, év);
```

Minthogy a (filmcím, év) kulcs, ebből következik, hogy ha megadjuk a filmcímet és évet, akkor tudjuk, hogy az index csak egy sort fog találni, és az lesz a keresett sor. Ezzel szemben, ha csak az ÉvIndex áll rendelkezésünkre, és a kérdésben a címet és az évet is megadták, akkor a legjobb rendszer is kénytelen beolvasni az adott évhez tartozó összes filmet, és ezek mindegyikét vizsgálni az adott cím szerint.

Ha – amint az gyakran előfordul – a többattribútumos index valójában attribútumok adott sorrendben való összefűzése, akkor ezt az indexet arra is használhatjuk, hogy ezen attribútumok közül az első attribútumnak megfelelő sorokat keressük meg. Így a többattribútumos index tervezésének része az attribútumok megadási sorrendjének eldöntése is. Például, ha a film címét sokkal inkább meg tudjuk adni, mint a film gyártási évét, akkor a fent megadott sorrend az

alkalmasabb; ha inkább az évet tudjuk megadni, akkor az index létrehozásakor (év, filmcím) sorrendet használjunk. □

Ha az indexet törölni kívánjuk, akkor az alábbihoz hasonló utasításban elég csak az index nevét megadnunk:

```
DROP INDEX ÉvIndex;
```

8.3.3. Feladatok

8.3.1. feladat. A filmekkel kapcsolatos példa adatbázisunkban:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

Deklaráljunk indexet a következő attribútumokra vagy attribútumkombinációkra:

a) a stúdióNév.

b) a GyártásIrányító cím-e,

c) a film műfaj-a és hossz-a.

8.4. Az indexek kiválasztása

Az adatbázis tervezője az előnyök-hátrányok elemzésével tudja eldönteni, hogy mely indexeket hozza létre. A gyakorlatban ez a választás az egyik legfontosabb befolyásolója annak, hogy az adatbázis elfogadható teljesítményű lesz-e. Két fontos tényezőt kell figyelembe vennünk:

- Az attribútumra épített index jelentősen gyorsíthatja a válasz kidolgozását olyan kérdésekre, amelyben az adott attribútumra vonatkozó értéket vagy értéktartományt adnak meg. Az ilyen attribútumot használó összekapcsolásokat is gyorsítja az index léte.

- Más oldalról minden (egy vagy több attribútumra épített) index megnöveli az adott relációba való beszúrás, törlés és módosítás bonyolultságát és időszükségletét.

8.4.1. Egyszerű költségmodell

Annak megértéséhez, hogyan válasszunk indexeket az adatbázishoz, először tudnunk kell, hogy egy kérdés megválaszolásakor mivel múlik az idő. Annak részleteit, hogy a relációk hogyan tárolódnak, az adatbázisrendszer megvalósításakor gondoljuk át. Most tételezzük fel, hogy a reláció sorai a mágneslemezeken sok lapon vannak elosztva.[1] Egy lap, mely tipikusan legalább néhány ezer bájt méretű, a reláció sok sorát tartalmazhatja.

Minden egyes sor megvizsgálásához a sort tartalmazó teljes lapot be kell hozni a memóriába. Másrészről egy lapon tárolt összes sor megvizsgálása csak kicsit kerül többe egyetlen sor megvizsgálásánál. Azzal sok idő spórolható meg, ha a szükséges lap már a memóriában van, de az egyszerűség kedvéért feltételezhetjük, hogy ez sosem fordul elő, és így minden szükséges lapot a mágneslemezről kell elérnünk.

8.4.2. Hatékony indexek

Gyakran a leghasznosabb index a reláció kulcsára épített index. Ennek két oka van:

1. Gyakoriak az olyan lekérdezések, amelyekben a kulcs értékét adjuk meg. Így a kulcsra épített indexet gyakran fogjuk használni.

2. Minthogy legfeljebb egy, az adott kulcsnak megfelelő sor van, az index vagy semmit, vagy egy sor helyét határozza meg. Így legfeljebb egy lapot kell a memóriába behozni a sor elérése érdekében (további lapok elérésére is szükség lehet magának az indexnek a használatához).

A következő példa mutatja a kulcsindexek erejét olyan lekérdezésben is, mely összekapcsolást is tartalmaz.

8.11. példa. Vegyük elő a 6.3. ábrát, amelyen a `Filmek` és a `GyártásIrányító` táblák összekapcsoltjának (join) előállításához a sorok teljes párosítását mutattuk be. Az összekapcsolás ilyen megvalósításához legalább egyszer be kell olvasnunk az összes olyan lapot, amely a `Filmek` sorait tartalmazza, és az összes olyan lapot, amely a `GyártásIrányító` sorait tartalmazza. Minthogy ezen lapok száma túl nagy lehet ahhoz, hogy egyszerre elférjenek a memóriában, így meglehet, hogy a lapokat többször is be kell olvasnunk a mágneslemezről. Megfelelő indexek alkalmazásával a teljes lekérdezést meg tudjuk oldani eredménysoronként mindössze két lap beolvasásával.

A `Filmek` táblában a `filmcím` és `év` attribútumok alkotta kulcsra épített index segítségével gyorsan megkapjuk a *Csillagok háborúja* film sorát. Csak egy lapot – az ezen sort tartalmazó lapot – kell a mágneslemezről beolvasni. Ezután a producer azonosító számát ebből a sorból már megismerve, a

[1] E lapokra az adatbázisok tárgyalásakor gyakran „blokk"-ként hivatkozunk, de ha az olvasó tisztában van az operációs rendszerek lapozott memóriakezelésével, akkor a mágneslemezt lapokra osztottnak tekintheti.

GyártásIrányító tábla azonosító attribútumára épített index segítségével a GyártásIrányító táblában gyorsan megtaláljuk a producer (egyetlen) sorát. Ismét csak egyetlen – a GyártásIrányító tábla megfelelő sorát tartalmazó – lapot kell beolvasnunk a mágneslemezről, és esetleg az index használata végett néhány további lap beolvasása is szükséges lehet. □

Ha az index nem kulcs, akkor a kérdés megválaszolásához szükséges sorok mágneslemezről való beolvasásának ideje esetleg nő, esetleg nem nő. Két olyan helyzet van, melyben az index használata akkor is hatásos lehet, ha nem kulcsra építettük.

1. Ha az attribútum majdnem kulcs; azaz ha viszonylag kevés sor tartalmazza az adott attribútumában az adott értéket. Ha minden sor más-más lapra esik is, akkor sem kell a mágneslemezről sok lapot elérnünk.

2. Ha a sorok az adott attribútum szerint csoportosítottak. A relációt csoportosíthatjuk az attribútumon azonos értékeket tartalmazó sorok néhány lapra való csoportosításával. Így még ha sok sort kell is elérnünk, akkor is közel sem kell annyi lapot beolvasnunk mágneslemezről, mint a sorok száma.

8.12. példa. Az első típusú indexre példaként tegyük fel, hogy a Filmek tábla filmcím attribútumára építettünk indexet (és nem filmcím és év kulcs attribútumhalmazra). Mivel a filmcím önmagában nem kulcsa a relációnak, így az olyan címek, mint a *King Kong* a filmcím-re épített index több sorában is előfordulhatnak. Ha filmcím attribútumra épített index használatát a 8.11. példához hasonlóan elemezzük, akkor azt találjuk, hogy a *King Kong* című filmek keresése három sort fog eredményezni (minthogy három, 1933-ban, 1976-ban és 2005-ben gyártott ilyen című film van). Az előfordulhat, hogy a három sor három különböző lapon helyezkedik el, így mindhárom lapot be kell olvasni a memóriába, ez nagyjából háromszorosa egy ilyen lépés időszükségletének. Mivel azonban a Filmek reláció valószínűleg háromnál sokkal több lapon tárolódik, így ezen index használata is jelentős időmegtakarítást eredményez.

Következő lépésként e három sorból ki kell nyernünk a három producerAzon értéket, és a GyártásIrányító relációban meg kell keresnünk e három film producerét. A GyártásIrányító tábla három érintett sorának megkeresésére az azonosító-ra épített indexet használhatjuk. Meglehet, hogy ezek három különböző lapon találhatók, de ez is kevesebb ideig tart, mint a teljes GyártásIrányító reláció memóriába való bevitele. □

8.13. példa. Tegyük most fel, hogy a Filmek táblára csak egy indexet építettünk, az év attribútumra, és a következő kérdésre várjuk a választ:

```
SELECT *
FROM Filmek
WHERE év = 1990;
```

Tegyük fel először, hogy a `Filmek` sorai nincsenek az évek szerint csoportosítva; hanem mondjuk a címük szerint ábécérendben tároltak. Akkor ez a lekérdezés keveset nyer az `év` index létezésével. Ha laponként mondjuk 100 film van, akkor jó eséllyel minden lapon van legalább egy 1990-ben gyártott film. Így a `Filmek` relációt tartalmazó lapok nagy részét be kell olvasni a memóriába.

Tegyük most fel, hogy a `Filmek` reláció sorai az évek szerint csoportosítottak. Ekkor az év attribútumra épített index használatával megtaláljuk azt a néhány lapot, melyek az `év = 1990` feltételt kielégítő sorokat tartalmazzák. Ebben az esetben az `év` index nagy segítségünkre volt. Összehasonlításképpen a `filmcím` és `év` attribútumokra épített index kevésbé segítene, mindegy hogy mely attribútumra vagy attribútumokra nézve csoportosítanánk a `Filmek` reláció sorait. □

8.4.3. A legjobb indexelés meghatározása

Úgy tűnhet, hogy ha több indexet hozunk létre, akkor egyre valószínűbb, hogy az éppen adott lekérdezéshez jól használható index rendelkezésünkre fog állni. Ha azonban a módosítás gyakori, akkor az indexek létrehozásával sokkal megfontoltabban kell bánnunk. Az R reláció minden módosítása arra kényszeríti az adatbázisrendszert, hogy az R attribútumaira épített indexeket is módosítsa. Így nemcsak az R módosítandó lapjait kell beolvasnunk és kiírnunk, hanem az indexeket tartalmazó lapokat is. De ezzel együtt a gyakran használt attribútumokra való index felépítése növelheti a hatékonyságot akkor is, ha az adatbázis használata során a módosítás a jellemző akció. A módosító utasítás végrehajtása ugyanis az adatbázis lekérdezését is magában foglalja (például a select-from-where alkérdést tartalmazó `INSERT`, vagy a feltételt is tartalmazó `DELETE`). Nagyon óvatosnak kell lennünk a módosítások és lekérdezések relatív gyakoriságának megbecsülésekor.

Emlékeztetünk arra, hogy a tipikus reláció sok mágneslemezes blokkban (lapon) tárolódik, és a lekérdezések vagy módosítások elsődleges költsége gyakran a memóriába beolvasandó lapok száma (azzal arányos). Így az indexek, melyekkel a teljes reláció átvizsgálása nélkül meg tudjuk találni a sorokat, sok időt takaríthatnak meg. Azonban maguk az indexek is – legalább részben – mágneslemezen tárolandók, így az indexek elérése és módosítása is költséget okoz. Valóban, a módosítás a lap beolvasásakor és a módosított lap kiírásakor is mágneslemez-elérést igényel, nagyjából kétszer olyan költséges, mint a lekérdezésben az index vagy az adat elérése.

Egy új index értékeléséhez meg kell becsülnünk, hogy az adatbázisban többnyire milyen kérdéseket tesznek fel és milyen módosításokat hajtanak végre. Olykor a kérdésekről van feljegyzés (naplóhoz hasonló) és ebből jó információt nyerhetünk, ha azzal a feltevéssel élünk, hogy az adatbázis jövőbeni használata a múltbelihez hasonló lesz. Máskor tudhatjuk, hogy az adatbázis valamilyen konkrét alkalmazást vagy alkalmazásokat szolgál ki, ekkor ezen alkalmazások forráskódjából láthatjuk, hogy ezekben milyen SQL-kérdéseket és módosításokat használnak. Mindkét helyzetben fel tudjuk sorolni, milyen típusúak a leg-

gyakoribb kérdések és módosítások. E típusutasításokban konstansok helyén változók is szerepelhetnek, de különben teljesen olyanok, mint a tényleges SQL-utasítások. Íme egy egyszerű példa az elemzési folyamatra és az elvégzendő számítások bemutatására.

8.14. példa. Legyen a relációnk:

```
SzerepelBenne(filmcím, filmÉv, színészNév)
```

Tegyük fel, hogy ezen a reláción leggyakrabban az alábbi három adatbázis-műveletet végezzük:

Q_1: Egy adott színész filmjeinek címét és gyártási évét keressük, tehát a következő lekérdezést hajtjuk végre:

```
SELECT filmCím, filmÉv
FROM SzerepelBenne
WHERE színészNév = s;
```

ahol s konstans.

Q_2: Egy adott film szereplőit keressük, tehát a következő lekérdezést hajtjuk végre:

```
SELECT színészNév
FROM SzerepelBenne
WHERE filmCím = t AND filmÉv = y;
```

ahol t és y konstansok.

I: Új sort adunk a relációhoz, azaz végrehajtjuk a következő `INSERT` utasítást:

```
INSERT INTO SzerepelBenne VALUES(t, y, s);
```

ahol t, y és s konstansok.

Az adatokra vonatkozóan tegyük fel:

1. A `SzerepelBenne` relációt 10 mágneslemezblokkban tároljuk, tehát a teljes reláció átvizsgálásának költsége 10.
2. A színészek átlag 3 filmben szerepelnek, és a filmeknek átlagosan 3 szereplőjük van.
3. Minthogy adott színész és adott film sorai a `SzerepelBenne` 10 darab mágneslemezblokkjában szétosztottak, így ha van indexünk a `név` vagy a `(filmCím, filmÉv)` attribútumokra, akkor egy adott színész vagy film (átlagosan) 3 sorának elérése 3 lemezolvasást igényel. Ha az adott attribútumra nincs indexünk, akkor 10 lemezolvasás szükséges.

4. Mindig, ha indexet használunk egy sor megkereséséhez, akkor egy olvasással az indexet tartalmazó blokkot is be kell olvasnunk. Ha az indexblokkot módosítani kell (például beszúráskor), akkor a módosított indexblokk kiírása egy újabb lemezművelettel történik.

5. Hasonlóan, beszúráskor egy lemezművelet szükséges annak a blokknak a beolvasásához, amelybe a beszúrandó sor kerül, és egy másik lemezművelet szükséges a blokk visszaírásához. Feltételezzük, hogy ha nem használunk indexet, akkor is a teljes reláció átvizsgálása nélkül meg tudjuk állapítani, hogy melyik blokkba kerül a beszúrandó sor.

Művelet	*Index nélkül*	*Név index*	*Cím index*	*Mindkét index*
Q_1	10	4	10	4
Q_2	10	10	4	4
I	2	4	4	6
Átlagosan	$2 + 8p_1 + 8p_2$	$4 + 6p_2$	$4 + 6p_1$	$6 - 2p_1 - 2p_2$

8.3. ábra. A három művelet költsége a használt indexek függvényében

A 8.3. ábra megadja a három műveletet, Q_1 (adott színész lekérdezése), Q_2 (adott film lekérdezése) és az I (beszúrás) költségeit. Ha nincs index, akkor a Q_1 és Q_2-höz a teljes relációt végig kell vizsgálnunk (ennek költsége 10)[2], a beszúrás kevesebb műveletet igényel, beolvassuk a szabad helyet tartalmazó blokkot és az új sorral kiegészítve visszaírjuk (költsége 2, mert feltételeztük, hogy ezt a blokkot index nélkül is „egyből" megtaláljuk). Ezek az eredmények láthatók az *Index nélkül* oszlopban.

Ha csak a **név**-re van indexünk, akkor Q_2 még mindig a teljes reláció végigvizsgálását kívánja (költsége 10). A Q_1 lekérdezés megválaszolásához egy olvasással az indexblokkot érjük el, és abból megtudjuk, hogy a kívánt sorok melyik három további blokk beolvasásával érhetők el. Az I beszúrás végrehajtásához mind az indexblokkot, mind az adatblokkot be kell olvasni és vissza is kell írni, tehát ez összesen 4 lemezművelet igényű.

Amennyiben csak a címre építettünk indexet, akkor a név indexeléséhez hasonló a műveletigény (a költség). Végül, ha mindkét indexet használjuk, akkor mind Q_1, mind Q_2 4 lemezműveletet igényel. Ugyanakkor az I beszúrás végrehajtásához mind a kettő indexblokkot és az adatblokkot be kell olvasni és vissza is kell írni, ez tehát összesen 6 lemezművelet igényű. Ezt mutatja a 8.3. ábra utolsó oszlopa.

A 8.3. ábra utolsó sorában egy művelet átlagos költségét adjuk meg azzal a feltevéssel, hogy Q_1-et az adatbázis működési idejének p_1 hányadában, Q_2-t p_2 hányadában és így I-t a működési idő $1 - p_1 - p_2$ hányadában használjuk.

[2] Itt egy apróságot figyelmen kívül hagytunk. Sokszor lehetőségünk van arra, hogy a relációt a mágneslemezen egymás utáni lapokon vagy sávokon tároljuk. Ilyen esetben a teljes reláció beolvasásának költsége lényegesen kisebb lehet, mint ugyanannyi lap véletlenszerű (szétszórt) elérésének költsége.

p_1 és p_2 függvényében az indexelés mind a négy esetére kiszámítható a három művelet legjobb átlagos költsége. Például, ha $p_1 = p_2 = 0,1$, akkor a $2 + 8p_1 + 8p_2$ kifejezés értéke a legkisebb, tehát azt javasolhatjuk, hogy ne használjunk semmilyen indexelést. Ha tehát sok beszúrást és csak néhány lekérdezést végzünk, akkor nincs szükségünk indexelésre. Másrészről ha $p_1 = p_2 = 0,4$, akkor a $6 - 2p_1 - 2p_2$ kifejezés értéke válik a legkisebbé, tehát mindkét index használatát javasoljuk. Ösztönösen is azt érezzük, hogy ha sok lekérdezést hajtunk végre, és azokban nagyjából egyformán gyakran hivatkozunk a filmekre és filmszínészekre, akkor mindkét index kívánatos.

Ha $p_1 = 0,5$ és $p_2 = 0,1$, akkor – lévén hogy a $4 + 6p_2$ formula adja a legkisebb értéket – az az eset adja a legjobb eredményt, ha csak a név indexet használjuk. Hasonlóan, ha $p_1 = 0,1$ és $p_2 = 0,5$, akkor a legjobb, ha csak a cím indexet használjuk. Intuitíve is úgy sejtjük, hogy ha csak egyetlen kérdéstípus használata gyakori, akkor csak olyan index építése éri meg, amely az adott típusú kérdés megválaszolásában segít. □

8.4.4. A létrehozandó indexek automatikus meghatározása

Adatbázisok „tuningolása" az a tevékenység, amely során nemcsak az indexeket határozzuk meg, hanem sok más paramétert is beállítunk. Még nem sokat foglalkoztunk az adatbázisok fizikai megvalósításával, de néhány példa hasznos a tuningolásra, hogy mennyi memóriát rendelünk a különböző folyamatokhoz, és hogyan határozzuk meg a mentések és ellenőrzőpont-képzések gyakoriságát (a katasztrófa utáni helyreállítás elősegítésére). Számos segédeszközt terveztek azért, hogy az adatbázis tervezőjéről levegyék a felelősséget és a rendszer önmagát tuningolja, vagy legalábbis adjon tanácsot a tervezőnek a jó döntésekhez.

A fejezet végén az irodalomjegyzékben megemlítünk néhány ilyen projektet. Mindazonáltal itt is adunk egy áttekintést a tuningolási tanácsadók indexmeghatározási tevékenységéről.

1. Az első lépés a lekérdezésekről kimutatást készíteni. Mivel az adatbázisrendszerek rendesen minden adatbázis-műveletet naplóznak, így lehetőségünk van arra, hogy átvizsgáljuk a naplót és megállapítsuk a kezelt adatbázisra vonatkozó tipikus lekérdezéseket és adatbázis-módosításokat. Vagy lehetséges az is, hogy az adatbázist használó alkalmazói programokból állapítsuk meg, melyek a tipikus műveletek.

2. A tervező előírhat megszorításokat, például, hogy bizonyos indexet kell vagy nem kell választani.

3. A tuningolási tanácsadó rendszer lehetséges *indexjelöltek* halmazát állítja elő, és mindegyiket értékeli. Az adatbázisrendszer lekérdezésoptimalizálójának tipikus lekérdezéseket ad. A lekérdezésoptimalizáló képes arra, hogy egy-egy indexhalmaz használatát feltételezve megbecsülje a lekérdezés végrehajtásának időszükségletét.

4. A legkisebb költséget eredményező indexhalmazt javasolja a tervezőnek, vagy automatikusan generálja azt.

A 3. lépésben, amikor a lehetséges indexeket tekintjük át, apró probléma jelentkezik. Korábban választott index léte hatással lehet arra, hogy az éppen vizsgált index mennyi *hasznot* hozhat (mennyivel javítja az utasításhalmaz átlagos végrehajtási idejét). A indexek kiválasztásában bizonyítottan hatásos a „mohó" megközelítés, azaz:

a) Kezdetben, amikor még nem választottunk indexet, megbecsüljük minden indexjelölt hasznosságát. Ha legalább egy pozitív hasznosságú (azaz csökkenti a kérdések megválaszolásának átlagos idejét), akkor azt az indexet választjuk.

b) Ezután – figyelembe véve a már korábban kiválasztott indexek használatát is – újra értékeljük a többi indexjelölt hasznosságát. Ismét kiválasztjuk a legnagyobb hasznot produkáló indexet, feltéve, hogy a haszon pozitív.

c) Ismételjük az indexjelöltek értékelését a már korábban kiválasztott indexek létezésének figyelembevételével. Mindig a legnagyobb hasznot biztosító indexet választjuk mindaddig, amíg a haszna pozitív.

8.4.5. Feladatok

8.4.1. feladat. Tegyük fel, hogy a 8.14. példában szereplő `SzerepelBenne` reláció tárolása 10 lap helyett 100 lapot igényel, a példa többi feltevéseit változatlanul megtartjuk. Adjuk meg a példában tárgyalt négy (az index használatával és nélküle kiszámított) esetre a Q_1, Q_2 lekérdezések és az I beszúrás költségének formuláit a p_1 és p_2 függvényében.

! **8.4.2. feladat.** Ebben a feladatban a hajós példánknak a

```
Hajók(név, osztály, felavatva)
```

relációindexeit vizsgáljuk. Tegyük fel, hogy:

i) A `név` a kulcs.

ii) A `Hajók` relációt 50 lapon tároljuk.

iii) A reláció az `osztály` attribútum szerint csoportosított és így feltehetjük, hogy egy adott osztályhoz tartozó hajók megtalálásához elegendő egyetlen lemezművelet.

iv) Az osztályokhoz átlagosan 5 hajó tartozik, és évente átlag 25 hajót avattak.

v) p_1 a valószínűsége a `SELECT * FROM Hajók WHERE név = n` típusú lekérdezés használatának.

vi) p_2 a valószínűsége a `SELECT * FROM Hajók WHERE osztály = o` típusú lekérdezés használatának.

vii) p_3 a valószínűsége a `SELECT * FROM Hajók WHERE felavatva = é` típusú lekérdezés használatának.

viii) $1 - p_1 - p_2 - p_3$ a valószínűsége annak, hogy a `Hajók` relációba új sort szúrunk be.

Az indexek elérésére és a beszúráshoz szükséges szabad hely megtalálására vonatkozóan legyenek azok a feltételezéseink, amiket a 8.14. példában is alkalmaztunk.

Tegyük fel, hogy a `név`, `osztály` és `felavatva` attribútumokra építünk indexeket. Az indexek minden kombinációjára becsüljük meg a műveletek költségét. A p_1, p_2 és p_3 függvényében hogyan válasszuk meg legjobban az indexeket?

8.5. Tárolt nézettáblák

A nézettáblát létrehozó utasítás leírja, hogy az adatbázis alaptábláiból, ezen táblákon végrehajtott lekérdezéssel, hogyan hozhatunk létre új táblákat. Eddig úgy tekintettük, hogy a nézettáblák csak a relációk logikai leírásai. Ugyanakkor, ha egy nézettáblát elég gyakran használnak, akkor hatékonyabb lehet, ha a nézettáblát ténylegesen *tároljuk*; azaz a tartalmát állandóan megtartjuk. Ahogy az indexek fenntartása is, úgy a tárolt nézettáblák fenntartása is költséggel jár, hiszen az alaptáblák minden módosítását a tárolt nézettáblákban is követni kell.

8.5.1. A tárolt nézettáblák karbantartása

Az alapelv az, hogy az adatbázisrendszer a tárolt nézettáblákat mindig újra számítja, ha az alaptáblákban bármilyen változás történt. Egyszerű nézettáblákra vonatkozóan korlátozhatjuk, milyen gyakran vegyük figyelembe a tárolt nézettábla megváltozását, ezzel korlátozhatjuk a nézettábla karbantartására fordítandó munka mennyiségét. Vizsgáljunk meg egy példát, amelyben a nézettábla összekapcsolással (join) keletkezik, és láthatjuk, hogy több lehetőségünk is van a karbantartáshoz szükséges munka mennyiségének a csökkentésére.

8.15. példa. Tegyük fel, hogy gyakran kell megkeresnünk egy adott film producerének a nevét. Előnyösnek találhatjuk a következő tárolt nézettábla alkalmazását:

```
CREATE MATERIALIZED VIEW FilmProducer AS
    SELECT filmcím, év, név
    FROM Filmek, GyártásIrányító
    WHERE producerAzon = azonosító
```

Kezdjük azzal, hogy az adatbázisrendszer nem kell hogy figyelembe vegye a `Filmek` vagy a `GyártásIrányító` tábla bármely olyan attribútuma módosításának hatását a `FilmProducer` nézettáblára, mely attribútum a tárolt nézettáblát létrehozó definícióban nincs megemlítve. Biztos az is, hogy minden olyan módosítás, amely sem a `Filmek`, sem a `GyártásIrányító` táblát nem érinti, figyelmen kívül hagyható. Számos további egyszerűsítésre van lehetőségünk, amellyel a `Filmek` vagy a `GyártásIrányító` táblák módosításainak hatásait a tárolt nézettáblában is érvényesíthetjük anélkül, hogy a tárolt nézettáblát létrehozó lekérdezést újra végrehajtanánk.

1. Tegyük fel, hogy új filmet veszünk fel a `Filmek` táblába, legyenek az új film attribútumai: `filmcím = 'Kill Bill'`, `év` = 2003 és `producerAzon` = 23456. Ekkor a `GyártásIrányító` táblában csak a 23456 azonosítójú sort kell megkeresnünk. Mivel az azonosító a `GyártásIrányító` táblában kulcs, így a következő kérdés legfeljebb egy nevet ad válaszul.

   ```
   SELECT név FROM GyártásIrányító
   WHERE azonosító = 23456;
   ```

   Minthogy e kérdés válasza `név = 'Quentin Tarantino'`, így az adatbázisrendszer egy ennek megfelelő sort kell hogy beszúrjon a `FilmProducer` tárolt nézettáblába. A megfelelő `INSERT` utasítás:

   ```
   INSERT INTO FilmProducer
   VALUES('Kill Bill', 2003, 'Quentin Tarantino');
   ```

   Megjegyezzük, hogy mivel a `FilmProducer` egy tárolt nézettábla, bármely másik táblához hasonlóan tárolódik, és ez a művelet is ugyanúgy értelmezett, ezért ezt nem kell instead-of triggerrel vagy más módszerrel külön értelmezni.

2. Tegyük fel, hogy a `Filmek` táblából mondjuk az 1994-ben gyártott, `'Dumb & Dumber'` című filmet töröljük. Az adatbázisrendszer ekkor a `FilmProducer` táblából is csak ezt a filmet kell hogy törölje a következő utasítással:

   ```
   DELETE FROM FilmProducer
   WHERE filmcím = 'Dumb & Dumber' AND év = 1994;
   ```

3. Tegyük fel, hogy a `GyártásIrányító` táblába új sort szúrunk be. Az új sor tartalma `azonosító` = 34567 és `név = 'Max Bialystock'`. Ekkor az adatbázisrendszer a `FilmProducer` táblába azon filmek sorait kell hogy beszúrja, melyek korábban a producer ismeretlensége miatt nem voltak még ott. A megfelelő utasítás:

```
INSERT INTO FilmProducer
    SELECT filmcím, év, 'Max Bialystock'
    FROM Filmek
    WHERE producerAzon = 34567;
```

4. Tegyük fel, hogy a GyártásIrányító tábla 45678 azonosítójú sorát töröljük. Ekkor az adatbázisrendszernek a FilmProducer táblából is törölnie kell az olyan sorokat, amelyekben szereplő producer azonosítója 45678, hiszen az ilyen filmekhez már nem tartozik sor a GyártásIrányító táblában. Így az adatbázisrendszer végrehajtja a következő utasítást:

```
DELETE FROM FilmProducer
WHERE (filmcím, év) IN
    (SELECT filmcím, év FROM Filmek
     WHERE producerAzon = 45678);
```

Megjegyzés: Nem jó megoldás csak a GyártásIrányító tábla szerint a 45678 azonosítóhoz tartozó nevet felhasználni, és kitörölni a FilmProducer táblából az összes ilyen nevű producert tartalmazó sorokat. Ennek az az oka, hogy mivel a név nem kulcs a GyártásIrányító táblában, így előfordulhat két azonos nevű producer.

A Filmek tábla olyan módosításait, amelyek a filmcím és év attribútumokat is érintik, és a GyártásIrányító tábla azonosító attribútumát érintő módosítások vizsgálatát gyakorlatként az olvasóra hagyjuk. □

A 8.15. példából levonható nagyon fontos következtetés az, hogy a tárolt nézettábla módosításai *növekményesek* (inkrementálisak). Azaz soha nem elölről kezdve konstruáltuk újra a teljes nézettáblát, hanem az alaptáblákban történt beszúrást, törlést és módosítást a tárolt nézettáblában is követtük. Ezt az alaptáblákon való néhány lekérdezéssel és a tárolt nézettábla módosításával értük el. Ezen felül ezek a módosítások nem érintették a nézettábla összes sorát, csak azokat, amelyeknek legalább egy attribútumában az adott konstans fordul elő.

Nem lehet minden tárolt nézettáblához a 8.15. példában bemutatott szabályokhoz hasonlókat konstruálni, esetleg csak túl bonyolultakat. De a tárolt nézettáblák sok hasonló típusa lehetővé teszi a nézet növekményes karbantartását. A fejezethez tartozó feladatokban a tárolt nézettáblák egy másik típusát – az összesítő nézettáblákat – feladatokon keresztül vizsgáljuk meg.

8.5.2. A tárolt nézettáblák rendszeres karbantartása

Van a tárolt nézettábláknak olyan felhasználása, amely nem igényli azok állandó karbantartását, amikor az azt felépítő alaptáblákban változás történik, és így nem kell aggódnunk a naprakészség fenntartásának költségén vagy bonyolultságán. A 10.6. alfejezetben fogjuk tanulmányozni az OLAP-ot, de már itt

megjegyezzük, hogy az adatbázisok általában kétféle célt szolgálnak. Például egy áruházi részleg arra használja az adatbázisát, hogy a pillanatnyi raktárkészletet nyilvántartsa, amely minden eladással megváltozik. Ugyanezt az adatbázist az elemzők arra használhatják, hogy a vásárlási szokásokat értékeljék, és megjósolják, hogy egy adott árucikkből mikor kell ismét feltölteni a raktárt.

Előfordulhat, hogy az elemző kérdéseire a tárolt nézettáblákból sokkal hatékonyabban lehet válaszolni, különösen, ha a nézettábla összesített adatokat tartalmaz (például a fazon szerinti csoportosítás után a különböző méretű ingek leltári összesenjét). Mivel az adatbázist minden eladás módosítja, így a módosítás sokkal gyakoribb, mint a lekérdezés. Ha a módosítás a jellemző, akkor a tárolt nézettáblák és az indexek fenntartása is költséges.

Hasznos lehet azonban, ha tárolt nézettáblát hozunk létre, de azt nem tartjuk állandóan naprakészen, ha az azt felépítő alaptáblákban változás történik. Az ilyen tárolt nézettáblákat inkább rendszeresen újraépítjük, amikor az adatbázist nem használják intenzíven (tipikusan éjszakánként). Az ilyen tárolt nézettáblákat csak az elemzők használják, az adatai nem naprakészek, hanem 24 óránként frissülnek. Átlagos helyzetben egy árucikk eladási mértéke lassan változik, így az ilyen (naponta frissülő) adat „elég jó" az elemzéshez arra, hogy megjósolhassuk, hogy az adott árucikk jól, vagy gyengén fog fogyni. Természetesen, ha Brad Pitt hawaii ingben látható, akkor az elemző nem tudja megjósolni, hogy másnap kifogy-e a hawaii ing, de ilyen rizikó nem gyakran fordul elő.

8.5.3. Lekérdezések átírása a tárolt nézettáblák használatához

A tárolt nézettáblákra a lekérdezések `FROM` záradékában ugyanúgy hivatkozhatunk, ahogy a nézettáblákra is (lásd 8.1.2. alfejezetben). Mivel a tárolt nézettábla az adatbázisban ténylegesen tárolódik, így a lekérdezéseket átírhatjuk olyanná, amely a tárolt nézettáblát használja még akkor is, ha az eredeti lekérdezésben a nézettábla nem volt megemlítve. Az ilyen átírás a lekérdezés végrehajtását sokkal gyorsabbá teheti azáltal, hogy a válasz kidolgozási idejének jelentős részét kitevő műveleteket – például a relációk összekapcsolását – már a tárolt nézettábla létrehozásakor elvégeztük.

A lekérdezés tárolt nézettáblák használatára való átírásakor nagyon körültekintően kell eljárnunk. A tárolt nézettáblák használatára vonatkozó összes szabály ismertetése túlmutat e könyv keretein. Mindazonáltal a 8.15. példában bemutatott – és ahhoz hasonló – tárolt nézettáblákra vonatkozó viszonylag egyszerű szabályt tudunk javasolni.

Tegyük fel, hogy V tárolt nézettáblát a következő formájú lekérdezéssel definiáltuk:

```
SELECT L_V
FROM R_V
WHERE C_V
```

ahol L_V attribútumok listája, R_V relációk listája, C_V pedig egy feltétel. Hasonlóan tegyük fel, hogy a Q lekérdezés a következő alakú:

SELECT L_Q
FROM R_Q
WHERE C_Q

Azok a szabályok, amelyek figyelembevételével a Q kérdés a V tárolt nézettáblát használó lekérdezéssé alakítható, a következők:

1. Az R_V listában szereplő relációk mindegyike előfordul az R_Q listában is.
2. A C_Q feltétel ekvivalens a C_V AND C feltétellel, ahol C valamilyen feltétel. Abban a speciális esetben, amikor C_Q feltétel C_V-vel azonos, akkor az „AND C" szükségtelen.
3. Ha C feltétel szükséges, akkor az R_V listában szereplő relációk C feltételben is hivatkozott attribútumainak az L_V listában is elő kell fordulniuk.
4. Az L_Q listában szereplő attribútumoknak az L_V listában (melyben az R_V relációk attribútumai állhatnak) is elő kell fordulniuk.

Ha a fenti feltételek mindegyike teljesül, akkor a Q lekérdezést a V tárolt nézettáblára vonatkozó lekérdezéssé írhatjuk át, mégpedig a következő módon:

a) Az R_Q listát cseréljük ki a V relációból és az R_Q-ban szereplő, de R_V-ben elő nem forduló relációkból álló listára.

b) A C_Q feltételt cseréljük ki a C feltételre. Ha C nem szükséges (azaz $C_V = C_Q$), akkor a WHERE záradékot hagyjuk el.

8.16. példa. Tegyük fel, hogy rendelkezésünkre áll a 8.15. példában szereplő FilmProducer tárolt nézettábla. Ez a tárolt nézettábla a következő V lekérdezéssel volt definiálva:

```
SELECT filmcím, év, név
FROM Filmek, GyártásIrányító
WHERE producerAzon = azonosító
```

Tegyük fel továbbá, hogy arra a Q kérdésre kell válaszolnunk, hogy milyen nevű színészek szerepelnek a Max Bialystock producer által gyártott filmekben. A kérdés megválaszolásához a következő relációkra van szükségünk:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

A megfelelő Q lekérdezés:

```
SELECT színészNév
FROM SzerepelBenne, Filmek, GyártásIrányító
WHERE filmCím = filmcím AND Filmek.év = SzerepelBenne.filmÉv
   AND producerAzon = azonosító AND név = 'Max Bialystock';
```

Hasonlítsuk össze a tárolt nézettábla V definícióját és a Q lekérdezést, látjuk, hogy a fentebb megadott feltételeket kielégítik.

1. A V definíció `FROM` záradékában szereplő relációk mind előfordulnak a Q lekérdezés `FROM` záradékában is.

2. A Q lekérdezésben szereplő feltétel a (V-ből származó feltétel) `AND` C alakban is írható, ahol C =

```
filmCím = filmcím AND Filmek.év = SzerepelBenne.filmÉv
                              AND név = 'Max Bialystock'
```

3. A C feltétel attribútumai (`filmcím`, `év` és `név`) a V-beli relációk (`Filmek` és `GyártásIrányító`) attribútumai.

4. A Q kérdés `SELECT` listája egyetlen attribútuma sem származik a V definíció `FROM` listájában felsorolt relációkból.

Így a Q lekérdezésben a V tárolt nézettáblát is használhatjuk a lekérdezés következő átírásával:

```
SELECT színészNév
FROM SzerepelBenne, FilmProducer
WHERE filmCím = filmcím
          AND FilmProducer.év = SzerepelBenne.filmÉv
               AND név = 'Max Bialystock';
```

Azaz a `FROM` záradékban a `Filmek` és `GyártásIrányító` relációneveket kicseréltük a `FilmProducer` tárolt nézettábla nevére. A tárolt nézettábla definiálásakor szerepelt feltételt is eltávolítottuk a lekérdezésből, és így csak a C feltételt hagytuk meg. Az átalakított lekérdezés három helyett már csak két reláció összekapcsolásával dolgozik, így számíthatunk arra, hogy az átalakított lekérdezés végrehajtása gyorsabb az eredetinél. □

8.5.4. Tárolt nézettáblák automatikus előállítása

A 8.4.4. alfejezetben az indexek automatikus előállítására vonatkozó meggondolások a tárolt nézettáblákra is alkalmazhatók. Először is fel kell mérnünk, vagy meg kell becsülnünk, hogy az adott alkalmazásban milyen lekérdezések fordulnak elő. Az automatikus „tárolt-nézettábla-előállító tanácsadó" nézettáblajelölteket állít elő. Ez a feladat jóval bonyolultabb, mint az indexjelöltek előállítása.

Az indexek esetében minden reláció minden attribútumára egy index építése lehetséges. Indexeket építhetünk a reláció attribútumainak (kisebb) halmazaira is, de ha így járunk el, az indexjelöltek előállítása ekkor is egyszerű. Ugyanakkor tárolt nézettáblákat elméletileg bármilyen lekérdezéssel definiálhatunk, így a figyelembe veendő nézettáblák mennyisége határtalan.

A feladatot korlátozhatjuk azzal, ha arra gondolunk, hogy nincs értelme olyan tárolt nézettáblát létrehozni, amely egyetlen – a felmérésünkben előforduló – szóba jöhető lekérdezés végrehajtását sem segíti. Tegyük fel például, hogy a felmérésünkben szereplő összes lekérdezés a 8.5.3. alfejezetben bemutatott formájú. Ekkor a 8.5.3. alfejezetben adott elemzést felhasználhatjuk azon nézettáblák megkereséséhez, amelyek az adott lekérdezésekben hasznosak lehetnek. A tárolt nézettáblajelöltek keresését a következőkre korlátozhatjuk:

1. A `FROM` záradék relációlistájában csak olyan relációk forduljanak elő, amelyek a felmérésben szereplő lekérdezések valamelyikének `FROM` záradékában is szerepeltek.

2. A `WHERE` záradékban `AND` művelettel összekapcsolva csak olyan feltételek szerepeljenek, amelyek mindegyike a felmérés legalább egy lekérdezésében is előfordult.

3. A `SELECT` záradék attribútumlistája olyan legyen, amely elegendő a felmérésben szereplő lekérdezések legalább egyikéhez.

A tárolt nézettáblák hasznosságának megítéléséhez a lekérdezésoptimalizáló megbecsüli a lekérdezések végrehajtási idejét tárolt nézettábla használatával és nélküle. Ehhez természetesen az optimalizálónak képesnek kell lennie a tárolt nézettáblák vizsgálatára; az indexek vizsgálatára a modern optimalizálók mindegyike képes, de a tárolt nézettáblák vizsgálatára nincs mindegyik felkészítve. A 8.5.3. alfejezet olyan okfejtésre volt példa, amilyent az optimalizálónak is végre kell hajtani, hogy megállapíthassa, hogy az adott nézettábla hoz-e valami előnyt.

Van másik szempont is, amely felvetődik akkor, amikor a tárolt nézettáblák automatikus kiválasztását vizsgáljuk, és amely az indexekkel kapcsolatban nem jelent meg. A relációhoz épített index általában kisebb, mint maga a reláció; egy relációhoz építhető indexek helyigénye nagyjából ugyanakkora. A tárolt nézettáblák nagyon változóak a helyigény tekintetében: azok, amelyek összekapcsolást (join) igényelnek, sokkal nagyobbak lehetnek, mint az vagy azok a relációk, amelyekből felépülnek. Így át kell gondolnunk a tárolt nézettáblák „hasznossága" definícióját. Definiálhatjuk például a hasznosságot úgy is, hogy a felmérésben szereplő lekérdezések átlagos végrehajtási idejének csökkenési mértékét elosztjuk a tárolt nézettábla által okozott tárolási igény növekedésének mértékével.

8.5.5. Feladatok

8.5.1. feladat. Egészítsük ki a 8.15. példát az alaptáblák tartalma módosításának figyelembevételével.

! **8.5.2. feladat.** Tegyük fel, hogy a 8.2.3. feladat `ÚjPC` nézettáblája tárolt nézettábla. A `Termék` és a `PC` alaptáblák mely módosításai igénylik a tárolt nézettábla módosítását is? Hogyan tudjuk ezeket a módosításokat növekményesen megvalósítani?

! **8.5.3. feladat.** Ez a feladat az összesítésekre alapozott tárolt nézettáblákat vizsgálja. Tegyük fel, hogy a hajókkal kapcsolatos példaadatbázisunk a következő alaptábláit használva:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
```

tárolt nézettáblát építünk:

```
CREATE MATERIALIZED VIEW HajóStatisztika AS
    SELECT ország, AVG(vízkiszorítás), COUNT(*)
    FROM Hajóosztályok, Hajók
    WHERE Hajóosztályok.osztály = Hajók.osztály
    GROUP BY ország;
```

A `Hajóosztályok` és a `Hajók` alaptáblák mely módosításai igénylik a tárolt nézettábla módosítását is? Hogyan tudja ezeket a módosításokat növekményesen megvalósítani?

! **8.5.4. feladat.** A 8.5.3. alfejezetben megadtunk feltételeket, amelyek teljesülése esetén egyszerű tárolt nézettáblák hasonló lekérdezések végrehajtásában felhasználhatók. A 8.15. példa tárolt nézettáblájára nézve határozzuk meg az összes lekérdezéstípusokat, melyekhez ez a nézettábla használható.

8.6. Összefoglalás

- *Virtuális nézettáblák:* A virtuális nézettábla egy definíció; annak definíciója, hogy a nézettáblát logikailag hogyan lehet megkonstruálni az adatbázis tárolt tábláiból vagy más nézettáblákból. A nézettáblák ugyanúgy lekérdezhetők, mint a tárolt táblák, de az SQL átalakítja a lekérdezést úgy, hogy az a nézettábla definiálásánál használt tárolt táblákon fog lefutni.
- *Módosítható nézettáblák:* Az egy relációból felépített virtuális nézettáblák módosíthatók abban az értelemben, hogy a nézettáblába sorokat szúrhatunk be, törölhetünk és módosíthatunk ugyanúgy, mintha tárolt tábla lenne. Az ilyen műveleteket az adatbázisrendszer a nézettábla definíciójában szereplő alaptáblára vonatkozó ekvivalens műveletekké alakítja át.

- *Instead-of-Trigger („helyette működő trigger"):* Az SQL lehetővé teszi, hogy a nézettáblákra speciális triggert alkalmazzunk. Amikor nézettáblát módosító utasítás kerül végrehajtásra, akkor helyette az instead-of-trigger az alaptáblákon a triggerben megadott utasításokat hajtja végre.

- *Index:* Az SQL-szabvány ugyan nem tartalmazza, de a kereskedelmi rendszerek általában biztosítják az indexek definiálásának lehetőségét, és ezek az indexek meggyorsítják az egyes lekérdezéseket és módosításokat, amelyek keresési feltételében egy adott indexelt attribútumhoz tartozó, adott érték vagy értéktartomány szerepel.

- *Indexek megválasztása:* Amíg az indexek a lekérdezéseket felgyorsítják, ugyanakkor az adatbázis módosításait lelassítják, ugyanis a módosított relációhoz tartozó indexeket is módosítani kell. Így az indexek kiválasztása komplex probléma, az adatbázison végrehajtott tipikus utasítások függvénye.

- *Az index automatikus kiválasztása:* Egyes adatbázisrendszerek az indexek automatikus kiválasztására alkalmas eszközöket kínálnak. Ezek felmérik az adatbázison végrehajtott tipikus lekérdezéseket és módosításokat, és értékelik a különböző létrehozható indexek költséghatását.

- *Tárolt nézettáblák:* Ahelyett, hogy a nézettáblát az alaptáblákon végrehajtott lekérdezésként kezelnénk, a nézettáblát definiáló lekérdezést egy újabb tárolt tábla definiálására is használhatjuk, ezen táblában tárolt értékek az alaptáblákban tárolt értékek függvényei.

- *A tárolt nézettáblák karbantartása:* Amint az alaptábla megváltozik, akkor a rá hivatkozó, olyan tárolt nézettáblákat is megfelelően módosítani kell, amelyek értékei a módosításban érintettek. A tárolt nézettáblák sok általános típusánál a módosítások növekményesen is végrehajthatók anélkül, hogy a teljes nézettáblát újra elő kellene állítani.

- *A lekérdezések átírása tárolt nézettáblák lekérdezésévé:* Azok a feltételek, amelyek mellett a lekérdezések átírhatók úgy, hogy a tárolt nézettáblákat (is) felhasználják, bonyolultak. Ezzel együtt, ha a lekérdezésoptimalizáló képes ilyen átírásra, akkor az automatikus tervezési eszköz figyelembe veheti a tárolt nézettáblák létrehozásával elérhető hatékonyságnövekedést, és így automatikusan kiválaszthatja a létrehozandó tárolt nézettáblákat.

8.7. Irodalomjegyzék

A tárolt nézettáblák technológiáit [2] és [7] elemzik. A tárolt nézettáblák kiválasztásának mohó algoritmusát [3] írja le.

Az AutoAdmin (Microsoft) és a SMART (IBM) két, az adatbázisok automatikus tuningolásával foglalkozó projekt. Az AutoAdminnal kapcsolatos aktuális

információkat [8]-ban találunk. Az e rendszerben alkalmazott technológia leírását [1] tartalmazza.

A SMART projekt áttekintése [4]-ben olvasható. A projekt indexkiválasztási módszerének leírása [6]-ban található.

Az e fejezetben áttekintett indexkiválasztást, tárolt nézettáblákat, automatikus tuningolást [5] tárgyalja részletesen.

[1] S. Agrawal, S. Chaudhuri, V. R. Narasayya, „Automated selection of materialized views and indexes in SQL databases," *Intl. Conf. on Very Large Databases*, pp. 496–505, 2000.

[2] A. Gupta, I. S. Mumick, *Materialized Views: Techniques, Implementations, and Applications*, MIT Press, Cambridge MA, 1999.

[3] V. Harinarayan, A. Rajaraman, J. D. Ullman, „Implementing data cubes efficiently," *Proc. ACM SIGMOD Intl. Conf. on Management of Data* (1996), pp. 205–216.

[4] S. S. Lightstone, G. Lohman, S. Zilio, „Toward autonomic computing with DB2 universal database," *SIGMOD Record* **31**:3, pp. 55–61, 2002.

[5] S. S. Lightstone, T. Teorey, T. Nadeau, *Physical Database Design*, Morgan-Kaufmann, San Francisco, 2007.

[6] G. Lohman, G. Valentin, D. Zilio, M. Zuliani, A. Skelley, „DB2 Advisor: an optimizer smart enough to recommend its own indexes," *Proc. Sixteenth IEEE Conf. on Data Engineering*, pp. 101–110, 2000.

[7] D. Lomet, J. Widom (eds.), Special issue on materialized views and data warehouses, *IEEE Data Engineering Bulletin* **18**:2 (1995).

[8] A Microsoft AutoAdmin projekt elektronikus dokumentációja:
`http://research.microsoft.com/dmx/autoadmin/`

9. fejezet

Az SQL szerverkörnyezetben

Most kitérünk arra a kérdésre, hogyan illeszkedik be az SQL egy komplett programozási környezetbe. A tipikus szerverkörnyezeteket a 9.1. alfejezetben vezetjük be. A 9.2. alfejezetben bevezetjük a kliens-szerver-számításokhoz és adatbázishoz történő csatlakoztatásokhoz tartozó SQL-fogalmakat.

Ezek után áttekintjük, hogyan zajlik a valódi programozás, amikor is egy tipikus alkalmazás részeként SQL-t kell használnunk az adatbázishoz történő hozzáférésre. A 9.3. alfejezetben láthatjuk majd, hogyan ágyazható be az SQL egy hagyományos programozási nyelven – például C-ben – megírt programba. Kritikus kérdésként merül fel, hogyan cseréljünk adatokat az SQL-relációk és a környezeti vagy „befogadó" nyelv változói között. A 9.4. alfejezetben bevezetünk egy másik módszert az SQL általános célú programozással történő egyesítésére folyamatosan tárolt modulok segítségével, amelyek az adatbázisséma részeként tárolt kódrészletek, és amelyek a felhasználó által kiadott parancsokkal végrehajthatóak.

A harmadik programozási megközelítés egy „hívásszintű felület", amelynél egy hagyományos nyelvben programozunk, és függvénykönyvtárat használunk az adatbázishoz való hozzáférésre. A 9.5. alfejezetben a C programokból hívható SQL-standard könyvtárral foglalkozunk, amelyet SQL/CLI-nek neveznek. Majd a Java JDBC-jével (adatbázis kapcsolódásával) találkozhatunk a 9.6. alfejezetben, amely egy választható hívásszintű felület. Végezetül a 9.7. alfejezetben egy másik népszerű hívásszintű felülettel, a PHP-val fogunk foglalkozni.

9.1. Háromrétegű architektúrák

Az adatbázisok használatának többféle beállítása létezik, beleértve ebbe a kis méretű, egyedülálló rendszereket is. Egy kutató például a kísérleti adatainak tárolására használhatja egy MySQL vagy egy Microsoft Access adatbázis másolatát.

A nagyméretű adatbázisok telepítésének viszont egy nagyon általános architektúrája is létezik. Ez az architektúra inspirálta a jelen fejezetet. Ezt *háromrétegű* architektúrának nevezzük, mivel három különböző, egymással együttműködő szolgáltatást különböztet meg:

1. *Webszerverek.* Ezek a folyamatok csatlakoztatják általában az interneten vagy egy lehetséges lokális hálózaton keresztül a klienseket az adatbázisrendszerhez.

2. *Alkalmazásszerverek.* Ezek a folyamatok alkotják az ún. „üzleti logikát" a szerver céljainak megfelelően.

3. *Adatbázisszerverek.* Ezek a folyamatok működtetik az ABKR-t, segítségükkel hajtódnak végre az alkalmazásszerverről érkező lekérdezési, illetve módosítási kérések.

A folyamatok kis rendszerek esetén ugyanazon a processzoron is futhatnak, ám elterjedtebb megoldás az, hogy az egyes fokozatokhoz nagyobb mennyiségű processzort rendelnek. A 9.1. ábra egy javasolt elrendezést tartalmaz nagyméretű adatbázisrendszerekhez.

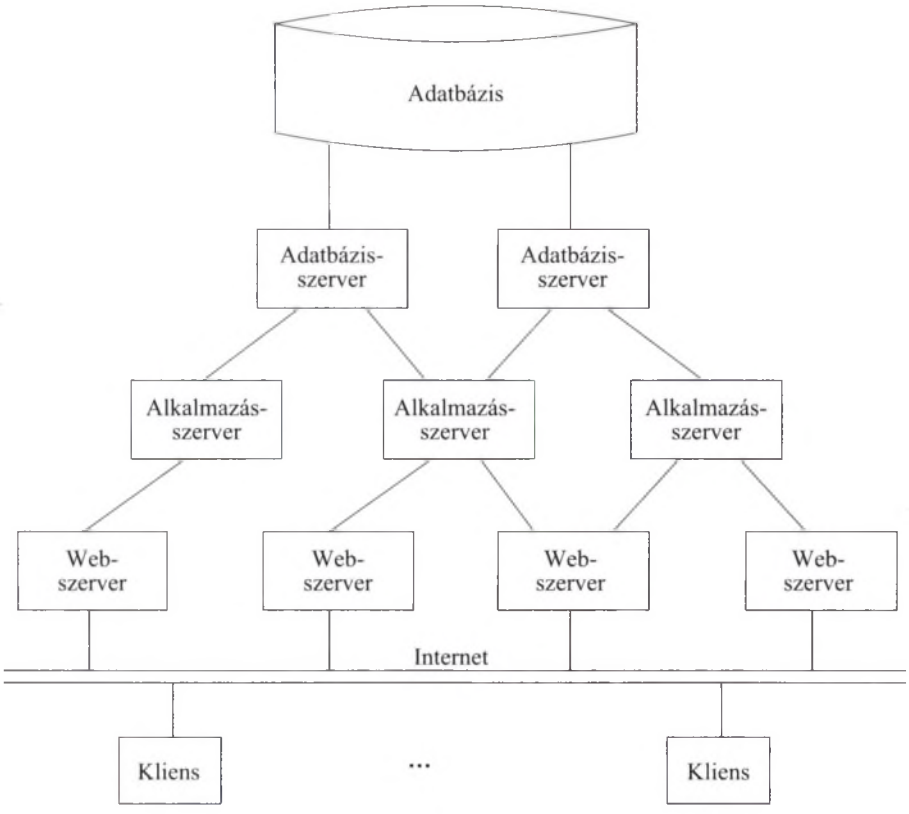
9.1.1. A webszerverréteg

A webszerver végzi a felhasználóval történő kommunikációt. Amikor egy felhasználó csatlakozik egy URL megnyitásán keresztül a rendszerhez, akkor egy webszerver, mint például az Apach/Tomcat, válaszol a kérésre. Ezáltal a felhasználó ennek a webszervernek egy *kliensévé* válik. A kliens által kezdeményezett műveleteket a böngésző segítségével lehet elindítani, amely például kezeli annak az űrlapnak a kitöltését is, amely később átadásra (POST) kerül a webszervernek.

Tegyük fel a példa kedvéért, hogy van egy *Amazon.com*-hoz hasonló weblapunk. Ekkor egy felhasználó (ügyfél) a saját böngészőjébe gépelt `www.amazon.com` URL segítségével nyithat egy kapcsolatot az Amazon adatbázisrendszerrel. Az Amazon webszervere egy olyan „honlapot" szolgáltat a felhasználónak, amely tartalmazza a felhasználó által kifejezhető műveletekhez szükséges űrlapokat, menüket és gombokat. A felhasználó beállíthat például egy `Könyvek` menüpontot, és egy űrlapon keresztül megadhatja azoknak a könyveknek a címét is, amelyek érdeklik őt. A kliens böngészője átküldi ezt az információt az Amazon webszervernek, és ezután a webszervernek le kell egyeztetni a kliens kérelmének a végrehajtását a következő szinttel, azaz az alkalmazási szinttel.

9.1.2. Az alkalmazásszerver-réteg

Az alkalmazásréteg feladata a webszervertől érkező kéréseknek az adatbázis adatainak segítségével történő megválaszolása. Minden egyes webszerverfolyamat



9.1. ábra. A háromrétegű architektúra

elindíthat egy vagy több alkalmazásszintű folyamatot a kérések végrehajtásához. Ezek a folyamatok lehetnek egy vagy akár több gépen is, sőt lehetnek a webszerver folyamataival megegyező vagy akár attól különböző gépen is.

Az alkalmazásszinten végzett műveleteket általában az adatbázis-műveletek szervezett *üzleti logikájának* nevezik. Azaz, egy programozó megtervezi a potenciális ügyfelek kéréseire adott válaszadási módszereket, majd meg is valósítja ezeket a stratégiákat.

A jelen munkánkban használt Amazon.com könyves példája esetén ez utóbbi válasz az Amazon által a könyvekre megjeleníthető adatokhoz tartozó weblapelemek lehetnek. Ezen adat tartalmazhatja a könyv címét, szerzőjét, árát és sok egyéb információt is a szóban forgó könyvről, sőt tartalmazhat további információkra vonatkozó hivatkozásokat is, mint például: áttekintéseket, további elérhető könyvkereskedőket és hasonló könyveket.

Egy egyszerűbb felépítésű rendszerben az alkalmazásréteg akár közvetlenül is átadhatja az adatbázis-lekérdezéseket az adatbázisszintnek, sőt a lekérdezé-

sek eredményét is összeszerkesztheti (természetesen egy HTML-dokumentum formájában). Bonyolultabb rendszerek esetében viszont az alkalmazásszintnek több olyan alrétege is lehet, amelynek van saját folyamata. Egy általános architektúrának van egy „objektumokat" támogató alrétege is. Ezen objektumok tartalmazhatnak adatokat is, egy „könyv objektum" esetén ilyen lehet például egy könyvnek a címe, illetve az ára. Az objektum adatai az adatbázis-lekérdezésekben is elérhetők lesznek. Az objektumnak – az alkalmazásszint folyamatai által is meghívható – metódusai is lehetnek. Sőt ezek a metódusok meghívásuk esetén eredményezhetnek további adatbázis-lekérdezéseket is.

Egy másik alréteg felelhet az *adatbázis-integrációért*. Vagyis több egymástól független adatbázis is támogathatja a műveleteket, és így lehetséges, hogy egynél több adatbázis egyidejű lekérdezése is szükséges egy lekérdezéshez. A kérdésekre különböző forrásokból nyert válaszokat az integrációs alrétegen kell összeraknunk. Az integrációt tovább bonyolíthatja, hogy az adatbázisok pár fontos dologban nem feltétlenül kompatibilisek. Az információ integrációjának technikáit máshol fogjuk elemezni. Most tekintsük a következő elképzelt példát.

9.1. példa. Az Amazon adatbázisában szereplő könyveknek az árai dollárban vannak megadva. Tegyük fel azonban, hogy van egy európai vásárló, akinek az azonosító adatai egy másik – Európában található – adatbázisban vannak letárolva, és a számla információi is euróban vannak megadva. Az integrációs szintnek tudnia kell arról, hogy különbség van a könyvek adatbázistól kapott ár értékének és az ügyfél részére készített számlán szereplő ár értékének valutaneme között. □

9.1.3. Az adatbázisréteg

A többi réteghez hasonlóan az adatbázisrétegen is több folyamat futhat, illetve a folyamatok több gépre el lehetnek osztva, de lehetnek azonos gépen is. Az adatbázisréteg az alkalmazásrétegtől kapott lekérdezéseket hajtja végre, és néhány adat tárolását is elvégezheti. Vegyünk például egy több sort eredményező lekérdezést, egy ilyen lekérdezés eredményét soronként adhatjuk át az alkalmazásréteg azon folyamatának, amely a kérést kezdeményezte.

Mivel az adatbázis-kapcsolatok létrehozása időigényes, ezért általában egy nagyobb számú kapcsolatot tartunk nyitva, az alkalmazás folyamatainak pedig megengedjük ezeknek a kapcsolatoknak a megosztását. A kapcsolatot minden egyes alkalmazásfolyamatnak ugyanazzal az állapottal kell visszaadnia, amelyben átvette azért, hogy elkerüljük az alkalmazás folyamatainak nem várt kölcsönhatásait.

Ez a fejezet egy adatbázisszint megvalósításának a miként jéről szól. Ehhez főként a következőket kell megtanulnunk:

1. Hogyan tesszük lehetővé, hogy egy adatbázis elérhető lehessen egy „közönséges" programból, amelyet valamilyen megszokott nyelvben – például C-ben vagy Javában – írtunk meg?

2. Hogyan kezelhetjük az SQL és a megszokott programozási nyelvek által támogatott adattípusok közötti különbségeket? Gyakorlatban a lekérdezések eredményei olyan relációk, amelyeket a megszokott nyelvek közvetlenül nem támogatnak.

3. Hogyan kezelhetjük egy konkrét adatbázishoz tartozó kapcsolatainkat, ha ezeken a kapcsolatokon több rövid lefutású folyamat is osztozhat?

9.2. Az SQL-környezet

Ebben a részben megvizsgáljuk egy adatbázis-kezelő rendszer környezetét, magukat az adatbázisokat és azokat a programokat, amelyek az adatbázisok feldolgozását segítik. Megmutatjuk, hogyan lesznek az adatbázisok klaszterekbe, katalógusokba, illetve sémákba szervezve, továbbá megvizsgáljuk a programok és az általuk feldolgozott adatok kapcsolatát. Mivel egy-egy adatbázis-kezelő vizsgálatakor ezen a területen sok implementációfüggő tényezővel kell számolni, ezért itt csak az SQL-szabványban felmerülő ötleteket próbáljuk meg bemutatni. A 9.5., 9.6., illetve a 9.7. alfejezetekben megmutatjuk, hogyan jelennek meg ezek a magasabb szintű elképzelések egy olyan „hívásszintű felületben", amely a programozóknak szükséges a közvetlen adatbáziskapcsolat létrehozásához.

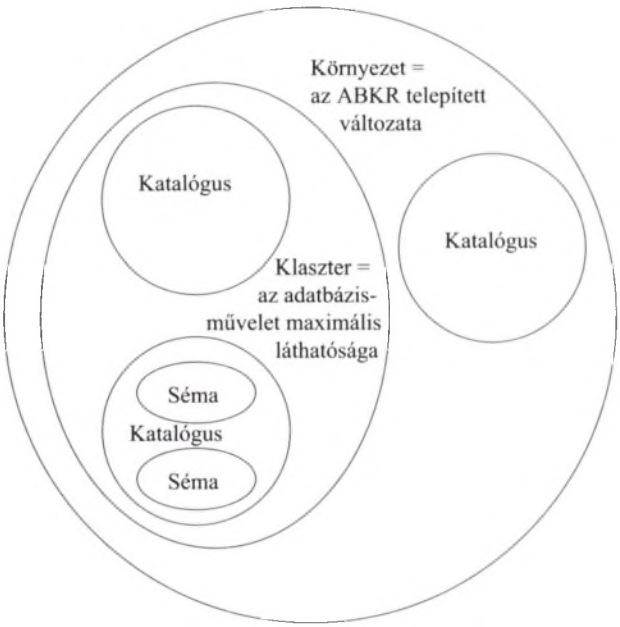
9.2.1. Környezetek

Az *SQL-környezet* egy olyan keretrendszer, amelyben adatokról és az azokat feldolgozó programokról beszélhetünk – az elnevezés a gyakorlatban általában egy telepített adatbázis-kezelő rendszerre vonatkozik. Például, ha egy ABC nevű vállalat megvásárolja a Megatron 2010 adatbázis-kezelő rendszerét, hogy a vállalat számítógépein ezt a rendszert futtathassa, akkor az ezeken a gépeken futó rendszerek tartalmazni fognak egy SQL-környezetet.

Az eddigiekben tárgyalt adatbáziselemek – így a táblák, nézettáblák, triggerek stb. – egy SQL-környezeten belül vannak definiálva. Ezen adatbáziselemek hierarchikus rendszerbe vannak szervezve úgy, hogy a hierarchiában minden egyes elem egy jól meghatározható szereppel bír. Az SQL-szabvány által definiált struktúrákat a 9.2. ábrán szemléltetjük.

Röviden összefoglalva: egy adatszervezés a következő szerkezeteket tartalmazhatja:

1. *Sémák.* Táblák, nézettáblák, önálló megszorítások, triggerek, PSM-modulok és más típusú információk gyűjteménye (lásd még a 9.2.2. alfejezet „További sémaelemek" nevű keretezett részében). A sémák az adatszervezés elemi összetevői. A séma fogalom talán a hétköznapi „adatbázis" fogalomhoz áll a legközelebb, de mint azt a 3. pontban látni fogjuk, kevesebb annál.



9.2. ábra. A környezetet alkotó adatbáziselemek struktúrája

2. *Katalógusok.* Szerepük az egységes terminológia biztosítása szempontjából lényeges. Minden katalógus alá egy vagy több séma tartozik, egy katalóguson belül a sémákat egyedi nevükkel lehet megkülönböztetni egymástól. Minden katalógus tartalmaz egy speciális `INFORMATION_SCHEMA` nevű sémát, amely a katalógusban lévő sémákról tartalmaz további információkat.

3. *Klaszterek.* Katalógusok gyűjteményei. Minden felhasználóhoz hozzá van rendelve egy klaszter: azok a katalógusok, amelyeket az illető felhasználó elérhet (a katalógusokhoz és más elemekhez való hozzáférési jogosultságok beállítását lásd a 10.1. alfejezetben). Egy klaszter tulajdonképpen azt a maximális láthatóságot jelenti, amely felett egy lekérdezés még végrehajtható, tehát egy átlagos felhasználó számára a klaszter tulajdonképpen maga az „adatbázis".

9.2.2. Sémák

Egy séma megadásának legegyszerűbb formája a következő:

`CREATE SCHEMA` <sémanév> <elemdeklarációk>

Az elemdeklarációk különféle alakjait már korábban a 2.3, 8.1.1., 7.5.1. és a 9.4.1. alfejezetekben megtárgyaltuk.

9.2. példa. Deklarálhatunk egy sémát, amely tartalmazza a filmekkel kapcsolatos példákban megismert öt relációt, és tartalmaz további nézettáblákat és más elemeket egyaránt. A 9.3. ábrán szemléltetjük egy ilyen deklaráció szerkezetét. □

```
CREATE SCHEMA FilmSéma
   CREATE TABLE FilmSzínész ... mint a 7.3. ábrán
      A másik 4 táblához tartozó create-table utasítások:
   CREATE VIEW FilmProd ... mint a 8.2. példában
      További nézettáblák deklarációi:
   CREATE ASSERTION GazdagElnök ... mint a 7.11. példában
```

9.3. ábra. Egy séma deklarációja

Egyáltalán nem kötelező a teljes sémát egyszerre deklarálnunk. A megfelelő `CREATE`, `DROP` vagy `ALTER` utasítások segítségével akár meg is változtathatjuk, illetve bővíthetjük az „aktuális" sémát, például egy `CREATE TABLE` kulcsszót követő új táblájának deklarációjával. Az „aktuális" sémát a `SET SCHEMA` paranccsal változtathatjuk meg. Például a

```
SET SCHEMA FilmSéma;
```

utasítás a 9.3. ábrán leírt sémát teszi aktuális sémává. A sémaelemek ezt követő deklarációi ebbe a sémába kerülnek, és a `DROP` vagy az `ALTER` utasítások is az adott sémában már létező elemekre fognak vonatkozni.

9.2.3. Katalógusok

Ugyanúgy, ahogyan a sémaelemek – mint a táblák – a sémán belül lettek létrehozva, egy sémát a katalóguson belül lehet létrehozni vagy módosítani. Elviekben a katalógusok létrehozásának és sokszorosításának a folyamata ugyanolyannak tekinthető, mint a sémák létrehozásának és sokszorosításának a menete. A SQL viszont sajnos nem definiál szabványos módszert ennek kivitelezésére, mint például egy:

`CREATE CATALOG` <katalógusnév>

jellegű utasítást, amelyet az adott katalógushoz tartozó sémáknak, illetve ezen sémák deklarációjának egy listája követ.

Ennek ellenére az SQL biztosít egy

`SET CATALOG` <katalógusnév>

formájú utasítást, amellyel kiválaszthatjuk az „aktuális" katalógust, amelybe az újonnan definiált sémák kerülnek, illetve amelyre a sémamódosító műveletek vonatkoznak, ha nem adunk meg más egyértelmű katalógusnevet.

További sémaelemek

Van még néhány, korábban még nem említett sémaelem, amely bizonyos esetekben hasznos lehet. Ezek az alábbiak:

- *Tartományok:* Értékek vagy sima adattípusok halmazai. Mostanában nem túl gyakran használják őket, mivel az objektumrelációs ABKR-ek sokkal hatékonyabb típuslétrehozási technikákat nyújtanak (lásd 10.4. alfejezet).
- *Karakterkészlet:* Szimbólumok és kódolási metódusaiknak a halmazai. Az ASCII és a Unicode a legelterjedtebb választási lehetőségek.
- *Gyűjtemények:* Egy gyűjtemény meghatározza, mely karakterek „kisebbek mint" a többi. Használhatjuk például az ASCII-kód által meghatározott rendezést, vagy azonos módon is kezelhetjük a nagy-, illetve kisbetűket, és emellett a nem betű karakterek összehasonlításától is eltekinthetünk.
- *Engedélyezési utasítások:* Ezek kezelik, hogy ki férhet hozzá egy séma elemeihez. A jogok engedélyezéséről bővebben szót ejtünk majd a 10.1. alfejezetben.
- *Tárolt eljárások:* Ezek tulajdonképpen futtatható kódok. A 9.4. alfejezetben tárgyaljuk őket részletesen.

9.2.4. Kliensek és szerverek az SQL-környezetben

Egy SQL-környezet több mint katalógusok és sémák gyűjteménye. Olyan elemeket is tartalmaz, amelyeknek célja a katalógusok és sémák által reprezentált adatbázison vagy adatbázisokon végezhető műveletek támogatása. Az SQL-szabvány alapján egy SQL-környezetben alapvetően kétféle folyamatról beszélünk: SQL-kliensekről és SQL-szerverekről.

A 9.1. ábra alapján egy „SQL-szerver" az „adatbázisszerver" szerepét látja el. Az „SQL-kliens" pedig az ábrán szereplő alkalmazásszerverhez hasonlít. Az SQL-szabvány viszont nem határoz meg olyan folyamatokat, mint a 9.1. ábrán szereplő „webszerverek" vagy a „kliensek".

9.2.5. Kapcsolatteremtés

Ha egy olyan számítógépen akarunk SQL-alapú alkalmazásokat futtatni, amelyen egy SQL-kliens érhető el, akkor kapcsolatot kell teremtenünk valamelyik SQL-szerverrel a következő SQL-utasítás végrehajtásával:

Sémaelemek teljes neve

Egy olyan sémaelemnek, mint egy táblának a neve formálisan az azt tartalmazó katalógus nevének, a hozzá tartozó séma nevének és saját nevének ebben a sorrendben, pontokkal elválasztott összefűzéséből áll. Eszerint például a `FilmKatalógus` katalógusban lévő, `FilmSéma` sémabeli `Filmek` tábla nevére az alábbi módon hivatkozhatunk:

```
FilmKatalógus.FilmSéma.Filmek
```

Ha egy sémaelem az aktuális katalógusban van vagy alapértelmezett, akkor a megnevezésekor elhagyhatjuk a katalógusnevet. Ezen felül, ha egy elem az aktuális sémában van vagy alapértelmezett, akkor a megnevezésekor a sémanevet is elhagyhatjuk, és ilyenkor csak a sémaelem nevét használhatjuk a sémaelem megnevezésére. Viszont a teljes név lehetőséget nyújt számunkra, hogy egy, az aktuális sémán vagy katalóguson kívüli elemet elérhessünk.

```
CONNECT TO <szerver neve> AS <kapcsolat neve>
    AUTHORIZATION <név és jelszó>
```

A szerver neve az SQL telepítési névtől függ. A legtöbb adatbázis-kezelőnél a `DEFAULT` kulcsszót beírhatjuk szervernévnek, és ekkor a felhasználó az SQL telepítésénél megadott „alapértelmezett szerverhez" fog csatlakozni. A fentiek alapján az `AUTHORIZATION` záradékot a felhasználó neve és jelszava követi. Ez utóbbi a tipikus megadási módja annak, ahogyan a szerver azonosítani tud egy felhasználót, azonban ettől eltérő karakterlánc is követheti az `AUTHORIZATION` záradékot.

A kapcsolatnevet használhatjuk a későbbiekben a megnevezett szerverrel való kapcsolatra hivatkozáskor. Erre azért van szükség, mert az SQL lehetőséget nyújt egyidejűleg több kapcsolat felépítésére, de ezek közül egyszerre csak egy lehet aktív. A kapcsolatok között választhatunk, például a `kapcs1` aktív kapcsolattá tételéhez a következő utasítást kell kiadnunk:

```
SET CONNECTION kapcs1;
```

Ilyenkor a korábban aktív kapcsolat alvó kapcsolattá válik, amíg ismét nem aktiváljuk egy újabb `SET CONNECTION` utasítással.

A nevet használhatjuk a kapcsolat megszüntetésére is. A `kapcs1` megszüntethető az alábbi utasítással:

```
DISCONNECT kapcs1;
```

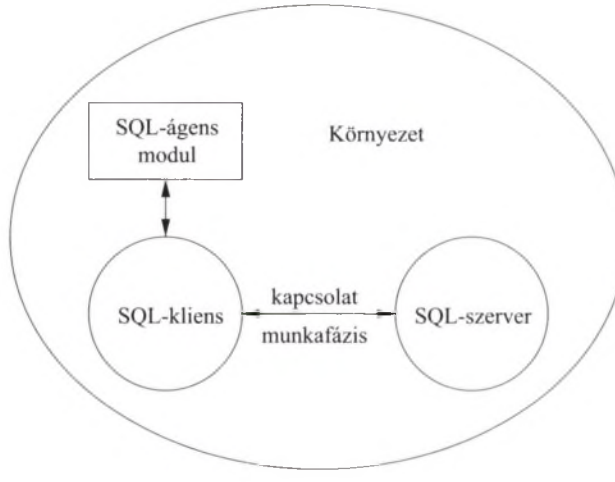
Ennek hatására a `kapcs1` terminál, tehát nem alvó lesz, és hivatkozni sem lehet rá a továbbiakban.

Ha egy kapcsolatfelépítési művelet után később nem akarunk a létrehozott kapcsolat nevére hivatkozni, akkor a `CONNECT TO` utasítás `AS` kulcsszavát és a mögötte következő kapcsolatazonosító nevét elhagyhatjuk. Sőt a kapcsolatfelépítési műveletet teljesen el is hagyhatjuk: ilyenkor, ha végrehajtunk egy SQL-utasítást, akkor az SQL-parancsértelmező automatikusan felépít egy kapcsolatot az alapértelmezés szerint elérhető SQL-adatbázis-kezelővel.

9.2.6. Munkafázisok

Egy kapcsolat aktív állapota alatt végrehajtott SQL-utasítások egy ún. munkafázist képeznek. Egy munkafázis az őt létrehozó kapcsolathoz van kötve. Például, amikor a kapcsolatot alvó állapotba helyezik, akkor a hozzá tartozó munkafázis is alvó állapotba kerül, majd amikor a kapcsolatot újra aktiválják a `SET CONNECTION` utasítással, akkor a hozzá tartozó munkafázis is aktiválva lesz. Ez alapján a 9.4. ábrán a kliens és a szerver kommunikációját két különböző szemszögből vizsgálva nevezhetjük kapcsolatnak, illetve munkafázisnak.

Minden munkafázishoz hozzá tartozik egy aktuális katalógus és ezen belül egy aktuális séma. Ezeket módosíthatjuk a `SET SCHEMA`, valamint a `SET CATALOG` utasításokkal a 9.2.2. és 9.2.3. alfejezetben tárgyalt módon. Munkafázisonként történik az adatbázist elérni kívánó felhasználó igazoltatása is, amelyre a 10.1. alfejezetben még visszatérünk.



9.4. ábra. Az SQL-kliensek és -szerverek együttműködése

9.2.7. Modulok

A *modul* elnevezést az SQL-ben az alkalmazói programok fogalmára használják. Az SQL-szabvány háromféle modult különböztet meg, de a szabvány csak

Az SQL-szabvány nyelvei

Egy – az SQL-szabványnak megfelelő – megvalósításnak támogatnia kell a következő hét befogadó nyelv legalább egyikét: ADA, C, Cobol, Fortran, M (hivatalosan Mumpsnak nevezik, és általában orvosi körökben használják), Pascal, illetve PL/I. Mi a példáinkban C-t használunk.

annyit követel meg, hogy az SQL-implementációknak ezek közül legalább egyet biztosítaniuk kell a felhasználók számára.

1. *Általános SQL-felület.* A felhasználók SQL-utasításokat írhatnak be, és az SQL-szerverrel végre is hajthatják azokat. Ezen megoldásnál minden lekérdezés, illetve minden más utasítás is modulnak minősül. Bár ebben a könyvben példáinkban leggyakrabban ezt a módszert használjuk, a gyakorlatban történő alkalmazása viszont ritka.

2. *Beágyazott SQL.* Ezzel foglalkozunk a 9.3. alfejezetben. A beágyazott SQL-utasításokat általában egy előfeldolgozó program alakítja befogadó nyelvi utasításokhoz illeszkedő SQL-rendszerbeli függvény- vagy eljáráshívásokká. A lefordított befogadó nyelvi program – az előzőleg említett függvényhívásokat is beleértve – alkot egy modult.

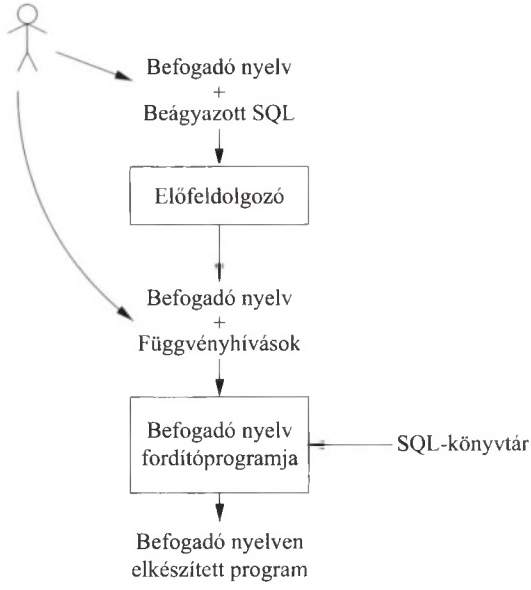
3. *Valódi modulok.* Az SQL által támogatott legáltalánosabb modulok, amelyek különféle tárolt eljárásokra és függvényekre épülnek (ezek akár valamilyen befogadó nyelven, akár SQL nyelven készülhetnek). Ezek az eljárások és függvények paraméterek átadásával és osztott változókkal kommunikálnak egymással. A PSM-modulok (lásd 9.4. alfejezet) alkalmas példák ezekre a modulokra.

Egy modul egy végrehajtását nevezzük *SQL-ágens*nek. A 9.4. ábrán egy SQL-ágenst és egy modult egyetlen alkotóelemként tüntettünk fel (az ábrán ez az alkotóelem egy SQL-kliens segítségével kommunikál egy SQL-szerverrel). Ennek ellenére ügyeljünk az ágens és a modul közt meglevő különbségekre, amelyek leginkább az operációs rendszerek program- és folyamatabsztrakciója közti analógiára hasonlítanak: a modul egy végrehajtható kód, az ágens pedig ezen kód végrehajtása.

9.3. Az SQL és a befogadó nyelv közötti felület

Az eddigiekben csak az ún. *általános SQL-felület*tel foglalkoztunk, amikor azt feltételeztük, hogy az SQL-parancsainkat egy azok bekérésére és végrehajtására felkészített SQL-interpreterrel hajtatjuk végre. Bár majdnem minden ABKR

nyújt erre lehetőséget, mégis ez a használati mód valójában igen ritka. A 9.1. ábrán bemutatottakhoz hasonló valós rendszerekben ezeket az alkalmazásokat általában valamilyen ún. befogadó nyelven (például C nyelven) készítik, de az alkalmazás egyes részei valójában SQL-utasítások.



9.5. ábra. Beágyazott SQL-utasításokat is tartalmazó program feldolgozása

A 9.5. ábra szemlélteti egy SQL-utasításokat tartalmazó tipikus programozási rendszer vázlatát. Láthatjuk, hogy a programozó feladata a befogadó nyelven megfogalmazott program megírása, amely „beágyazott" SQL-utasításokat is tartalmaz. A beágyazás az elhelyezkedésétől függően kétféleképpen történhet.

1. *Hívásszintű felület.* Tartalmaz egy könyvtárat és ezen könyvtárcsomag függvényeinek és eljárásainak hívásait végző befogadó nyelvben írt beágyazott SQL-t is. Az SQL-utasítások általában a könyvtárbeli eljárásoknak a karaktersor argumentumai. Ezt a megközelítést, amelyet gyakran hívásszintű felületnek vagy CLI-nek neveznek, a 9.5. alfejezetben tárgyaljuk. Ezt az elképzelést szemlélteti a 9.5. ábrán szereplő, a felhasználó és a befogadó nyelv között lévő görbített nyíl.

2. *Közvetlenül beágyazott SQL.* Itt a befogadó nyelven megfogalmazott teljes program (a beágyazott SQL-utasításokkal együtt) átkerül egy előfeldolgozóhoz, amely átalakítja a beágyazott SQL-utasításokat a befogadó nyelven is értelmezhető alakra. Az SQL-utasításokat általában könyvtári függvény-, illetve eljáráshívásokkal helyettesíti. Azaz a CLI és a közvetlenül beágyazott SQL közötti különbség inkább csak „kinézet és felfogás"

kérdése, és nem lényegi eltérés. A változtatások után az előfeldolgozott befogadónyelvi program a megszokott módon kerül lefordításra, és az adatbázison végzendő műveletek végrehajtását könyvtári hívásokon keresztül végzi el.

Ebben az alfejezetben az SQL-szabvány a befogadó nyelv (főként a C) közvetlen beágyazásával foglalkozó részeit tekintjük át. Sőt, néhány olyan fogalmat is bevezetünk, mint a kurzor, amely mindegyik vagy majdnem mindegyik beágyazott SQL-t támogató rendszerben előfordul.

9.3.1. A típuseltérés problémája

Az SQL-utasítások és a hagyományos programozási nyelvek összekapcsolásának alapvető problémája az úgynevezett *típuseltérés*, azaz az a tény, hogy az SQL adatmodellje lényegesen eltér a többi programozási nyelv modelljétől. Mint tudjuk, az SQL magját a relációs adatmodell képezi. Azonban a C és a hozzá hasonló programozási nyelvek egy olyan modellt használnak, amely tartalmazhat egész számokat, valós számokat, aritmetikai értékeket, karaktereket, mutatókat, rekordstruktúrákat, tömböket és egyéb hasonló struktúrákat. Például a C programozási nyelv sem támogatja közvetlenül nyelvi eszközökkel a halmazfogalom alkalmazását, míg az SQL nem támogatja a tömb, mutató és több más programnyelvi elem alkalmazását. Ezért az SQL és más nyelvek közötti adatátvitel nem közvetlen, így szükségesek az elkövetkezőkben ismertetett mechanizmusok, amelyek támogatják az SQL-t és a befogadó nyelven írt kódrészeket egyaránt tartalmazó alkalmazások fejlesztését.

Az olvasó első ránézésre azt is gondolhatná, hogy a legjobb megoldás az, hogy egyáltalán ne keverjük a programozási nyelveket: vagy végezzünk minden számítást SQL-ben, vagy egyáltalán ne is használjuk az SQL-t. Könnyen belátható, hogy ez az út nem járható, mivel ha szükségünk van adatbázisok elérésére, akkor erre az SQL egy jól használható, hatékony és magas szintű eszköz. Az SQL-t használva a programozónak nem kell törődnie azzal, hogyan szervezi meg az adatok hatékony tárolását a rendelkezésre álló háttértáron, illetve azzal sem kell törődni, hogy az adatbázison végzett műveleteknél hogyan használják ki hatékonyan a tárolt struktúrákat.

Másrészről vannak dolgok, amelyeket egyáltalán nem lehet SQL nyelven kifejezni. Például nem írhatunk egy olyan SQL-lekérdezést, amely kiszámítja egy tetszőleges szám faktoriálisát. Ilyen számítások például C vagy hozzá hasonló nyelvek[1] bármelyikén könnyen elvégezhetők. Hasonlóan nem lehetséges SQL nyelven például a lekérdezések eredményének grafikus megjelenítését specifikálni. Látható, hogy a gyakorlati életben használt adatfeldolgozó programok elké-

[1] Bánjunk óvatosan a választással, ugyanis vannak olyan kiterjesztései az alap SQL nyelvnek (mint például a 10.2. alfejezetben tárgyalt rekurzív SQL vagy a 9.4. alfejezetben tárgyalt SQL/PSM), amelyek „Turing-teljességet" garantálnak, azaz a képességet ahhoz, hogy mindent kiszámíthassunk, ami bármely más nyelven is kiszámítható lenne. Ezek a kiterjesztések viszont sohasem általános célú számításokra lettek bevezetve, és ezeket nem tekinthetjük általános célú nyelveknek sem.

szítéséhez szükség van mind az SQL, mind pedig a hagyományos, ún. befogadó programozási nyelvekre.

9.3.2. Az SQL és a befogadó nyelv közötti interfész

Ha egy befogadó nyelven írt programban SQL-utasítást kívánunk használni, akkor ezt jeleznünk kell az előfeldolgozónak az utasítás elé írt `EXEC SQL` kulcsszó segítségével. Az *osztott változók* használatával információt cserélhetünk a befogadó nyelvi program és a csak SQL-utasításokkal elérhető adatbázis között. Ezen változók már mind a befogadó nyelvi, mind az SQL-utasításokban szerepelhetnek. Az SQL-utasításokon belül az osztott változókat egy kettőspont előzi meg, a befogadó nyelvi utasításokban viszont kettőspont nélkül szerepelnek.

Az SQL-szabvány definiál egy `SQLSTATE` nevű változót az SQL és a befogadó nyelvi környezetek összekapcsolására. Ez a változó öt karaktert tartalmaz (általában egy ötelemű karakteres tömb). Az adatbázis elérését támogató könyvtárak úgy vannak elkészítve, hogy visszatéréskor ebben a változóban helyezik el a végrehajtott SQL-művelet során fellépett esetleges problémákat leíró kódokat. Az SQL-szabvány jó néhány 5 karakter hosszú kódot és a hozzá tartozó jelentést is meghatározza.

Például a `'00000'` (öt darab nulla számjegy) azt jelzi, hogy a függvény végrehajtása során nem léptek fel problémák, míg a `'02000'` kód egy lekérdezési művelet végrehajtása után azt jelezheti, hogy nincs a lekérdezésben megadott kritériumoknak eleget tevő sor az adatbázisban. Az utóbbi kód nagyon fontos, hiszen lehetővé teszi számunkra, hogy egy olyan ciklust fogalmazhassunk meg a befogadó nyelvben, amely egyenként megvizsgálja az adott reláció sorait, illetve amely az utolsó sor megvizsgálása után be is fejeződik.

9.3.3. A deklarációs rész

Az osztott változók deklarációját két beágyazott SQL-utasítás közé kell tenni az alábbi minta szerint:

```
EXEC SQL BEGIN DECLARE SECTION;
    ...
EXEC SQL END DECLARE SECTION;
```

A fenti két SQL-utasítás közti részt (amelyet a példában kipontoztunk) *deklarációs rész*nek nevezzük. A deklarációs részben a változó deklaráció szintaxisa megegyezik a befogadó nyelven megszokott deklarációs szintaxissal, és mivel a változókat a befogadó nyelven is és az SQL-ből is el akarjuk érni, ezért csak olyan adattípussal deklarálhatjuk őket, amelyek mindkét nyelven egyaránt elérhetők (például egészek, valósak, karakterláncok, tömbök).

9.3. példa. Az alábbi utasítások előfordulhatnak például a Stúdió relációt módosító C függvényekben.

```
EXEC SQL BEGIN DECLARE SECTION;
    char stúdióNév[50], stúdióCím[256];
    char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
```

Az első és az utolsó (negyedik) utasítás a deklarációs rész kezdetét és végét jelzik. A középső két sor két változót definiál: a stúdióNév és a stúdióCím változókat. Ezek mindketten karakterekből álló tömb típusú változók: a későbbiekben a Stúdió relációba beszúrni kívánt sorok stúdiónév- és stúdiócímértékeit fogjuk bennük tárolni. A harmadik sorban pedig az SQLSTATE változót egy hat elem hosszú karaktertömbnek[2] definiáljuk. □

9.3.4. Osztott változók használata

Az osztott változókat az SQL-utasítások azon részein használhatjuk, ahol vagy konstans értéket várunk, vagy a konstans érték megengedett. Mint már azt korábban említettük, a nevét kettősponttal megelőzve használjuk. A következő példában a 9.3. példában definiált változók értékét fogjuk beszúrni a Stúdió relációba.

9.4. példa. A 9.6. ábrán egy olyan C nyelvű – beolvasStúdió nevű – függvény vázlatát láthatjuk, amely a felhasználótól egy stúdiónevet és egy stúdiócímet vár bemenetként, és beszúrja a beadott adatokat a Stúdió táblába. Az 1–4. sor megegyezik a 9.3. példában bemutatott változódeklarációval. A példából kihagytuk a képernyőre író és a billentyűzetről olvasó sorokat (azok helyét egy megjegyzésben megadtuk).

Az 5. és a 6. sorban egy INSERT utasítást láthatunk, amit az említett EXEC SQL kulcsszavak vezetnek be azt jelölve, hogy ez nem egy szokásos C nyelvű utasítás, hanem egy beágyazott SQL-utasítás. Az említett előfordító program majd elvégzi a szükséges átalakításokat az EXEC SQL kulcsszavakkal bevezetett utasításokon. Az 5. és 6. sorokban megadott beszúrandó értékek nem szövegkonstansok, mint azt az SQL-ben azon a helyen írhatnánk (lásd például a 6.34. példát), hanem (megosztott elérésű) változók, amelyeknek az aktuális értéke lesz behelyettesítve az SQL-utasítás kiértékelésekor. A program a 6. sorban megadott változók aktuális értékéből állít össze egy sort, amelyet beszúr a táblába. □

[2] Az 5 karakteres SQLSTATE változó számára azért foglaltunk le 6 karaktert, mert a továbbiakban olyan C-függvényekkel fogjuk azt kezelni, mint amilyen pl. az strcmp, amelyik elvárja, hogy a karakterláncokat a '\0' karakterrel zárjuk le. Ezt a lezáró karaktert kell a 6. pozícióban tárolnunk. Ennek a 6. karakternek a kezdeti '\0'-ra való beállítását a programjainkban nem fogjuk feltüntetni.

```
    void beolvasStúdió() {

1)      EXEC SQL BEGIN DECLARE SECTION;
2)          char stúdióNév[50], stúdióCím[256];
3)          char SQLSTATE[6];
4)      EXEC SQL END DECLARE SECTION;

        /* Kiírja a képernyőre azt a felhasználónak
       szóló utasítást, hogy adjon be egy nevet és
       egy címet, majd a beadott válaszokat eltárolja
       a stúdióNév és stúdióCím változókba */

5)      EXEC SQL INSERT INTO Stúdió(név, lakcím)
6)               VALUES (:stúdióNév, :stúdióCím);
    }
```

9.6. ábra. Osztott változók alkalmazása egy új stúdió beszúrására

Az EXEC SQL kulcsszó segítségével az összes olyan SQL-utasítás beágyazható, amelynek nincs visszatérési értéke, azaz amely nem lekérdezés: beágyazhatók például a beszúrást, törlést, módosítást leíró utasítások, valamint az adatbázissémát manipuláló utasítások, amelyek például táblákat vagy nézettáblákat törölnek vagy hoznak létre.

A SELECT SQL-lekérdezések általában nem ágyazhatóak be közvetlenül a befogadó nyelvbe az ún. „típuseltérés" miatt. A lekérdezések egy multihalmazt adnak vissza eredményül, és a legtöbb programozási nyelv viszont nem támogatja közvetlenül a halmaz vagy a multihalmaz adatszerkezetet. Azaz a beágyazott SQL-nek az alábbi két módszer valamelyikét kell használni ahhoz, hogy össze tudja kapcsolni a lekérdezések eredményeit és a befogadó nyelvi programjait:

1. Az *egyetlen sort eredményező lekérdezések* a lekérdezés eredményeként létrejött eredménysort eltárolhatják akár osztott változókba oly módon, hogy az eredménysor egyes komponensei külön-külön osztott változókba lesznek elhelyezve.

2. *Kurzorok.* Egy olyan lekérdezés, amely egynél több sort (azaz egy sorhalmazt) is visszaadhat eredményül, csak úgy hajtható végre, ha egy *sormutatót* definiálunk hozzá. A sormutató majd befutja az eredményreláció összes sorát, és egy-egy eredménysor egyes komponenseit külön-külön osztott változókba elhelyezve juttathatjuk el az adatokat a befogadó programnak.

A következő alfejezetekben mindkét fenti lehetőséget megvizsgáljuk.

9.3.5. Egyetlen sort eredményező lekérdezések

Egy alkalmazásokba beágyazható egyetlen sort eredményező lekérdezés formája annyiban különbözik az SQL nyelv elemeinek bemutatásakor ismertetett közönséges SELECT utasítástól, hogy a SELECT záradék után egy INTO kulcsszót kell írni, e kulcsszó mögé pedig fel kell sorolni azokat a változókat, amelyekbe a lekérdezés eredményeként visszakapott adatokat el akarjuk tárolni. A már megismert szabályok alapján az itt felsorolt változók neve elé egy-egy kettőspont karaktert kell írni. Amennyiben a lekérdezés egyetlen sort eredményez, úgy a megadott változók rendre felveszik az eredménysor komponenseiben képződött értékeket. Amennyiben a lekérdezés egyetlen sort sem eredményez, vagy éppen egynél több sort eredményez, akkor a felsorolt változók semmilyen értéket sem kapnak, és az adatbázisrendszer az SQLSTATE változóba beírja a megfelelő hibakódot.

9.5. példa. Most elkészítünk egy C nyelvű függvényt, amely bekéri egy stúdió nevét a felhasználótól, és kiírja a stúdió igazgatójának nettó jövedelmét. A program vázlatos forráskódját a 9.7. ábra tartalmazza. Az 1–5. sorok tartalmazzák a szükséges változók deklarációját. Ezután – a megjegyzéssel jelölt helyen – következne a stúdió nevét bekérő programrész (az ezeket megvalósító C nyelvű sorokat a példából elhagytuk).

```
    void nettóFizetésKiírása() {

1)      EXEC SQL BEGIN DECLARE SECTION;
2)          char stúdióNév[50];
3)          int igazgatóNettóFizetése;
4)          char SQLSTATE[6];
5)      EXEC SQL END DECLARE SECTION;

        /* Kérjünk be a felhasználótól egy stúdiónevet,
      és a választ tegyük a stúdióNév változóba */

6)      EXEC SQL SELECT nettóBevétel
7)               INTO :igazgatóNettóFizetése
8)               FROM Stúdió, GyártásIrányító
9)               WHERE elnökAzon = azonosító AND
                       Stúdió.név = :stúdióNév;

        /* Itt kiírhatjuk az eredményt,
      miután megbizonyosodtunk arról, hogy az SQLSTATE
      változó öt darab nulla karaktert tartalmaz */
  }
```

9.7. ábra. Egy egyetlen sort eredményező lekérdezés beágyazása

Majd a 6–9. sorokban egy egyetlen sort eredményező lekérdezést láthatunk, amelynek felépítése hasonlít a korábbi fejezetekben már megismert lekérdezések szerkezetéhez. A korábban bemutatott lekérdezésekhez képest itt két különbséget is megfigyelhetünk: egyrészt a 7. sorban használjuk azt az `INTO` kulcsszóval bevezetett záradékot, amelyben megadjuk, hogy melyik változókban akarjuk visszakapni a lekérdezés eredményét; másrészt pedig a `stúdióNév` változó értékét helyezzük az SQL-utasítás feltételei közé a 9. sorba (ahol régebben ehelyett például szöveges konstansértékeket írtunk). E lekérdezés végrehajtása után egyetlen soros válaszra számítunk, és a válaszsor láthatóan egyetlen oszlopot tartalmaz, amely a visszakapott `nettóBevétel` attribútum értékét tárolja. Ezen egyetlen sor egyetlen oszlopának tartalma az `igazgatóNettóFizetése` nevű közös változóban lesz eltárolva. □

9.3.6. Sormutatók

Az SQL-lekérdezések és a befogadó nyelv összekapcsolásának legsokoldalúbb módszere egy ún. sormutató használata, amely végigmegy egy reláció sorain. A reláció lehet egy tárolt tábla vagy egy lekérdezés által előállított reláció is. Egy sormutató létrehozása és használata a következőképpen történhet:

1. Deklarálni kell a sormutatót. A sormutató deklarációjának legegyszerűbb módja a következő:

   `EXEC SQL DECLARE` <sormutatónév> `CURSOR FOR` <lekérdezés>

   A lekérdezés lehet akár egy egyszerű select-from-where lekérdezés vagy egy reláció neve. A sormutató a lekérdezés eredményeként meghatározott reláció sorain fut végig.

2. Ezután egy `EXEC SQL OPEN` utasítás következik, amelyet a sormutató neve követ. Ez az utasítás inicializálja a sormutatót a hozzá tartozó reláció első sorának pozíciójával.

3. Ezt követheti egy vagy több ún. *fetch utasítás*. Ennek az utasításnak a feladata a kurzorhoz tartozó reláció következő sorának meghatározása. Egy fetch utasítás általános alakja a következő:

   `EXEC SQL FETCH FROM` <sormutatónév> `INTO` <változók listája>

   A változók listáján a reláció sorának minden egyes attribútumához tartozik egy változó. Ha van elérhető sor, akkor ezeknek a változóknak az értéke a sor megfelelő komponensének az értéke lesz. Ha egy sormutatóval már nincs több elérhető sor, akkor a fetch utasítás nem tesz semmit a benne megadott változókba, a `SQLSTATE` változóba pedig a `'02000'` konstans értéket teszi, ami azt jelenti, hogy nincs több beolvasható sor.

4. Végül az `EXEC SQL CLOSE` utasítás következik, amelyet a sormutató neve követ. Ez az utasítás lezárja a sormutatót, amely így már nem fut tovább a reláció sorain. Egy újabb `OPEN` utasítással azonban újrainicializálható a sormutató, és ekkor a szóban forgó reláció sorainak új értékein fut majd végig.

9.6. példa. Meg szeretnénk tudni azoknak a filmgyártásvezetőknek a nevét, akiknek a nettó jövedelmük egy előre megadott, exponenciálisan növekedően meghatározott sávok valamelyikébe esik; az egyes sávokat a jövedelem számjegyeinek a száma szerint definiáltuk. Egy olyan lekérdezést készítünk, amely beolvassa a `GyártásIrányító` tábla összes sorának a `nettóBevétel` attribútumát egy `jövedelem` nevű osztott változóba. A tábla sorain egy `irányítókSormutató` nevű sormutatóval haladunk végig, kiszámoljuk az aktuálisan beolvasott jövedelemérték számjegyeinek a számát, ez alapján határozzuk meg, hogy a beolvasott jövedelem mely sávba tartozik, majd az erre a célra létrehozott `számláló` tömb megfelelő elemét eggyel megnöveljük.

```
1)  void jövedelemSávok() {

2)      int i, számjegyek, számláló[15];
3)      EXEC SQL BEGIN DECLARE SECTION;
4)          int jövedelem;
5)          char SQLSTATE[6];
6)      EXEC SQL END DECLARE SECTION;
7)      EXEC SQL DECLARE irányítókSormutató CURSOR FOR
8)          SELECT nettóBevétel FROM GyártásIrányító;

9)      EXEC SQL OPEN irányítókSormutató;
10)     for(i=1; i<15; i++) számláló[i] = 0;
11)     while(1) {
12)         EXEC SQL FETCH FROM irányítókSormutató
                INTO :jövedelem;
13)         if(NINCS_TÖBB_SOR) break;
14)         számjegyek = 1;
15)         while((jövedelem /= 10) > 0) számjegyek++;
16)         if(számjegyek <= 14) számláló[számjegyek]++;
        }
17)     EXEC SQL CLOSE irányítókSormutató;
18)     for(i=0; i<15; i++)
19)         printf("számjegyek = %d: előfordulások
                    száma = %d\n", i, számláló[i]);
    }
```

9.8. ábra. A jövedelmek exponenciálisan növekvő sávok szerinti osztályozása

A `jövedelemSávok` nevű C-függvény a 9.8. ábra 1. sorában kezdődik. A 2. sorban deklaráljuk azokat a változókat, amelyekre csak C-utasításokban kívánunk hivatkozni (beágyazott SQL-ben nem). Az itt deklarált `számláló` tömb minden egyes jövedelemsávhoz tartalmaz egy-egy számlálót. A `számjegyek` változóban számoljuk meg a `jövedelem` változóban levő érték leírásához szükséges számjegyek számát. Az `i` változót pedig tömbindexként használjuk a `számláló` tömb elemein történő végighaladáshoz.

A 3–6. sorokban deklaráljuk a `jövedelem` és a szokásos `SQLSTATE` változókat, amelyeket az SQL-utasításokból is elérhetünk. Az `irányítókSormutató` nevű sormutatót a 7. és 8. sorokban deklaráljuk, amely a 8. sorban megadott lekérdezés eredményeként létrejött tábla sorain megy végig. Ez a lekérdezés a `GyártásIrányító` tábla `nettóBevétel` oszlopának tartalmát adja vissza eredményül. A sormutatót a 9. sorban megnyitjuk, majd a 10. sorban inicializáljuk nulla értékkel az előbb említett `számláló` nevű tömb elemeit.

A függvény lényegi része a 11–16. sorokban látható. A 12. sorban olvassuk be a következő rekordot, a beolvasott érték a `jövedelem` változóba kerül. Mivel a 8. sorban deklarált lekérdezés egyoszlopos, ezért itt elég csak egy változót megadni (általában az oszlopok számával megegyező számú változót kell megadni). A 13. sorban ellenőrizzük azt, hogy volt-e még beolvasható adat. Itt egy `NINCS_TÖBB_SOR` nevű makrót adtunk meg, amelyet az alábbi módon definiálhatunk:

```
#define NINCS_TÖBB_SOR !(strcmp(SQLSTATE,"02000"))
```

Emlékezzünk rá, hogy a `"02000"` érték jelöli azt, ha nincs több adat a sormutató olvasása során. Ha nem sikerült adatot olvasni, akkor kiugrunk a ciklusból a 17. sorra.

Ha sikerült adatot olvasni, akkor a 14. sorban inicializáljuk a számjegyeket számláló változót 1-re. A 15. sorban megszámoljuk a beolvasott érték számjegyeinek a számát (minden egyes 10-zel való osztás után növeljük eggyel a számjegyeket számláló változót, amíg a 10-zel való osztások során nullát nem kapunk). Végül a 16. sorban megnöveljük eggyel a `számláló` tömb megfelelő elemét. Láthatjuk, hogy a programban a 15 vagy annál több jegyű számokkal nem foglalkozunk – a statisztikát nem befolyásoljuk az ilyen óriási számokkal (ilyenkor nem növelünk egyetlen tömbelemet sem).

A 17. sorban kezdődik a függvény kiíratási része. Lezárjuk a sormutatót, majd a 18–19. sorban kiírjuk a statisztikát tároló számláló tömb tartalmát. □

9.3.7. Sormutatóval történő módosítások

Amikor egy sormutatóval végigmegyünk egy adatbázistábla (vagyis egy adatbázisban tárolt reláció) sorain, az egyes sorokat nemcsak olvashatjuk, hanem módosíthatjuk is azok tartalmát, vagy akár törölhetjük is az illető sort. Az ilyen módosító és törlő `UPDATE` és `DELETE` utasítások szintaxisa megegyezik a 6.5. alfejezetben megismertekkel attól eltekintve, hogy itt a `WHERE` záradékban

csak a CURRENT OF kulcsszavakat adhatjuk meg, azután pedig annak a sormutatónak a nevét kell megadni, amelyikből a módosítandó vagy törlendő sort olvastuk. Természetesen egy ilyen módosító vagy törlő beágyazott SQL-utasítás végrehajtása bekerülhet a befogadó nyelven írt elágazások egy-egy ágába attól függően, hogy az alkalmazásnak milyen feladatot kell megoldania.

```
1)  void bevételVáltozás() {

2)      EXEC SQL BEGIN DECLARE SECTION;
3)          int azonosító, jövedelem;
4)          char gyártóNév[31], gyártóCím[256], SQLSTATE[6];
5)      EXEC SQL END DECLARE SECTION;
6)      EXEC SQL DECLARE irányítókSormutató CURSOR FOR
                                    GyártásIrányító;

7)      EXEC SQL OPEN irányítókSormutató;
8)      while(1) {
9)          EXEC SQL FETCH FROM irányítókSormutató INTO
                    :gyártóNév, :gyártóCím,
                    :azonosító, :jövedelem;
10)         if(NINCS_TÖBB_SOR) break;
11)         if (jövedelem < 1000)
12)             EXEC SQL DELETE FROM GyártásIrányító
                    WHERE CURRENT OF irányítókSormutató;
13)         else
14)             EXEC SQL UPDATE GyártásIrányító
                    SET nettóBevétel = 2 * nettóBevétel
                    WHERE CURRENT OF irányítókSormutató;
        }
15)     EXEC SQL CLOSE irányítókSormutató;
    }
```

9.9. ábra. A gyártó nettó bevételének módosítása

9.7. példa. A 9.9. ábrán egy olyan C-függvényt láthatunk, amely végigmegy a GyártásIrányító tábla sorain, és az illető sorra vonatkozóan eldönti, hogy törölje-e azt vagy kétszerezze meg a nettó bevételének értékét. A 3. és 4. sorokban a GyártásIrányító négy attribútumának, illetve a szükséges SQLSTATE értékének a tárolására alkalmas változókat deklarálunk. Majd a 6. sorban az irányítókSormutató-t is deklaráljuk, hogy a GyártásIrányító reláció sorait végigolvashassuk a segítségével.

A 8–14. sorban található a ciklus, amelyben az irányítókSormutató nevű sormutató minden lépésben a GyártásIrányító reláció egyes soraira mutat. A 9. sorban kérjük be az aktuális sor értékeit – az erre a célra fenntartott –

négy változóba. Vegyük észre, hogy most csak a `jövedelem` változót használjuk. A 10. sor ellenőrzi, hogy van-e egyáltalán elérhető sor a `GyártásIrányító` relációban. Itt is a korábban már használt `NINCS_TÖBB_SOR` makrót használjuk feltételként (az `SQLSTATE` jelentése „nincs több sor", kódja `"02000"`).

A 11. sorban vizsgáljuk, hogy az aktuális nettó bevétel kevesebb-e 1000 $-nál. Amennyiben a válasz igen, akkor a 12. sorban lévő `DELETE` utasítás segítségével töröljük ezt a sort. Vegyük észre, hogy a `WHERE` záradékban a sormutatóra hivatkozunk, és így a `GyártásIrányító` aktuális sorára. Éppen arra a sorra, amelyet az előbb kértünk be. Ezt a sort töröljük a `GyártásIrányító` relációból. Ha a nettó bevétel legalább 1000 $, akkor a 14. sor pedig – az előzőekkel szemben – megduplázza az aktuális sor nettó bevételének értékét. □

9.3.8. Egyidejű módosítások elleni védelem

Tegyük fel, hogy a gyártásirányítóknak a nettó bevételeit vizsgáljuk a 9.8. ábra `jövedelemSávok` függvényét felhasználva, és hogy eközben valamilyen más alkalmazás módosítja a vizsgálatunk alapjául szolgáló `GyártásIrányító` relációt. Mit tehetnénk ekkor? A válasz: „Talán semmit." Megelégedhetünk a hozzávetőleges adatokkal, és figyelmen kívül hagyjuk azt, hány olyan végrehajtás van futtatás alatt, amely például sorokat fog törölni a relációból. Ekkor egyszerűen csak elfogadjuk a sormutató által visszaadott sorokat.

Az is elképzelhető viszont, hogy nem szeretnénk megengedni az olyan konkurens módosításokat, amelyek hatással lehetnek a sormutatónk által mutatott sorokra. Ehelyett inkább úgy tekintenénk a relációkra, mintha azok egy adott pillanatban levő állapotot tükröznének. A 6.6. alfejezetben szereplő tranzakciók jegyében a reláción futó sormutató kódját a reláción végezett egyéb műveletekkel együtt sorolható módon kívánjuk végrehajtani. Ezt a hatást érhetjük el a módosításokra *érzéketlen* sormutató deklarálásával.

9.8. példa. Most módosítsuk a 9.8. ábrában megadott beágyazott SQL-utasítás 7. és 8. sorát a következőkre:

```
7) EXEC SQL DECLARE irányítókSormutató INSENSITIVE CURSOR FOR
8)     SELECT nettóBevétel FROM GyártásIrányító;
```

Ekkor az adatbázisrendszer az `irányítókSormutató` nevű sormutató megnyitása és lezárása közben a `GyártásIrányító` táblán elvégzett módosításokat az alkalmazás elől teljesen eltakarja, vagyis ez nem fogja befolyásolni a visszakapott sorokat. □

Egy-egy sormutatóról esetleg biztosan tudhatjuk, hogy végigolvasása során nem módosít egy R nevű relációt, amelyből adatokat olvas. Az ilyen sormutatók feldolgozása az ugyanezen táblán végzett módosításokra érzéketlen sormutatókkal egyidejűleg is történhet, hiszen ekkor nem fordulhat elő, hogy azt az R relációt módosítják, amelyet az említett erre érzéketlen sormutató is felhasznál. Ha egy sormutató definícióját kiegészítjük a `FOR READ ONLY` kulcsszavakkal, úgy

az adatbázis-kezelő rendszer biztosan tudhatja róla, hogy végigolvasása során nem módosítja a benne felhasznált relációt.

9.9. példa. Ha a 9.8. ábrán látott `jövedelemSávok` lekérdezést kiegészítenénk a 8. sor utáni alábbi sorral:

```
FOR READ ONLY;
```

akkor ebben az esetben az `irányítókSormutató` nevű sormutatóval történő módosítási kísérletek hibát adnának. □

9.3.9. Dinamikus SQL

Az eddig megismert befogadó nyelvbe ágyazott SQL-modellünkben a speciális SQL-lekérdezések és -utasítások egy befogadó nyelvben írt programban szerepeltek. Egy másik beágyazott SQL-típus, amikor az utasítások kiszámítását a befogadó nyelvvel végeztetjük el. Az ilyen utasítások viszont a fordítási időben még nem ismertek, ezért ezeknek a feldolgozását nem bízhatjuk egy SQL-előfeldolgozó programra vagy a befogadó nyelv fordítóprogramjára.

Ilyen helyzet fordul elő például az SQL parancsértelmező programoknál: ezek bekérnek egy SQL-utasítást a felhasználótól, beolvassák az utasítást, majd végrehajtják azt. Ilyen például a 6. fejezetben vázolt, SQL-lekérdezések végrehajtására képes felület. Ha a lekérdezéseket futásidőben kérjük be és hajtjuk végre, akkor nem tehetünk semmit a fordítási időben. Így már a beolvasás után a lekérdezésnek illeszkednie kell az SQL-rendszer követelményeihez, és a lekérdezés végrehajtása is meg kell feleljen ugyanennek.

A befogadó nyelven megírt program utasíthatja az SQL adatbázis-kezelő rendszert arra, hogy a felhasználótól beolvasott SQL-utasítást elemezze, és állítsa elő egy olyan – belső ábrázolású – formáját, amelyet az adatbázis-kezelő rendszer végre tud hajtani, és végre is hajt. Ezeket a lépéseket két, ún. *dinamikus SQL-utasítás* segítségével végezhetjük el.

1. `EXEC SQL PREPARE` *V* `FROM` <kifejezés>, ahol *V* egy SQL-változó neve. A kifejezés lehet a befogadó nyelv valamely kifejezése, amely általában karakterlánc típusú. Ezt a karakterláncot tekintjük az SQL-utasításnak. Feltehető, hogy az adatbázis-kezelő rendszer elemzi az SQL-utasítást és futtatásra alkalmas formára is hozza, de a végrehajtására itt nem kerül sor, hanem az utasítás elemzett, végrehajtható formája a *V* változóban lesz eltárolva.

2. `EXEC SQL EXECUTE` *V*. Ennek az utasításnak a hatására a *V* által jelölt SQL-utasítás végrehajtásra kerül.

E fenti két lépést egy utasításban is elvégezhetjük, amely a következő:

`EXEC SQL EXECUTE IMMEDIATE` <kifejezés>

A két lépés összevonásának a hátulütői olyan utasításoknál látszanak igazán, amelyeket egyszer készítünk elő, majd több alkalommal is lefuttatjuk. Ugyanis az EXECUTE IMMEDIATE használatával az előkészítésnek a költségét minden egyes futtatásnál meg kell fizessük, nem pedig egyszer, mint a külön előkészített esetben.

```
1)  void kérdésBeolvasás() {

2)      EXEC SQL BEGIN DECLARE SECTION;
3)          char *kérdés;
4)      EXEC SQL END DECLARE SECTION;

5)      /* olvassuk be az SQL-utasítást, a kérdésnek
    lefoglalt memóriaterületre állítsuk rá (malloc
    segítségével) a kérdés osztott változóban tárolt
    mutatót */

6)      EXEC SQL PREPARE SQLquery FROM :kérdés;
7)      EXEC SQL EXECUTE SQLquery;
    }
```

9.10. ábra. Egy dinamikus SQL lekérdező utasítás előkészítése és végrehajtása

9.10. példa. A 9.10. ábrán vázolt C-program beolvas egy SQL-utasítást a szabványos bemenetről (alapértelmezés szerint a billentyűzetről) a `kérdés` változóba, előkészíti a végrehajtását, majd végrehajtja. Az `SQLquery` változó tartalmazza az utasítás előkészített formáját. Mivel a beolvasott SQL-lekérdezést csak egyszer hajtjuk végre, ezért a 9.10. ábra 6. és 7. sorai helyett írhatnánk egyszerűen az alábbi sort is:

```
EXEC SQL EXECUTE IMMEDIATE :kérdés;
```

□

9.3.10. Feladatok

9.3.1. feladat. Készítsük el a beágyazott SQL-lekérdezéseket az alábbi – 2.4.1. feladat leírásában megismert – adatbázissémához:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

A megoldás leírására bármilyen programozási nyelvet használhatunk. A befogadó nyelvi programozási részeket elegendő világos „megjegyzésekkel" helyettesíteni. A megvalósítandó lekérdezések a következők:

a) Kérjünk be a felhasználótól egy árértéket, és keressük meg a PC-táblában a megadott árhoz legközelebb eső áron kapható számítógép adatait. A megtalált számítógépről írassuk ki a gyártóját, modellszámát, sebességadatát.

b) Kérjük be a felhasználótól a számára elfogadható minimális sebességet, RAM-, winchester- és képernyőméretet, majd keressük ki az összes olyan laptop számítógépet, amely megfelel az igényeknek. A megtalált laptopokról írassuk ki a technikai adataikat (vagyis a `Laptop` reláció összes attribútumát) és gyártójuk nevét.

! *c)* Kérjük be a felhasználótól egy gyártó nevét, majd írjuk ki a képernyőre az illető gyártó összes termékét és az egyes termékek jellemzőit (tehát a modellszámot, típusazonosítót és a megfelelő típushoz tartozó összes technikai adatot).

!! *d)* Kérjünk be a felhasználótól egy összeget (egy PC és egy nyomtató vásárlására szánható maximális árat), valamint a megvásárolni kívánt PC-vel szemben támasztott minimális sebességkövetelményt. Keressük meg a legolcsóbb olyan PC-nyomtató kombinációt, amely megvásárolható a megadott összegből, és sebességét tekintve megfelel a felhasználó elvárásainak (vagyis nem lassabb a felhasználó által adott minimálisan elvárt sebességnél). Lehetőleg színes nyomtatót válasszunk. A program írja ki a kiválasztott PC-nyomtató kombináció modellszámait.

e) Kérjük be a felhasználótól egy új PC törzsadatait: gyártójának nevét, modellszámát, sebességét, RAM-, winchesterméretét, CD-sebességét és árát. Ellenőrizzük az adatbázisban, hogy van-e benne már ilyen PC. Ha az adatbázisban már vannak ilyen adatok, akkor írjunk ki a képernyőre egy figyelmeztető üzenetet; ha pedig nincs, akkor vegyük fel az adatbázisba.

9.3.2. feladat. Készítsünk beágyazott SQL-lekérdezéseket az alábbi adatbázissémához. Az adatbázissémát a 2.4.3. feladatból vettük.

```
Hajóosztályok(hajóosztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, hajóosztály, felavatva
Csaták(név, időpont)
Kimenetelek(hajó, csata, eredmény)
```

A megvalósítandó lekérdezések a következők:

a) Egy hajó tűzereje megközelítőleg arányos a rajta levő fegyverek számának és a fegyverkaliberek köbének a szorzatával. Keressük meg a legmagasabb tűzerejű hajóosztályt.

! *b)* Kérjük be a felhasználótól valamely csata nevét. Keressük meg a csatában részt vevő hajókhoz nyilvántartott országokat. Keressük meg annak

az országnak a nevét, amelynek a legtöbb hajója veszett oda az illető csatában, és azt az országot, amelynek a legtöbb hajója megsérült az illető csatában, és írassuk ki e két ország nevét.

c) Kérjük be a felhasználótól a `Hajóosztályok` tábla egy új sorának az adatait, majd kérjünk be további adatokat: az újonnan beadott hajóosztályba tartozó hajók neveit és felavatásuk időpontját. Vegyük fel ezeket az adatokat a fenti adatbázisba. Ne kelljen minden egyes hajónál beadni, hogy melyik osztályba tartozik, mivel mindegyik a feladat első lépéseként beadott osztályba tartozik.

! *d)* Vizsgáljuk meg a `Csaták`, `Kimenetelek`, `Hajók` táblákat. Keressünk olyan hajókat, amelyek részt vettek egy csatában, még mielőtt felavatták volna. Ha ilyen hajót találunk, akkor írjuk ki ezt a minden bizonnyal hibás adatot a felhasználónak, és kérdezzük meg, hogyan javítsuk az adatokat. Két javítási lehetőséget ajánljunk fel: vagy a csata időpontjának a módosítását, vagy pedig a hajó felavatásának az időpontját (ha szükséges, akkor a felhasználó végezhesse el mindkét módosítást).

9.4. Sémában tárolt eljárások

Ebben az alfejezetben betekintést kívánunk adni az olvasónak az SQL-szabvány ún. *Tartós, Tárolt Modulok* (Persistent, Stored Modules) részébe, amelyekre SQL/PSM vagy egyszerűen PSM néven szoktak hivatkozni. A PSM az SQL-szabvány legutóbbi módosításának (az SQL:2003-nak) a része. A PSM segítségével eljárásokat fogalmazhatunk meg egy egyszerű, általános célú nyelvben, illetve sémaelemként letárolhatjuk ezeket az adatbázisba. A tárolás után ezeket az eljárásokat használhatjuk SQL-lekérdezésekben, illetve egyéb olyan utasításokban, amelyekkel kizárólag az SQL használatával nem kiszámítható számításokat is elvégezhetjük. Minden forgalomban lévő rendszernek van saját PSM-kiterjesztése. Jelen könyvünkben az SQL/PSM-szabványt kívánjuk bemutatni, amely tartalmazza a PSM lehetőségeinek fő alapelveit. Emellett segít megérteni a konkrét rendszerhez tartozó nyelveket is. Néhány főbb kereskedelmi rendszer PSM-kiterjesztésére történő hivatkozás is szerepel az irodalomjegyzékben.

9.4.1. PSM-függvények és eljárások létrehozása

A PSM-ben létrehozhatunk ún. *modul*okat, amelyek tartalmazhatják függvények és eljárások definícióinak gyűjteményét, ideiglenes relációk deklarációit és számos választható deklarációs lehetőséget. Az eljárásdeklaráció lényeges elemei az alábbiak:

```
CREATE PROCEDURE <név> (<paraméterek>)
    <lokális deklarációk>
    <eljárás törzse>;
```

A fenti alak ismerősnek tűnhet jó néhány programozási nyelvből, tehát az eljárásdefiníció tartalmaz egy eljárásnevet, egy zárójelezett paraméterlistát, néhány opcionális lokális változódeklarációt és a törzs futtatható kódját. A függvények definíciója is hasonló módon történik azzal a két kivétellel, hogy `FUNCTION` kulcsszót használunk, illetve, hogy egy visszatérési típust is meg kell adnunk. Azaz egy függvénydefiníció elemei a következők:

```
CREATE FUNCTION <név> (<paraméterek>) RETURNS <típus>
   <lokális deklarációk>
   <függvény törzse>;
```

A PSM-eljárások paraméterei mód-név-típus hármasok lesznek, vagyis a paraméter nevét nemcsak a deklarált típusa követi, mint ahogy az a programozási nyelvekben megszokott, hanem a nevet megelőzi egy „mód", ami lehet `IN`, `OUT` vagy `INOUT` is. Ez a három kulcsszó – sorrendben – azt fejezi ki, hogy az adott paraméter csak bemenet, csak kimenet vagy egyben bemenet és kimenet is. Mivel az `IN` az alapértelmezett, ezért az `IN` el is hagyható.

Másrészről a függvényparaméterek csak `IN` módúak lehetnek, vagyis a PSM tiltja azt, hogy a függvénynek mellékhatásai legyenek, tehát csak a függvény visszatérési értékét használhatjuk a függvényre vonatkozó információként. Habár az eljárások paramétereinél megadtuk az `IN` módot, a függvények paramétereinél ezt nem szükséges megtennünk.

9.11. példa. Mivel egyenlőre még nem néztük meg azon utasítástípusokat, amelyek szerepelhetnek eljárások, illetve függvények törzsében, ezért az SQL-típusú utasítás használata remélhetőleg nem fog meglepni minket. Ezen utasítások korlátai ugyanazok, mint a beágyazott SQL esetében, amit a 9.3.4. alfejezetben láthattunk, azaz csak egysoros kiválasztás és a sormutató alapú hozzáférés engedélyezett lekérdezésként. A 9.11. ábrán szereplő PSM-eljárás két cím (egy új és egy régi cím) alapján az összes olyan színészhez tartozó `cím` attribútumot kicseréli az újra, amelynél a színész címe megegyezik `régiCím` értékével.

```
1)  CREATE PROCEDURE Költözés(
2)       IN régiCím VARCHAR(255),
3)       IN újCím VARCHAR(255)
    )
4)  UPDATE FilmSzínész
5)  SET cím = újCím
6)  WHERE cím = régiCím;
```

9.11. ábra. Címet megváltoztató eljárás

Az 1. sorban vezetjük be az eljárást és a nevét (`Költözés`). A 2. és 3. sorok két input paramétert tartalmaznak, amelyeknek a típusa maximum 255 hosszú, változó méretű karakterlánc (`VARCHAR(255)`) lesz. Figyeljük meg, hogy ez a típus megegyezik a `FilmSzínész` reláció `cím` attribútumának típusával, amelyet a

2.8. ábrán láthatunk. A 4. és 6. sorok között egy hagyományos UPDATE utasítást találhatunk. Azzal a megjegyzéssel, hogy a paraméternevek csak konstansként használhatóak. A befogadó nyelv változóival ellentétben, amelyek előtt egy kettőspont használata volt kötelező SQL-beli alkalmazásuk esetén (lásd 9.3.2. alfejezet), a paraméterek és a PSM-eljárások, illetve -függvények egyéb, lokális változóihoz nem kell kettőspontot használnunk. □

9.4.2. Néhány egyszerű utasítás alakja PSM-ben

Kezdjük egy olyan, utasításalakok egyvelegével, amelyek megértése egyszerű:

1. *A Call utasítás:* Az eljáráshívás alakja az alábbi:

   CALL <eljárás neve> (<argumentumlista>);

   Azaz a nyelvek túlnyomó többségéhez hasonlóan egy CALL kulcsszó, amelyet az eljárás neve és argumentumainak zárójelezett listája követ. Viszont ezen hívás több helyről is történhet. Pontosabban:

   i) Egy befogadó nyelvi programból, amelyben az alakja például lehet a következő:

   ```
   EXEC SQL CALL Foo(:x, 3);
   ```

   ii) Egy másik PSM-függvény vagy -eljárás utasításaként.

   iii) Egy általános SQL-felületen kiadott SQL-parancsként. Egy ilyen felületen kiadhatunk például egy olyan utasítást, mint a következő:

   ```
   CALL Foo(1, 3);
   ```

   Ennek hatására a Foo tárolt eljárás lefut az 1 és 3 értékű két paraméterével.

   Megjegyezzük azt, hogy a fentiek függvényhívásra nem megengedettek. PSM-ben a C-hez hasonló függvényhívás létezik, vagyis a függvényt és a megfelelő argumentumokat egy kifejezés részeként használhatjuk.

2. *Visszatérési utasítás:* Alakja a következő:

   RETURN <kifejezés>;

   Ez az utasítás csak függvényekben szerepelhet. Először kiértékeli a kifejezés értékét, majd ennek eredményét értékül adja a függvény visszatérési értékének. A hagyományos programozási nyelvekkel ellentétben viszont a PSM visszatérési értéke *nem* fejezi be a függvényt, hanem folytatja a vezérlést a következő utasítással, és így az is előfordulhat, hogy a visszatérési érték a függvény lefutása előtt megváltozik.

3. *Lokális változók deklarációja:* Az alábbi utasításforma egy adott nevű, adott típusú változót deklarál:

   DECLARE <név> <típus>;

   Ez a változó lokális, és így az értékét az ABKR a függvény vagy eljárás lefutását követően nem őrzi meg. A deklaráció meg kell előzze a függvény vagy az eljárás törzsében szereplő, futtatható utasításokat.

4. *Értékadó utasítások:* Az értékadás alakja a következő:

   SET <változó> = <kifejezés>;

   A bevezetőként szereplő SET kulcsszótól eltekintve egy PSM-beli értékadás nagyon hasonlít más nyelvek értékadásához. Az egyenlőség jobb oldalán lévő kifejezés kiértékelésre kerül és a bal oldalon szereplő változó értéke a kapott érték lesz. Hiányzó kifejezés esetén NULL értéket kap. A kifejezés akár egy SQL-lekérdezés is lehet, amennyiben egy értékkel tér vissza.

5. *Utasításcsoportok:* Pontosvesszővel végződő utasítások egy listáját fogalmazhatjuk meg egy BEGIN és egy END kulcsszavak között. Ez a szerkezet egy utasításként kezelendő, és így bárhol előfordulhat, ahol egy egyszerű utasítás is előfordulhat. Azaz, mivel egy eljárás, illetve függvény törzse egy egyszerű utasítás kell legyen, így ezzel utasítások sorozatát tehetjük be a törzsbe a BEGIN és az END kulcsszavak közé.

6. *Utasításcímkék:* Egy utasítás címkézéséhez az utasítás elé írunk egy nevet (a címkét), amit egy kettőspont követ.

9.4.3. Elágazásutasítások

Az első összetettebb PSM-utasítástípusunkhoz tekintsük az if utasítást, amelynek alakja kicsit furcsának tűnhet, mivel eltér a C-ben vagy más nyelvben megszokottaktól a következő értelemben:

1. Az utasítás az END IF kulcsszóval végződik.

2. Az else záradékba ágyazott if utasításokat az ELSEIF kulcsszóval vezethetjük be.

Azaz a 9.12. ábra egy if utasítás általános alakját mutatja be. A feltétel bármilyen olyan logikai értékű kifejezés lehet, amely SQL-utasítások WHERE záradékában is szerepelhet. Minden utasításlista pontosvesszővel végződő utasításokat tartalmazhat, és nem kell hozzá BEGIN...END környezet sem. Végezetül az ELSE utasítás és ennek utasítása(i) opcionálisak, ugyanis az IF...THEN...END IF önmagában és ELSEIF-ekkel együtt is helyes.

```
IF <feltétel> THEN
   <utasításlista>
ELSEIF <feltétel> THEN
   <utasításlista>
ELSEIF
      ...
ELSE
  <utasításlista>
END IF;
```

9.12. ábra. Az if utasítás egy alakja

9.12. példa. Írjunk függvényt, amely az *é* év és az *s* stúdió bementre vár, és pontosan akkor és csak akkor tér vissza igaz értékkel, ha az s stúdió legalább egy vígjátékot produkált vagy egyetlen filmet sem adott ki az *é* évben. A 9.13. ábrán szerepel az ehhez tartozó kód.

```
1)  CREATE FUNCTION Vígjáték(é INT, s CHAR(15))
        RETURNS BOOLEAN

2)  IF NOT EXISTS(
3)      SELECT * FROM Filmek WHERE év = é AND
            stúdióNév = s)
4)  THEN RETURN TRUE;
5)  ELSEIF 1 <=
6)      (SELECT COUNT(*) FROM Filmek WHERE év = é AND
            stúdióNév = s AND műfaj = 'vígjáték')
7)  THEN RETURN TRUE;
8)  ELSE RETURN FALSE;
9)  END IF;
```

9.13. ábra. Ha van legalább egy film, akkor legalább az egyiknek vígjátéknak kell lennie

Az 1. sor vezeti be a függvényt és annak argumentumait. Nem kell megadjuk az argumentumok módját, mivel egy függvénynek csak `IN` módú argumentumai lehetnek. A 2. és 3. sorok azt az esetet vizsgálják, hogy az *s* stúdiónak az *é* évben volt-e egyáltalán filmje. Amennyiben nem, a 4. sorban a visszatérési értéket `TRUE`-ra állítjuk. Megjegyezve azt, hogy a 4. sor nem eredményezi a függvény visszatérését. Technikailag az if utasítás vezérlésével folytatódik a végrehajtás, ami azt eredményezi, hogy a 4. sorból a 9. sorra ugrik a vezérlés, ahol a függvény befejeződik és visszatér.

Ha *s* stúdió készített filmet az *é* évben, akkor az 5. és 6. sor megvizsgálja, hogy ezek közül volt-e legalább egy vígjáték. Ha volt, akkor most a 7. sor ha-

tására itt is igaz értékre állítjuk a visszatérési értéket. A megmaradó esetben pedig, amikor az s stúdió csak olyan filmet adott ki, amely nem vígjáték műfajú volt, akkor a 8. sorban `FALSE`-ra állítjuk a visszatérési értéket. □

9.4.4. Lekérdezések PSM-ben

PSM-ben több módja is van a select-from-where típusú lekérdezések használatának:

1. Feltételekben vagy – általánosabban – minden olyan helyen, ahol SQL-ben is megengedett lenne, szerepelhetnek alkérdések. Például a 9.13. ábra 3., illetve 6. sorában már láthattunk erre két példát.

2. Az olyan lekérdezések, amelyek egyetlen értéket adnak vissza, használhatók értékadó utasítások jobb oldalán.

3. PSM-ben egy egysoros select utasítás is megengedett. Emlékezzünk vissza arra, hogy az ilyen utasításnak volt egy `INTO` záradéka, amely azt a változót adta meg, amelybe az egyetlen visszatérő sor került be. Ez a változó lehet egy lokális változó vagy egy PSM-eljárás paramétere is. Ennek az általános alakját a beágyazott SQL összefüggésében már a 9.3.5. alfejezetben megtárgyaltuk.

4. Deklarálhatunk és használhatunk sormutatókat is. Lényegében ezt ugyanúgy tehetjük meg, mint ahogyan azt a 9.3.6. alfejezetben megbeszéltek szerint a beágyazott SQL-re tettük. A sormutató deklarációja, `OPEN`, `FETCH` és `CLOSE` utasítások ugyanazok, mint korábban, a következő eltérésekkel:

   a) `EXEC SQL` nem szerepelhet az utasításokban, illetve

   b) a változóknál, lévén lokálisak, nem kell kettőspontot használni.

9.13. példa. A 9.14. ábrán szereplő egysoros kiválasztásnak PSM-be átírt változata a 9.7. ábrán látható egy képzelt eljárás definíciójában. Figyeljük meg, hogy ugyanezt a hatást érhetnénk el egy értékadás segítségével is, mivel az egysoros kiválasztásunknak csak egyetlen komponense van, azaz írhatnánk a következőt:

```
SET elnökNettóFizetése = (SELECT nettóBevétel
    FROM Stúdió, GyártásIrányító
    WHERE elnökAzon = azonosító AND Stúdió.név = stúdióNév);
```

A sormutatós példát a következő alfejezetben tárgyaljuk, amelyben megtanuljuk a PSM-ciklus használatát. □

```
CREATE PROCEDURE ValamilyenEljárás (IN stúdióNév CHAR(15))

DECLARE igazgatóNettóFizetése INTEGER;

SELECT nettóBevétel
INTO elnökNettóFizetése
FROM Stúdió, GyártásIrányító
WHERE elnökAzon = azonosító AND Stúdió.név = stúdióNév;
    ...
```

9.14. ábra. Egy egysoros kiválasztás (select) PSM-ben

9.4.5. Ciklusok a PSM-ben

Egy PSM-ben szereplő ciklus alapszerkezete az alábbi:

```
LOOP
   <utasításlista>
END LOOP;
```

Sok esetben hasznosnak bizonyulhat a `LOOP` utasítások címkézése, mivel lehetővé teszi, hogy kilépjünk az adott ciklusból a következő utasítás segítségével:

```
LEAVE <cikluscímke>;
```

A legtöbb esetben a ciklusokat egy sormutatóval történő sorkinyerésekhez (ún. fetcheléséhez) használjuk, továbbá ki is szeretnénk lépni a ciklusból, ha már nincs több sor. Hasznosnak bizonyulhat egy feltétel deklarálása az `SQLSTATE` értékéhez, ami azt fejezi majd ki, hogy nincs több sor (emlékeztetőül: `'02000'` érték). Ezt a következőképpen tehetjük meg:

```
DECLARE Nincs_Több CONDITION FOR SQLSTATE '02000';
```

Általános esetben tetszőleges nevű feltételt deklarálhatunk az alábbi módon, ami az `SQLSTATE` változó valamely értékéhez lesz rendelve:

```
DECLARE <név> CONDITION FOR SQLSTATE <érték>;
```

Ezek után már készen állunk arra, hogy a sormutató-műveleteket és a PSM-ciklusokat egyszerre mutassuk meg egy példán keresztül.

9.14. példa. A 9.15. ábrán láthatjuk azt a PSM-eljárást, amely az *s* stúdiónév argumentum alapján kiszámolja az adott stúdió által gyártott filmek hosszainak átlagát és szórását. (*Megjegyzés:* A négyzetes szórás, pontosabban tapasztalati szórásnégyzet kiszámítása következik.) Az 1–4. sorokban található az eljárás és a paramétereinek a deklarációja.

```
1)  CREATE PROCEDURE ÁtlagEltér(
2)      IN s CHAR(15),
3)      OUT átlag REAL,
4)      OUT szórás REAL
    )
5)  DECLARE Nincs_Több CONDITION FOR SQLSTATE '02000';
6)  DECLARE filmSormutató CURSOR FOR
        SELECT hossz FROM Filmek WHERE stúdióNév = s;
7)  DECLARE újHossz INTEGER;
8)  DECLARE filmSzámláló INTEGER;

    BEGIN
9)      SET átlag = 0.0;
10)     SET szórás = 0.0;
11)     SET filmSzámláló = 0;
12)     OPEN filmSormutató;
13)     filmCiklus: LOOP
14)         FETCH FROM filmSormutató INTO újHossz;
15)         IF Nincs_Több THEN LEAVE filmCiklus END IF;
16)         SET filmSzámláló = filmSzámláló + 1;
17)         SET átlag = átlag + újHossz;
18)         SET szórás = szórás + újHossz * újHossz;
19)     END LOOP;
20)     SET átlag = átlag/filmSzámláló;
21)     SET szórás = szórás/filmSzámláló - átlag * átlag;
22)     CLOSE filmSormutató;
    END;
```

9.15. ábra. A filmhosszak átlagának, illetve szórásának kiszámítása egy stúdióhoz

Az 5–8. sorokban szerepelnek a lokális deklarációk, ahol az 5. sorban definiáljuk a `Nincs_Több` nevű feltételt is annak kifejezésére, hogy a `FETCH` nem tudott újabb sort visszaadni. Ezek után a 6. sorban a `filmSormutató` nevű sormutatót úgy definiáljuk, hogy adja vissza az s stúdió filmjeinek hosszát tartalmazó halmazt. A 7. és 8. sorok két – a számításokhoz szükséges – változót vezetnek be. Az `újHossz` változó tartalmazza a `FETCH` utasítás eredményét, miközben a `filmSzámláló` változó pedig az s stúdió filmjeit számlálja. A végén majd szükségünk lesz a `filmSzámláló` változóra ahhoz, hogy az összesített filmhosszt átlagos hosszá alakítsuk, illetve, hogy a filmhosszak négyzeteinek összesítéséből megkapjuk a hosszak szórását.

A hátralévő sorok az eljárás törzsét határozzák meg. Az átlag, illetve szórás változókat lokális változókként használjuk azért, hogy a végén visszatérési értékek lehessenek. A főciklusunk során az átlag a hosszak összegét, a szórás pedig a hosszak négyzetösszegét tartalmazzák, azaz a 9–11. sorokban 0 kezdőértékkel

Más cikluskonstrukciók

A PSM megengedi a while és a repeat ciklusszerkezeteket, amelyeknek a jelentése hasonlít a C-beli társaikéhoz. Vagyis, egy ciklus alakja lehet a következő:

```
WHILE <feltétel> DO
   <utasításlista>
END WHILE;
```

vagy egy másik formája lehet az alábbi:

```
REPEAT
   <utasításlista>
UNTIL <feltétel>
END REPEAT;
```

Ha esetleg még címkéket is rendelünk ezekhez a ciklusokhoz, vagy a ciklus, vagy ha a ciklus loop vagy for utasítás alakú, akkor a címkét az `END LOOP` vagy más befejező kulcsszó után is elhelyezhetjük. Ez azért lehet előnyös, mivel világosabbá teheti, hogy hol érnek véget az egyes ciklusok, és emellett megengedik a PSM értelmezőnek, hogy megtaláljon néhány szintaktikai hibát (a hiányzó `END` kulcsszavat is beleértve).

inicializáljuk ezen két és a számláló változót is. A 12. sorban megnyitjuk a sormutatónkat, ami után a 13. és 19. sorok között futtatjuk a `filmCiklus` címkéjű ciklusunkat.

A 14. sor bekér egy sort, és a 15. sor pedig ellenőrzi, hogy volt-e egyáltalán bekérhető sor. Ha nincs, akkor kilépünk a ciklusból. A 16–18. sorok összegyűjtik a megfelelő értékeket. Vagyis a `filmSzámláló` értéke eggyel nő, a hossz hozzáadódik az `átlag` változóhoz (azaz valóban a hosszak összegét számlálja), a `szórás` változóhoz pedig hozzáadjuk a hossz négyzetét.

Az *s* stúdió összes filmjének végigvétele után befejezzük a ciklust, és a vezérlés a 20. sorra kerül át, ahol is a filmhosszak összegét a filmek számával elosztva megkapjuk az átlag valódi értékét. A 21. sorban a `szórás` változó értékét átalakítjuk a valódi szórásra azzal, hogy a hosszak négyzeteinek az összegét elosztjuk a filmek számával, és ebből levonjuk az átlag négyzetét. Ahhoz, hogy lássa az olvasó, hogy a számítás helyes, tekintse meg a 9.4.4. példát. A 22. sor lezárja a sormutatót, és ezzel végeztünk is. □

9.4.6. A for ciklus

Egy for ciklusszerkezet is elérhető a PSM-ben, de csak egy sormutatóban történő léptetésre alkalmazható. Az utasítás alakja a 9.16. ábrán látható. Ez az utasítás

nemcsak deklarál egy sormutatót, hanem jó néhány „kényelmetlen részletet" is kezel helyettünk: megnyitja és lezárja a sormutatót, bekéri a sorokat és ellenőrzi, hogy van-e egyáltalán bekérendő sor. Viszont mivel nem mi kérjük be a sorokat, így olyan változó(ka)t sem adhatunk meg, amely(ek)be egy sor komponense(i)t elhelyezhetnénk. Vagyis a PSM a lekérdezés eredményeinek attribútumainál meghatározott neveket használja megfelelő típusú lokális változókként.

```
FOR <ciklus neve> AS <sormutató neve> CURSOR FOR
   <lekérdezés>
DO
   <utasításlista>
END FOR;
```

9.16. ábra. A PSM for ciklusa

9.15. példa. Vegyük a 9.15. ábrán szereplő eljárás átírását for ciklussal. A kapott kód a 9.17. ábrán látható, ami sok tekintetben hasonlít az elődjéhez. Az eljárás 1–4. sorok közötti deklarációja megegyezik a 9.17. ábrán lévőkkel, mint ahogy a `filmSzámláló` nevű lokális változó 5. sorbeli deklarációja is ugyanolyan, mint korábban.

Itt viszont nincsen szükség sem az eljárás deklarációs területén szereplő sormutató deklarációjára, sem a `Nincs_Több` feltétel definíciójára. A 6–8. sorokban a korábbiakkal azonos kezdeti értékadás történik. Ezt követően a 9. sorban láthatjuk a for ciklust, amely az előzőhöz hasonlóan definiál egy `filmSormutató` nevű sormutatót. A 11–13. sorok közötti részben látható a ciklus törzse. Vegyük észre azt, hogy a 12–13. sorban a hosszt a sormutatón keresztül a `hossz` nevű attribútumból kapjuk meg, nem pedig egy `újHossz` nevű lokális változóból, amely a jelen eljárásban nem is szerepel. A 15–16. sorok – a korábbi eljárás verzióval szinkronban – kiszámítják a kimeneti változók helyes értékét. □

9.4.7. PSM-kivételek

Az SQL-rendszer a hibákat egy ötjegyű, nem csak nulla számjegyekből álló `SQLSTATE` nevű karakterlánc beállításával jelzi. Az ilyen kódokra már láthattunk példát: `'02000'` jelzi, hogy nem talált sort. Egy másik példaként említhető a `'21000'`, ami azt fejezi ki, hogy egy egysoros lekérdezés több mint egy sort eredményezett.

A PSM egy ún. *kivételkezelő* kódrészlet deklarálását engedi meg számunkra, amely minden esetben kiváltódik, ha ezen hibakódok egy listájának valamely tagja előfordul az `SQLSTATE`-ben az utasítás vagy utasítássorozat végrehajtása során. Minden egyes kivételkezelőhöz egy kódblokk tartozik, amelyet egy `BEGIN...END` foglal magába. A kezelő ezen blokkban szerepel, és csak a blokkban lévő utasításokat használja fel.

```
1)  CREATE PROCEDURE ÁtlagEltér(
2)      IN s CHAR(15),
3)      OUT átlag REAL,
4)      OUT szórás REAL
    )
5)  DECLARE filmSzámláló INTEGER;

    BEGIN
6)      SET átlag = 0.0;
7)      SET szórás = 0.0;
8)      SET filmSzámláló = 0;
9)      FOR filmCiklus AS filmSormutató CURSOR FOR
            SELECT hossz FROM Filmek WHERE stúdióNév = s;
10)     DO
11)         SET filmSzámláló = filmSzámláló + 1;
12)         SET átlag = átlag + length;
13)         SET szórás = szórás + hossz * hossz;
14)     END FOR;
15)     SET átlag = átlag/filmSzámláló;
16)     SET szórás = szórás/filmSzámláló - átlag * átlag;
    END;
```

9.17. ábra. A filmhosszak átlagának és szórásának kiszámítása for ciklus segítségével

A kezelő komponensei a következők:

1. Azon kivételfeltételek listája, amelyek kiváltódásuk esetén meghívják a kivételkezelőt.
2. Az a kód, amelyet lefuttat, amikor egy hozzá tartozó kivétel végrehajtódik.
3. Egy utalás arra nézve, hogy a kezelő lefutása után hol folytatódjon a végrehajtás.

A kezelő deklarációja az alábbi formát ölti:

DECLARE <hova menjen> HANDLER FOR <feltétellista>
<utasítás>

A „hova menjen" lehetőségek a következők lehetnek:

a) CONTINUE, amely azt fejezi ki, hogy a kezelő deklarációjában szereplő utasítás lefutását követően folytatjuk a hibát kiváltó utasítást követő utasítás végrehajtásával.

Miért szükséges neveket használni a for ciklusokban?

Vegyük észre, hogy a `filmCiklus`-t és a `filmSormutató`-t ugyan deklaráltuk a 9.17. ábra 9. sorában, ám az eljárás során sehol sem használtuk őket. Ennek ellenére viszont mind a for ciklushoz, mind az iterálható sormutatóhoz be kell vezetnünk őket. Ennek oka, hogy a PSM-értelmező minden for ciklust egy olyan hagyományos ciklusra fordít le, amely nagyon hasonlít a 9.15. ábrán szereplőhöz, és az ilyen kódoknál pedig mindkettőnek nevet kell adni.

b) `EXIT`, amely azt fejezi ki, hogy a kezelő utasításának lefutását követően, a vezérlés a kezelőhöz deklarált `BEGIN...END` blokknál marad. Majd az ezt a blokkot követő utasítás futtatásával folytatja a futást.

c) `UNDO`, amely ugyanolyan, mint az `EXIT`, azzal a különbséggel, hogy az adatbázisban vagy a lokális változókban a blokk végrehajtásának eredményeként történt változásokat „visszavonja". Vagyis, ezen utasításoknak nem lesz hatásuk, így tulajdonképpen ugyanolyanok lesznek, mintha végre sem hajtottuk volna őket.

A „feltétellista" a feltételeknek egy olyan vesszőkkel elválasztott listája, amelyben akár deklarált feltételek is szerepelhetnek, mint például a 9.15. ábra 5. sorában található `Nincs_Több`, vagy az `SQLSTATE`-ből és egy 5 karakteres karakterláncból álló kifejezések.

9.16. példa. Írjunk egy olyan PSM-függvényt, amely egy filmargumentumra visszaadja annak gyártási évét. Amennyiben nincs, vagy több azonos nevű film is van, akkor `NULL`-lal térjen vissza. Ennek kódja a 9.18. ábrán található.

```
1)  CREATE FUNCTION KiadásiÉv(fc VARCHAR(255))
        RETURNS INTEGER

2)  DECLARE Nincs_Találat CONDITION FOR SQLSTATE '02000';
3)  DECLARE Túl_Sok CONDITION FOR SQLSTATE '21000';

    BEGIN
4)      DECLARE EXIT HANDLER FOR Nincs_találat, Túl_Sok
5)          RETURN NULL;
6)      RETURN (SELECT év FROM Filmek WHERE filmcím = fc);
    END;
```

9.18. ábra. Kivételkezelés, ahol egy egysoros kiválasztás nem egy sorral tér vissza

A 2., illetve 3. sorok szimbolikus feltételeket deklarálnak. Nem feltétlen kellene használnunk ezeket a definíciókat, mivel helyettük az SQL-állapotokat is ugyanígy használhatnánk a 4. sorban. A 4., 5. és 6. sorok egy olyan blokkot alkotnak, amelyben először deklaráljuk a hibakezelőket arra a két esetre vonatkozóan, amikor üres sort, illetve egynél több sort kapunk vissza. Az 5. sorban lévő kezelő egyszerűen csak beállítja NULL-ra a visszatérési értéket.

A 6. sorban lévő utasítás pedig elvégzi a KiadásiÉv függvény feladatát. Ezt tulajdonképpen egy egész értékkel visszatérő SELECT utasításnak feltételezzük, mivel a KiadásiÉv függvény visszatérési értéke is ez. Ha pontosan egy *fc* című film van, ahol *fc* a függvény bemeneti paramétere, akkor a függvény ennek a kiadási évével tér vissza. Habár, ha a 6. sor végrehajtása kivételt vált ki – attól függően, hogy egy sor sincs vagy több sor is van –, akkor a kezelő meghívódik, és a visszatérési érték NULL-ra változik. Sőt, mivel a kezelő EXIT kezelő, ezért a vezérlés az END kulcsszó utánra adódik át, és mivel ez a függvény lefutását jelenti, ezért a KiadásiÉv befejeződik, és a visszatérési érték pedig NULL lesz. □

9.4.8. PSM-függvények és -eljárások használata

A 9.4.2. alfejezetben említettek alapján egy PSM-eljárást vagy -függvényt meghívhatunk egy beágyazott SQL-beli programból, magából a PSM-kódból vagy az általános felületen kiadott hagyományos SQL-utasításokból. Egy eljárást egy CALL előtaggal hívhatunk meg, a függvényeket pedig valamilyen kifejezésnek a részeként használhatjuk, például a WHERE záradékban. A most következőkben egy példát mutatunk arra, hogyan hívható meg egy függvény egy kifejezésen belül.

9.17. példa. Tegyük fel, hogy a 9.18. ábrán szereplő KiadásiÉv függvényt modulként tartalmazza az adatbázissémánk. Képzeljük azt, hogy egy általános felület előtt ülünk, és azt a tényadatot kívánjuk rögzíteni, hogy Denzel Washington az *Emlékezz a titánokra!* című film szereplője volt, ám a film elkészítésének évét elfelejtettük. Mivel csak egy ilyen című film van, ami a Filmek relációban meg is található, egy előzetesen kiadott lekérdezés megtekintése nélkül visszakaphatjuk ezt az értéket. Ezt elérhetjük az általános felületen kiadott alábbi beszúrás segítségével:

```
INSERT INTO SzerepelBenne(filmCím, filmÉv, színészNév)
VALUES('Emlékezz a titánokra!',
    KiadásiÉv('Emlékezz a titánokra!'),
        'Denzel Washington');
```

Mivel a KiadásiÉv függvény NULL értékkel tér vissza, ha az *Emlékezz a titánokra!* című film nem pontosan egyszer szerepel, így előfordulhat, hogy a beszúrás eredményének középső komponense NULL értékű lesz. □

9.4.9. Feladatok

9.4.1. feladat. Használjuk a korábbi filmadatbázisunkat, azaz:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió (név, cím, elnökAzon)
```

Írjunk PSM-függvényt vagy -eljárást a következő feladatokhoz:

a) Adott nevű filmstúdióhoz állítsuk elő a stúdió elnökének nettó bevételét.

b) Adott név és cím esetén adjuk vissza az 1 értéket, ha a kapott személy filmszínész és nem gyártásirányító, a 2 értéket, ha a kapott személy gyártásirányító és nem filmszínész, a 3 értéket, ha a kapott személy gyártásirányító és filmszínész is egyben, illetve a 4 értéket, ha egyik sem.

! *c)* Adott stúdiónév mellett jelölje a stúdió két leghosszabb filmje címének két kimeneti paraméterét. Rendeljünk NULL értéket az egyik vagy mindkét paraméterhez azokban az esetekben, amikor vagy az egyikre, vagy egyikre sincs megfelelő film (például, amikor csak egyetlen film tartozik az adott stúdióhoz, akkor nem lehet „második leghosszabb" film).

! *d)* Adott a filmszínész neve, keressük meg a legkorábbi (év szerint) olyan filmet, amelyben szerepelt, és amely 120 percnél hosszabb volt. Ha nincs a feltételnek megfelelő film, akkor a visszatérési év értéke legyen 0.

e) Adott lakcímhez keressük meg azt az egyértelműen meghatározható színész nevét, akinek ez a lakcíme. Ha nincs, vagy egynél több ilyen név van, akkor `NULL`-t adjunk vissza.

f) Adott a színész neve, töröljük ki a `FilmSzínész` táblából a hozzá tartozó bejegyzés(eke)t, és a `Filmek`, illetve `SzerepelBenne` táblákból is töröljük az összes ezzel kapcsolatos bejegyzést is.

9.4.2. feladat. Írjuk meg az alábbi PSM-függvényeket vagy -eljárásokat a 2.4.1. feladat adatbázissémája alapján:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez,
    cserélhető lemez, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

a) Egy árargumentum alapján adjuk vissza azon PC-modellszámokat, amelyeknek az ára nagyságrendileg akkora, mint az adott ár.

b) Egy gyártót, egy modellt és egy árat argumentumként használva adjuk meg a modellnek megfelelő összes terméktípusnak az árát.

! *c)* Vegyük úgy, hogy az argumentum tartalmazza a modell, a sebesség, a cserélhető lemez, illetve az ár információkat, és szúrjuk be ezeket az információkat a `PC` relációba. Ám, ha már volt egy PC, amelynek modellszáma ugyanaz (egy kulcs ütközési megszorítás feltételezése esetén a beszúrás ekkor kiváltja a `'23000'` értékű `SQLSTATE` hibát), akkor addig növeljük egyesével a modellszámot, amíg találunk egy olyan modellszámot, amely már nem PC-modellszám.

! *d)* Adott ár alapján állítsuk elő azon PC-knek, laptopoknak és nyomtatóknak a számát, amelyek az adott összegnél drágábbak.

9.4.3. feladat. Írjuk meg az alábbi PSM-függvényeket vagy -eljárásokat a 2.4.3. feladat adatbázissémája alapján:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

a) Egy hajó tűzereje megközelítőleg arányos a rajta levő fegyverek számának és a fegyverkaliberek köbének a szorzatával. Keressük meg a legmagasabb tűzerejű hajóosztályt.

! *b)* Egy adott csata neve alapján határozzuk meg azt a két országot, amelyeknek a hajói részt vettek a csatában. Amennyiben több mint két ország, vagy egy sem szerepelt a csatában, akkor mindkettőnek legyen az értéke `NULL`.

c) Legyenek az argumentumok a következők: egy új hajóosztálynév, egy típus, egy ország, egy ágyúszám, egy kaliberérték, egy vízkiszorítási érték. Adjuk hozzá ezeket az információkat a `Hajóosztályok` relációhoz, illetve adjunk hozzá egy ilyen hajóosztálynévvel rendelkező hajót a `Hajók` relációhoz is.

! *d)* Adott hajónévhez határozzuk meg, hogy részt vett-e a hajó olyan csatában, amelynek dátuma korábbi, mint a hajó felavatási dátuma. Ha igen, akkor mind a felavatás, mind a csata dátumát állítsuk nullára.

! **9.4.4. feladat.** A 9.15. ábrán egy trükkös formulát használtunk egy $x_1, x_2, \ldots, x_n$ számsorozat szórásának a kiszámításához. Megjegyezzük, hogy a számoknak az átlaguktól való eltérésük négyzetének átlaga adja a szórást. A

szórás eredeti képlete: $(\sum_{i=1}^{n}(x_i - \overline{x})^2)/n$, ahol $\overline{x}$ a $(\sum_{i=1}^{n} x_i)/n$ átlagot jelöli. Bizonyítsuk be, hogy a 9.15. ábráról származó

$$(\sum_{i=1}^{n}(x_i)^2)/n - ((\sum_{i=1}^{n} x_i)/n)^2$$

képlet is ugyanazt az értéket adja, mint a szórás képlete.

9.5. Hívásszintű interfészek használata

Hívásszintű interfész (Call Level Interface, CLI) használatakor szokványos befogadó nyelvű kódot írunk, és olyan függvénykönyvtárakat használunk, amelyek lehetővé teszik az adatbázishoz való kapcsolódást, annak elérését és SQL-utasítások átadását. A mostani megközelítés és a beágyazott SQL használata közti különbség – bizonyos értelemben – csupán kozmetikai, mivel az előfeldolgozó a beágyazott SQL-hívásokat az SQL/CLI-szabványban szereplő függvényekhez hasonló könyvtári függvényekkel helyettesíti.

Három példát adunk hívásszintű interfészre. Ebben a fejezetben a szabványos SQL/CLI-interfészt tekintjük át, ami az ODBC (Open Database Connectivity, nyílt adatbázis-összekapcsolhatóság) egy átdolgozása. A következő szakaszban áttekintjük a JDBC-t (Java Database Connectivity, Java adatbázis-összekapcsolhatóság), amely Java-programokat köt össze adatbázisokkal egy osztálygyűjtemény segítségével. Végezetül átnézzük a PHP-t, amely a HTML-ben írt weblapokból adatbázishoz történő hozzáféréshez ad egy keretmódszert.

9.5.1. Bevezetés az SQL/CLI-be

Egy C nyelven írt, SQL/CLI-t (a továbbiakban csak CLI-t) használó program beemeli az `sqlcli.h` fejállományt (header), melyből nagy számú függvényt, típusdefiníciót, struktúrát és szimbolikus konstanst nyer. Ezután a program négyfajta rekord (C-ben ezeket „struct"-nak hívjuk) létrehozására és kezelésére képes:

1. *Környezetek (environments).* Az ilyen típusú rekordot az alkalmazás (a kliens) hozza létre egy vagy több adatbázis-kapcsolat előkészítéséhez.

2. *Kapcsolatok (connections).* Ezen létrehozott rekordok egyike kapcsolja az alkalmazási programot az adatbázishoz. Minden egyes kapcsolat valamely környezeten belül létezik.

3. *Utasítások (statements).* Az alkalmazói program létrehozhat egy vagy több utasításrekordot. Mindegyik egy-egy SQL-utasításról hordoz információt, beleértve egy magában foglalt sormutatót, ha az utasítás egy lekérdezés. Különböző időpontokban ugyanaz a CLI-utasítás különböző SQL-utasításokat reprezentálhat. Minden egyes utasítás valamely kapcsolaton belül van értelmezve.

4. *Leírások (descriptions).* Ezek a rekordok sorokról vagy paraméterekről tartalmaznak információt. Az alkalmazói program vagy bizonyos esetekben az adatbázisszerver beállítja a leíró rekordok komponenseit, jelezve ezzel az attribútumok vagy azok értékeinek nevét vagy típusát. Minden utasítás rendelkezik implicit módon több ilyen leírással, a felhasználó pedig, ha szükséges, továbbiakat tud létrehozni. A CLI mostani bemutatásánál a leíró rekordok általában láthatatlanok lesznek.

Minden ilyen rekordot egy *nyél* (handle) reprezentál az alkalmazói programban, ami egy mutató a rekordra. Az `sqlcli.h` fejállomány megad típusokat a környezetek, kapcsolatok, utasítások és leírások nyeleihez: sorrendben `SQLHENV`, `SQLHDBC`, `SQLHSTMT` és `SQLHDESC`, jóllehet gondolhatunk rájuk, mint mutatókra vagy mint egészekre. Mi ezeket a típusokat és néhány másik, hasonlóan definiált típust a kézenfekvő értelmezésükkel fogjuk használni, mint az `SQL_CHAR` és az `SQL_INTEGER` típusokat, melyeket szintén az `sqlcli.h` szolgáltat.

Nem merülünk el annak részleteiben, hogyan kell leírásokat beállítani és használni. A másik három rekordtípushoz viszont nyeleket a következő függvény használatával tudunk létrehozni:

`SQLAllocHandle(`*hType, hIn, hOut*`)`

Az itt szereplő három argumentum pedig a következő:

1. *hType* a létrehozni kívánt nyél típusa. Használható az `SQL_HANDLE_ENV` egy új környezethez, az `SQL_HANDLE_DBC` új kapcsolathoz vagy az `SQL_HANDLE_STMT` új utasításhoz.

2. *hIn* a felsőbb szintű elem nyele, ahol az újonnan kiosztott (allokált) elem él. Ez a paraméter `SQL_NULL_HANDLE`, ha környezetet akarunk; ez a név konstansként definiált, és közli az `SQLAllocHandle` függvénnyel, hogy itt nincs ide vonatkozó érték. Ha kapcsolatnyelet szeretnénk, akkor *hIn* annak a környezetnek a nyele, amelyben az új kapcsolat fennáll, ha pedig egy utasításnyelet szeretnénk, *hIn* annak a kapcsolatnak a nyele, ahol az utasítás előfordul.

3. *hOut* annak a nyélnek a címe, amelyet az `SQLAllocHandle` függvénnyel hozunk létre.

Az `SQLAllocHandle` vissza is ad egy értéket `SQLRETURN` típussal (ami egy egész). Ez az érték 0, ha nem fordult elő hiba, és létezik több nem nulla visszatérési érték hibák esetére.

9.18. példa. Nézzük meg, hogy a 9.8. ábra `jövedelemSávok` függvénye, amit a beágyazott SQL példájaként használtunk, hogyan kezdődne CLI-ben. Emlékeztetőül, ez a függvény megvizsgálja a `GyártásIrányító` összes sorát, majd a nettó bevételeket intervallumokba tördeli. A kezdő lépéseket a 9.19. ábra szemlélteti.

```
1)  #include sqlcli.h
2)  SQLHENV környezetem;
3)  SQLHDBC kapcsolatom;
4)  SQLHSTMT execStat;
5)  SQLRETURN hibakód1, hibakód2, hibakód3;

6)  errorCode1 = SQLAllocHandle(SQL_HANDLE_ENV,
        SQL_NULL_HANDLE, &környezetem);
7)  if(!hibakód1) {
8)      errorCode2 = SQLAllocHandle(SQL_HANDLE_DBC,
            myEnv, &kapcsolatom);
9)  if(!hibakód2)
10)     errorCode3 = SQLAllocHandle(SQL_HANDLE_STMT,
            myCon, &execStat); }
```

9.19. ábra. Környezet, kapcsolat és utasítás deklarációja és létrehozása

A 2–4. sorok sorban deklarálják a környezet-, a kapcsolat- és az utasításnyeleket, sorrendben `környezetem`, `kapcsolatom` és `execStat` neveken. Azt tervezzük, hogy az `execStat` fogja reprezetnálni a következő SQL-utasítást:

```
SELECT nettóBevétel FROM GyártásIrányító;
```

ugyanúgy, mint ahogy a 9.8. ábra `irányítókSormutató` sormutatója tette azt, de egyelőre az `execStat` változóhoz nincs SQL-utasítás rendelve. Az 5. sor három változót deklarál, amelyekbe a függvényhívások tehetik a válaszukat és jelezhetnek hibát. A 0 érték jelzi, ha nem lépett fel hiba a hívásnál, mi pedig erre az esetre számítunk a továbbiakban.

A 6. sor meghívja az `SQLAllocHandle` függvényt, kérve egy környezetnyelet (első paraméter), átadva egy nullérték nyelet második paraméterként (mivel ez nem szükséges, ha környezeti nyelet igénylünk), valamint átadva a `környezetem` címét, mint harmadik paramétert, itt lesz elhelyezve a létrehozott nyél. Ha a 6. sor hiba nélkül lefut, a 7–8. sorok használják a környezeti nyelet egy kapcsolati nyél létrehozásához a `kapcsolatom`-ba. Feltéve hogy a hívás is sikeres volt, a 9–10. sorok egy utasításnyelet hoznak létre az `execStat`-ba. □

9.5.2. Utasítások feldolgozása

A 9.19. ábra végén létrehoztunk egy utasításrekordot, amelynek a nyele `execStat`-ba kerül, bár mindeddig nincs SQL-utasítás, amivel a rekord össze lenne kapcsolva. Az SQL-utasítások társításának és futtatásának folyamata utasításnyelekkel hasonló a 9.3.9. alfejezetben leírt dinamikus SQL-hez. Akkor az SQL-utasítás szövegét hozzárendeltük egy „SQL-változó"-hoz `PREPARE` használatával, majd `EXECUTE` segítségével futtattuk.

CLI esetén a helyzet ehhez hasonló, ha az „SQL-változó"-ra utasításnyélként gondolunk. Rendelkezésünkre áll egy

`SQLPrepare(`*sh, st, sl*`)`

függvény, melynek paraméterei:

1. egy *sh* nevű utasításnyél,
2. egy mutató az *st* SQL-utasításra, valamint
3. egy *sl* hosszúság az *st* által mutatott karakterlánchoz. Ha nem tudjuk a hosszúságot, az előre definiált `SQL_NTS` konstans közli az `SQLPrepare`-rel, hogy a hossz magából a karakterláncból számolandó. Többnyire valószínűleg a karakterlánc egy „nullértékkel lezárt karakterlánc", és az `SQLPrepare`-nek elég addig átvizsgálnia, amíg rá nem akad egy sorvégét jelölő `'\0'`-ra.

A függvény lefutása azt eredményezi, hogy az *sh* nyéllel reprezentált utasítás a továbbiakban már a konkrét *st* SQL-utasítást reprezentálja. Egy másik, az

`SQLExecute(`*sh*`)`

függvény felelős az *sh* nyél által hivatkozott utasítás végrehajtásáért. Az SQL-utasítások sok változatára, mint amilyen a beszúrás és a törlés, az utasítás futtatásának adatbázisra gyakorolt hatása kézenfekvő. Kevésbé nyilvánvaló, hogy mi történik akkor, ha az *sh* által hivatkozott utasítás egy lekérdezés. Mint ahogy a 9.5.3. alfejezetben látni fogjuk, egy implicit sormutató tartozik az utasításhoz magának az utasítás rekordjának részeként. Az utasítást elvileg végrehajtottuk, gondolhatjuk tehát, hogy a válaszsorok már ott ülnek valahol, készen arra, hogy elérjük őket. Le is hívhatjuk a sorokat egyesével az implicit sormutató használatával, hasonló módon, ahogy a 9.3. és 9.4. alfejezetekben valódi sormutatókkal tettük.

9.19. példa. Foglalkozzunk tovább a 9.19. ábrán elkezdett `jövedelemSávok` függvénnyel. A következő két függvényhívás kapcsolódik az alábbi lekérdezéshez:

```
SELECT nettóBevétel FROM GyártásIrányító;
```

az `execStat` nyél által hivatkozott utasítással:

```
11)  SQLPrepare(execStat, "SELECT nettóBevétel
         FROM GyártásIrányító", SQL_NTS);
12)  SQLExecute(execStat);
```

A fenti függvényhívások megjelenhetnek közvetlenül a 9.19. ábra 10. sora után. Emlékezzünk rá, hogy az `SQL_NTS` közli az `SQLPrepare`-rel, hogy a második paraméter által hivatkozott nullértékkel lezárt karakterlánc hossza kiszámítandó. □

Mint ahogy a dinamikus SQL-nél, az előkészítési és végrehajtási lépések egyesíthetők, mégpedig az SQLExecDirect függvény használatával. Egy példa, ami összefogja a 11–12. sorokat:

```
SQLExecDirect(execStat, "SELECT nettóBevétel
    FROM GyártásIrányító", SQL_NTS);
```

9.5.3. Adatok lehívása lekérdezés eredményéből

A függvény, amely megfelel a FETCH parancsnak beágyazott SQL vagy PSM esetén, a következő:

SQLFetch(*sh*)

ahol *sh* egy utasításnyél. Feltételezzük, hogy az *sh* által hivatkozott utasítás már végre lett hajtva, különben a lehívás hibát idéz elő. Az SQLFetch, mint minden CLI-függvény, egy SQLRETURN típusú értékkel tér vissza, ami jelzi a sikeres futást vagy a hibát. Az SQL_NO_DATA visszatérési érték jelzi, hogy nem maradt több sor a lekérdezés eredményében. Ezt az értéket, mint ahogy az előző lehívásos példákban is tettük, arra használjuk, hogy kilépjünk abból a ciklusból, amiben egyre újabb és újabb sorokat olvasunk ki az eredményből.

Hol jelenik meg egyébként a sor, ha a 9.19. példa SQLExecute utasítása után egy vagy több SQLFetch hívást teszünk? A válasz, hogy ennek komponensei bekerülnek azon leíró rekordok egyikébe, amelyek az SQLFetch hívásban szerepelő nyelű utasítással vannak összetársítva. Minden egyes lehívásnál kinyerhetjük ugyanazt a komponenst, ha a lehívások elkezdése előtt hozzákötjük a komponenst egy befogadó nyelvi változóhoz. A függvény, amely megvalósítja ezt a feladatot, a következő:

SQLBindCol(*sh, colNo, colType, pVar, varSize, varInfo*)

A fenti hat paraméter jelentése:

1. *sh* a szóban forgó utasítás nyele.
2. *colNo* azon komponensek száma (a soron belül), melyeknek értékét meg fogjuk kapni.
3. *colType* egy kód azon változó típusához, amelybe a komponens értéke kell hogy kerüljön. Példák ilyen kódokra az sqlcli.h által kínált SQL_CHAR karaktertömbök és karakterláncok számára, SQL_INTEGER egészekhez.
4. *pVar* egy mutató arra a változóra, amelybe az érték kell hogy kerüljön.
5. *varSize* a *pVar* által mutatott változó értékének bájtokban mért hossza.
6. *varInfo* egy mutató egy egészre, amelyet az SQLBindCol használhat az előállított érték járulékos leírására.

9.20. példa. Alakítsuk át a 9.8. ábra teljes `jövedelemSávok` függvényét beágyazott SQL helyett CLI-hívások használatával. A 9.19. ábrának megfelelően kezdünk, de a tömörség kedvéért elhagyunk minden hibaellenőrzést, kivéve azt az ellenőrzést, ahol az `SQLFetch` jelzi, ha már nincs jelen több sor. A kódot a 9.20. ábra mutatja.

```
1)  #include sqlcli.h
2)  void jövedelemSávok() {

3)      int i, digits, counts[15];
4)      SQLHENV környezetem;
5)      SQLHDBC kapcsolatom;
6)      SQLHSTMT execStat;
7)      SQLINTEGER bevétel, jövedelemInfo;

8)      SQLAllocHandle(SQL_HANDLE_ENV,
            SQL_NULL_HANDLE, &környezetem);
9)      SQLAllocHandle(SQL_HANDLE_DBC, környezetem,
            &kapcsolatom);
10)     SQLAllocHandle(SQL_HANDLE_STMT, kapcsolatom,
            &execStat);
11)     SQLPrepare(execStat,
            "SELECT nettóBevétel FROM GyártásIrányító",
                SQL_NTS);
12)     SQLExecute(execStat);
13)     SQLBindCol(execStat, 1, SQL_INTEGER, &bevétel,
            sizeof(bevétel), &jövedelemInfo);
14)     while(SQLFetch(execStat) != SQL_NO_DATA) {
15)         digits = 1;
16)         while((bevétel /= 10) > 0) digits++;
17)         if(digits <= 14) counts[digits]++;
        }
18)     for(i=0; i<15; i++)
19)         printf("digits = %d: number of execs = %d\n",
                i, counts[i]);
    }
```

9.20. ábra. Nettóbevételek csoportosítása: CLI-változat

A 3. sor deklarálja ugyanazokat a lokális változókat, amelyeket a függvény beágyazott SQL-változata is használ, a 4–7. sorok pedig ehhez újabb, az `sqlcli.h` által kínált típusú lokális változókat deklarálnak. Bizonyos értelemben ezek a változók hozzák be az SQL-t. A 4–6. sorok azonosak a 9.19. ábra megfelelő soraival. Új elemek a 7. sor `bevétel`-re (amely megfelel a 9.8. ábra

Komponensek kinyerése SQLGetData segítségével

A változók eredményrelációhoz való kötése mellett van egy másik lehetőségünk is, nevezetesen, hogy behozzuk a sorokat kötés nélkül, majd a komponenseket változókhoz rendeljük. Ehhez használhatjuk az SQLGetData nevű függvényt, amelynek ugyanazok a paraméterei, mint az SQLBindCol-nak. Ez azonban csak egyszer másolja az adatokat, és így minden beolvasás után meg kell hívnunk, hogy ugyanazt elérjük, mint az oszlopok változóhoz kötésével.

ugyanilyen nevű osztott változójának) és jövedelemInfo-ra vonatkozó deklarációi, amelyek az SQLBindCol miatt szükségesek, azonban használaton kívüliek.

A 8–10. sorok kiosztják a szükséges nyeleket a 9.19. ábrához hasonlóan, a 11. és 12. sorok pedig előkészítik és végrehajtják az SQL-utasítást a 9.19. példában tárgyaltaknak megfelelően. A 13. sorban láthatjuk a lekérdezésnek megfelelő eredmény első (és egyetlen) oszlopának kötését a bevétel változóhoz. Az első paraméter a kapcsolódó utasítás nyele, a második pedig a szóban forgó oszlop, ebben az esetben első. A harmadik paraméter az oszlop típusa, a negyedik pedig egy mutató arra a helyre, ahová az érték kerül, azaz a bevétel változó. Az ötödik paraméter a változó mérete, az utolsó paraméter pedig a jövedelemInfo-ra mutat, egy olyan helyre, ahová az SQLBindCol tesz járulékos információkat (melyeket most nem használunk).

A függvény hátralevő része nagyban hasonlít a 9.8. ábra 11–19. soraihoz. A while ciklus a 14. sorban kezdődik a 9.20. ábrán. Figyeljük meg, ahogyan lehívunk egy sort, majd ellenőrizzük, hogy nem fogytunk-e ki a sorokból, mindezt a 14. sor while ciklusának feltételében. Ha van még eredménysor, akkor a 15–17. sorig meghatározzuk, hány számjegyből áll az egész értékünk (ami a bevétel változóhoz van kötve), majd növeljük a megfelelő számlálót. Miután a ciklus befejeződött, vagyis a 12. sorban szereplő utasítás végrehajtásából kapott összes sor ellenőrzésre került, a 18–19. sorokban kiírjuk az eredményként kapott számlálókat. □

9.5.4. Paraméterek átadása lekérdezéseknek

A beágyazott SQL lehetőséget biztosít arra, hogy olyan SQL-lekérdezést hajtsunk végre, amelynek részei osztott változók aktuális tartalma által meghatározott értékekből állnak. A CLI is rendelkezik hasonló képességekkel, azonban jóval összetettebb módon. A szükséges lépések a következők:

1. Használjuk az SQLPrepare függvényt olyan SQL-utasítás előkészítésére, amelyben néhány, paraméternek nevezett részt kérdőjelekkel helyettesítünk. Az i-edik kérdőjel az i-edik paramétert jelképezi.

2. Használjuk az `SQLBindParameter` függvényt értékek kötéséhez olyan helyekre, ahol kérdőjelek találhatók. Ennek a függvénynek tíz argumentuma van, de ezek közül mi csak az alapvető fontosságúakat mutatjuk be.

3. Futtassuk a lekérdezést ezekkel a kötésekkel az `SQLExecute` hívásával. Figyeljünk, hogy ha megváltoztatjuk egy vagy több változó értékét, akkor újra meg kell hívnunk az `SQLExecute` függvényt.

A következő példa szemlélteti a fenti eljárást, valamint bemutatja az `SQLBindParameter` által igényelt fontosabb változókat.

9.21. példa. Nézzük meg újra a 9.6. ábra beágyazott SQL-kódját, ahol két, `stúdióNév` és `stúdióCím` változóba kaptuk az értékeket, amelyeket aztán egy sor komponenseiként használtunk, majd beszúrtuk őket a `Stúdió` táblába. A 9.21. ábra vázolja, hogyan működik ez a folyamat CLI használatával. Feltételezzük, hogy van egy `myStat` utasításnyelünk, amelyet a beszúrás utasításoknál használunk.

```
        /* értékek lekérése stúdióNév és stúdióCím számára */

1)  SQLPrepare(myStat,
        "INSERT INTO Stúdió(név, cím) VALUES(?, ?)",
        SQL_NTS);
2)  SQLBindParameter(myStat, 1,..., stúdióNév,...);
3)  SQLBindParameter(myStat, 2,..., stúdióCím,...);
4)  SQLExecute(myStat);
```

9.21. ábra. Új stúdió beszúrása paraméterek értékekhez kötésével

A kód (az itt nem szereplő) értékadó lépésekkel kezdődik, amivel a `stúdióNév` és `stúdióCím` változóknak adunk értékeket. Az 1. sor mutatja, ahogy a `myStat` utasítást előkészítjük egy beszúrási utasítássá, amely két paraméterrel (a kérdőjelek) rendelkezik a `VALUE` záradékban. Ezután a 2. és 3. sorok kötik sorrendben az első és második kérdőjeleket a `stúdióNév` és `stúdióCím` aktuális értékéhez. Végül a 4. sor hajtja végre a beszúrást. Ha a 9.21. ábra összes lépésének sorozatát – beleértve az itt nem látható új értékeket, `stúdióNév`-be és `stúdióCím`-be kinyerő munkát is – egy ciklusba helyezzük, akkor minden egyes iterációnál beszúródik a `Stúdió` táblába egy új sor, új névvel és címmel a stúdió számára. □

9.5.5. Feladatok

9.5.1. feladat. Oldjuk meg ismét a 9.3.1. feladat problémáit, de ezúttal C-ben, CLI-hívásokkal írjunk kódot.

9.5.2. feladat. Oldjuk meg ismét a 9.3.2. feladat problémáit, de ezúttal C-ben, CLI-hívásokkal írjunk kódot.

9.6. Java adatbázis-összekapcsolhatóság (JDBC)

A JDBC, a Java Database Connectivity *(Java adatbázis-összekapcsolhatóság)* rövidítése, a CLI-hez hasonló eszközkészlet, amely lehetővé teszi Java-programok számára SQL-adatbázisok elérését. A fogalmak nagyban hasonlóak a CLI fogalmaihoz, jóllehet JDBC esetén a Java objektumorientált természete magától értetődően adódik.

9.6.1. Bevezetés a JDBC-be

Az első lépések a JDBC használatához a következők:

1. Írjuk be a következő sort, hogy a JDBC-osztályokat el tudjuk érni:

   ```
   import java.sql.*;
   ```

2. Betöltünk egy „drivert", egy meghajtóprogramot a használandó adatbázisrendszerhez. Ez a „driver" ugyan az általunk használt ABKR típusától függ, de betölthető a következő utasítás segítségével:

   ```
   Class.forName(<driver neve>);
   ```

   Például ahhoz, hogy egy MySQL-adatbázishoz kapjunk meghajtót, hajtsuk végre a

   ```
   Class.forName("com.mysql.jdbc.Driver");
   ```

   parancsot. Ennek hatására létrejön egy DriverManager nevű osztály. Ez az osztály sok szempontból hasonló ahhoz a környezethez, amelynek a nyelét első lépésben kapjuk meg CLI használata esetén.

3. Létrehozunk egy kapcsolatot az adatbázissal. Ha alkalmazzuk a DriverManagerre a `getConnection` metódust, létrejön a Connection osztálynak egy változója.

A kapcsolat létrehozását szolgáló Java-utasítás a következőképp néz ki:

```
Connection kapcsolatom = DriverManager.getConnection(<URL>,
                         <név>, <jelszó>);
```

Tehát a `getConnection` metódus megkap paraméterként egy URL-t ahhoz az adatbázishoz, amihez kapcsolódni szeretnénk, valamint a felhasználói nevünket és a jelszónkat. Eredményül egy Connection osztályba tartozó objektummal tér vissza, amelynek a `kapcsolatom` nevet választottuk.

9.22. példa. Minden egyes ABKR-nek megvan a saját módszere a `getConnection` metódus URL-jének specifikálására. Egy MySQL adatbázis esetén például az URL a következő alakot öltheti:

`jdbc:mysql://`<kiszolgáló neve>`/`<adatbázis neve>

□

A JDBC Connection osztály objektuma nagyban hasonló a CLI kapcsolatához, és ugyanazokat a célokat szolgálja. A `kapcsolatom` jellegű Connection-objektumok megfelelő metódusainak használatával létrehozhatunk utasításobjektumokat, elhelyezhetünk SQL-utasításokat ezekben az objektumokban, köthetünk értékeket SQL-utasítás paramétereihez, végrehajthatunk SQL-utasításokat és soronként vizsgálhatjuk az eredményeket.

9.6.2. Utasítások létrehozása JDBC-ben

Utasítások létrehozásához a kapcsolat típusú objektumok két metódusát alkalmazhatjuk:

1. `createStatement()` visszaad egy Statement típusú objektumot. Ez az objektum még nem rendelkezik hozzárendelt SQL-utasítással, ezért a `createStatement()` metódus felfogható a CLI `SQLAllocHandle` hívásának analógiájaként, amely egy kapcsolati nyelet kap, és egy utasításnyéllel tér vissza.

2. `prepareStatement(`Q`)`, ahol Q egy SQL-lekérdezés, amelyet karakterlánc-paraméterként adunk át, a visszatérési érték pedig egy PreparedStatement típusú objektum. Ennélfogva felvázolhatunk egy párhuzamot a JDBC `createStatement(`Q`)`-jának végrehajtása és aközött a két CLI-utasítás között, amelyben az `SQLAllocHandle` segítségével kapunk egy utasításnyelet, amelyre aztán egy Q lekérdezéssel egyetemben alkalmazzuk az `SQLPrepare` eljárást.

Négy különböző metódus létezik SQL-utasítások végrehajtásához. A fenti metódusokhoz hasonlóan ezek is különböznek abban, hogy kapnak-e paraméterként SQL-utasítást vagy sem. Mindazonáltal ezek a metódusok különbséget is tesznek SQL-utasítások között aszerint, hogy lekérdezésekről, vagy pedig más utasításokról van szó, amelyeket gyűjtőnéven frissítéseknek (`UPDATE`) hívnak. Megjegyezzük azonban, hogy az `SQL UPDATE` utasítása csak kicsiny szelete a JDBC „frissítés" fogalmának. Az utóbbiak közé tartozik minden módosító utasítás, mint amilyen a beszúrás (insert), és minden sémához köthető utasítás, mint a `CREATE TABLE`. A négy „végrehajtó" (execute) metódus a következő:

a) `executeQuery(`Q`)` kap egy Q utasítást, amelynek lekérdezésnek kell lennie, amit aztán alkalmazunk a Statement-objektumra. Ez a metódus egy ResultSet típusú objektummal tér vissza, amely Q lekérdezés által létrehozott sorok halmaza (multihalmaza, hogy precízebbek legyünk). A sorokhoz történő hozzáférések miként jét a 9.6.3. alfejezetben tárgyaljuk.

b) Az `executeQuery()` utasítást egy PreparedStatement-objektumon hajtjuk végre. Mivel az előkészített utasításhoz már eleve tartozik hozzárendelt lekérdezés, ezért nincs paramétere. Ez a metódus is egy ResultSet-objektummal tér vissza.

c) `executeUpdate(`U`)` kap egy nem lekérdezés jellegű U utasítást, és amikor egy Statement-objektumra alkalmazzuk, végrehajtja U-t. A hatás csak az adatbázison érezhető, ResultSet-objektumot nem kapunk vissza.

d) Az `executeUpdate()` paraméterek nélkül alkalmazható egy PreparedStatement-objektumra. Ebben az esetben lefut az előkészített utasításhoz tartozó SQL-parancs. Természetesen ez az SQL-parancs nem lehet lekérdezés.

9.23. példa. Tegyük fel, hogy van egy `kapcsolatom` Connection-objektumunk, és végre szeretnénk hajtani a következő utasítást:

```
SELECT nettóBevétel FROM Gyártásirányító;
```

Az egyik lehetőségünk, hogy létrehozunk egy `execStat` nevű Statement-objektumot, majd ezt közvetlenül használjuk a lekérdezés végrehajtásához.

```
Statement execStat = kapcsolatom.createStatement();
ResultSet Bevételek = execStat.executeQuery(
    "SELECT nettóBevétel FROM Gyártásirányító");
```

Az eredmény egy `bevételek` nevű ResultSet (eredményhalmaz) osztályba tartozó objektumba kerül. A 9.6.3. szakaszban majd látni fogjuk, hogyan kell ebből a nettó bevételt kinyerni és feldolgozni.

Egy másik lehetőség, hogy a lekérdezést először előkészítjük, majd később végrehajtjuk. Ez a megoldás akkor előnyös, ha többször szeretnénk ugyanazt a lekérdezést végrehajtani. Ekkor van annak értelme, hogy egyszer előkészítjük, majd többször végrehajtjuk ahelyett, hogy ugyanazt a lekérdezést többször kellene előkészítenie az adatbázis-kezelőnek. E megoldás JDBC-lépései a következők:

```
PreparedStatement execStat = kapcsolatom.prepareStatement(
    "SELECT nettóBevétel FROM GyártásIrányító");
ResultSet bevételek = execStat.executeQuery();
```

A lekérdezés eredménye ismét egy `bevételek` nevű ResultSet osztályba tartozó objektum lesz. □

9.24. példa. Ha egy paraméter nélküli utasítást hajtunk végre, ami nem lekérdezés, akkor hasonló lépésekeket kell tennünk mindkét megoldási forma esetén. Ez esetben azonban nincs eredményhalmaz. Tegyük fel például, hogy a `SzerepelBenne` relációba szeretnénk beszúrni azt, hogy Denzel Washington szerepelt az *Emlékezz a titánokra!* című filmben 2000-ben. Ehhez létrehozhatunk egy `színészStat` nevű utasítást a következő két módszer bármelyikével:

```
Statement színészStat = kapcsolatom.createStatement();
színészStat.executeUpdate("INSERT INTO SzerepelBenne VALUES(
  " + "'Emlékezz a titánokra!', 2000, 'Denzel Washington')");
```

vagy a másik módszer:

```
PreparedStatement színészStat = kapcsolatom.prepareStatement(
    "INSERT INTO SzerepelBenne VALUES(
        'Emlékezz a titánokra!', " + "2000,
        'Denzel Washington')");
színészStat.executeUpdate();
```

Vegyük észre, hogy mindkét fenti utasítássorozatban használtuk a Java nyelv konkatenáció operátorát, a „+" jelet. Ennek segítségével tudtunk egy SQL-utasítást több Java-sorban elhelyezni. □

9.6.3. Kurzorműveletek JDBC-ben

Amikor egy lekérdezést hajtunk végre, és eredményül kapunk egy ResultSet-objektumot, annak sorain végigfuttathatunk egy kurzort. Ennek támogatására a ResultSet osztály a következő metódusokat biztosítja:

1. `next()`: egy ResultSet-objektumra alkalmazva azt eredményezi, hogy az implicit kurzor a következő sorra lép (amikor először hívjuk, akkor az első sorra). A metódus `HAMIS` értékkel tér vissza, ha nincs következő sor.

2. `getString(`i`)`, `getInt(`i`)`, `getFloat(`i`)` és más további hasonló metódusok, amelyek az SQL-beli típusoknak felelnek meg. Ezek mindegyike a kurzor aktuális sorának i-edik elemével tér vissza. Azt a metódust kell használnunk, amelyik az i-edik elem típusának megfelel.

9.25. példa. Miután megkaptuk a `bevételek` nevű eredményhalmazt, ahogy azt a 9.23. példában láttuk, egyesével férhetünk hozzá a soraihoz. Emlékezzünk rá, hogy ezeknek a soroknak most csak egyetlen, egész típusú eleme van. A ciklus formája a következő lesz:

```
while(bevételek.next()) {
    int bevétel = bevételek.getInt(1);
    /* feldolgozzuk a nettó bevételt */
};
```

□

9.6.4. Paraméterátadás

A CLI-ben megismert módszerhez hasonlóan, itt is a kérdőjelet használhatjuk a lekérdezés egyes részeinek helyén, majd értékeket rendelünk ezekhez a

*paraméterek*Shez. A JDBC-ben ezt úgy tehetjük meg, hogy előkészített utasításokat hozunk létre, majd ezekre olyan metódusokat alkalmazunk, mint a `setString(`i, v`)` vagy a `setInt(`i, v`)`. Ezek a metódusok a v értéket rendelik a lekérdezés i-edik paraméteréhez.

9.26. példa. Utánozzuk le a 9.21. példában szereplő CLI-kódot, amelyben előkészítettünk egy utasítást egy új sornak a `Stúdió` relációba való beszúrására. Paraméterként a stúdió nevét és a címét adtuk meg ott. A Java-kódot, amely előkészíti, beállítja a paramétereket, majd végrehajtja az utasítást, a 9.22. ábrán láthatjuk. Most is feltesszük, hogy van egy `kapcsolatom` nevű Connection-objektumunk.

```
1)  PreparedStatement studioStat =
        kapcsolatom.prepareStatement(
2)          "INSERT INTO Stúdió(név, cím) VALUES(?, ?)");
    /* bekérjük a nevet és a címet a felhasználótól
       a stúdióNév és stúdióCím változókba */
3)  studioStat.setString(1, stúdióNév);
4)  studioStat.setString(2, stúdióCím);
5)  studioStat.executeUpdate();
```

9.22. ábra. Paraméterek beállítása és használata JDBC-ben

Az 1. és 2. sorban létrehozzuk és előkészítjük a beszúró utasítást. A beszúrandó értékek helyén paraméterek állnak. A 2. sor mögé egy ciklust tehetünk, amelyben bekérjük a felhasználótól a stúdió nevét és címét, és ezeket a `stúdióNév` és `stúdióCím` változókba tesszük. Ezeket az értékadásokat a kódrészletben csak egy megjegyzéssel jeleztük. A 3. és 4. sor rendeli hozzá az aktuális karaktersorokat, amelyek ebben a sorrendben a `stúdióNév` és `stúdióCím` értékei lesznek, az első, illetve második paraméterhez. Végül az 5. sor végrehajtja a beszúró utasítást, ekkor már a paraméterek aktuális értékét használva. Az 5. sor után visszatérhetünk a ciklusba, ahol a megjegyzéssel jelzett résztől kezdjünk újra a lépések végrehajtását. □

9.6.5. Feladatok

9.6.1. feladat. Oldjuk meg ismét a 9.3.1. feladat problémáit, de ezúttal Javában, JDBC használatával.

9.6.2. feladat. Oldjuk meg ismét a 9.3.2. feladat problémáit, de ezúttal Javában, JDBC használatával.

9.7. PHP

A PHP[3] tulajdonképpen egy olyan szkriptnyelv, amelyet HTML-alapú weblapok elkészítésénél használnak. A JDBC-hez hasonlóan egy betölthető könyvtár segítségével támogatja az adatbázis-műveleteket is. Ebben az alfejezetben vázlatosan áttekintjük a PHP-t, és megmutatjuk, hogyan fejezhetők ki az adatbázis-műveletek ebben a nyelvben.

9.7.1. A PHP alapjai

Minden PHP-kód HTML-szövegen belül van jelen. A böngésző felismeri a speciális címkén (*tag*en) belül elhelyezett PHP-kódot, amelynek az alakja az alábbi:

```
<?php
   a PHP-kód innen kezdődik
?>
```

A C-, illetve Java-programozóknak a PHP több lehetősége ismerősnek tűnhet, ilyenek: az értékadó utasítások, az elágazások és a ciklusok. Így ezekkel nem foglalkozunk most. Van azonban néhány érdekes PHP-lehetőség, amellyel célszerű megismerkedni.

Változók

A változóknak nincs típusa, és deklarálni sem kell őket. Minden változó neve egy $ jellel kezdődik.

A változók általában egy „osztály" elemeként definiáltak, ez esetben ennek a változónak – a Java metódusaihoz hasonlóan – lehetnek *függvényei*. A függvény alkalmazásához használt operátor a -> szimbólum, amely ugyanolyan szereppel bír, mint a pont a C++ vagy a Java nyelvben.

Karaktersorok

A karaktersorértékek a PHP-ban lehetnek akár macskaköröm- ("), akár idéző- (') jelek között, de a két eset között van egy lényeges különbség. Egy idézőjelek közötti karaktersor az SQL-karaktersorokhoz hasonlóan szó szerint értendő, egy macskakörmök közötti karaktersor viszont tartalmazhat változóneveket is, amelyeknek az értéke behelyettesítésre kerül a változó neve helyére.

9.27. példa. A következő kódban a `$x` értéke `Odalép a $foo`.

```
$foo = 'bárhoz';
$x = 'Odalép a $foo';
```

[3] Az érthetőség miatt ezen alfejezet példáiban többször is ékezetes betűket használunk a PHP-kódon belül. A PHP-ban viszont érdemes óvatosan bánni az ékezetes betűk használatával. A PHP-val foglalkozó munkákban ezt részletesen is tárgyalják. *(A fordító megjegyzése.)*

Mire jó a PHP?

Eredetileg a PHP a „Personal Home Page" (magyarul „Személyes Honlap") betűszava volt. Mostanában viszont a „PHP: Hypertext Preprocessor" rekurzív betűszavaként emlegetik a hasonló rekurzív betűszavak szellemében, mint amilyen a GNU is („GNU is Not Unix" = „A GNU nem Unix").

A következő kód lefutásának eredményeként a `$x` értéke ezzel szemben `Odalép a bárhoz` lesz:

```
$foo = "bárhoz";
$x = "Odalép a $foo";
```

Az nem számít, hogy a `bárhoz` idéző- vagy macskaköröm jelek között van-e, hiszen nincs benne változónév. A `$foo` változó értéke viszont csak akkor kerül behelyettesítésre, ha macskakörmök között van, mint a második példánknál is. □

A konkatenációt (vagyis összefűzést) ponttal lehet kifejezni, ezért a

```
$y = "$foo" . 'bárhoz';
```

értékadás hatására `$y` értéke `bárhozbárhoz` lesz.

9.7.2. Tömbök

A PHP közönséges ún. *számozott* (numerikus) tömböket használ, amelyeknek az indexei nullától kezdve természetes számok lehetnek $0, 1, \ldots$. Vannak azonban olyan ún. *asszociatív tömb*jei is, amelyek valójában leképezést adnak. Az ilyen asszociatív tömbök indexei (kulcsai) lehetnek karaktersorok is, ahol a tömb minden kulcsnak egy értéket feleltet meg. Mind a két előbb említett tömb a hagyományos szögletes zárójeleket ([]) használja az indexeléshez. Egy asszociatív tömbelemet pedig a következőképpen ábrázolhatunk:

<kulcs> => <érték>

9.28. példa. A következő sor a `$a` változót egy négyelemű számozott tömbként hozza létre:

```
$a = array(30,20,10,0);
```

Ezáltal a `$a[0]` értéke 30, a `$a[1]` értéke 20 lesz (és így tovább). □

9.29. példa. A következő sor a `$évszakok` változót egy négyelemű asszociatív tömbként hozza létre:

```
$évszakok = array('tavasz' => 'meleg', 'nyár'  => 'forró',
                  'ősz' => 'meleg', 'tél' => 'hideg');
```

A `$évszakok['nyár']` értéke például `'forró'` lesz. □

9.7.3. A PEAR DB könyvtárcsomag

A PHP-nak van egy könyvtárgyűjteménye, amelyet PEAR („PHP Extension and Application Repository" – PHP Kiterjesztés és Alkalmazás Tárház) néven szoktak emlegetni. Ezen könyvtárak egyikének (a DB-nek) a JDBC-metódusokhoz hasonló függvényei vannak. A `DB::connect` függvény segítségével megadhatjuk, hogy melyik kereskedelmi ABKR-hez kívánunk hozzáférni, a DB csomag többi függvényének viszont már nem kell megadni, hogy melyik adatbázist kívánjuk használni. Megjegyezve, hogy a két kettőspont a `DB::connect` utasításban a PHP-módszer arra, hogy jelezzük, hogy „a `connect` függvény a DB könyvtárcsomagban található". A következő utasítás lehetővé teszi a PHP-programunkban a DB könyvtárcsomag elérését:

```
include(DB.php);
```

9.7.4. Adatbázis-kapcsolat létrehozása a DB használatával

A `connect` függvény egy meghívása a következő alakot ölti:

```
$kapcsolatom = DB::connect(<gyártó>://<felhasználói név>:<jelszó>
                           <kiszolgáló neve>/<adatbázis neve>);
```

A fenti függvényhívás komponensei hasonlóak a kapcsolatot létrehozó JDBC-utasítások komponenseihez (lásd 9.6.1. alfejezet). Az egyetlen kivétel a gyártó, amely kódot csak a DB könyvtárcsomag használja. A `mysqli` kód például az aktuális verziójú MySQL-adatbázisra utal.

Az utasítás végrehajtása után a `$kapcsolatom` változó egy adatbázis-kapcsolat lesz. A `$kapcsolatom` a többi PHP-változóhoz hasonlóan megváltoztathatja a típusát. Amíg viszont Connection típusú, addig jó pár olyan hasznos függvényt alkalmazhatunk, amelyek lehetővé teszik a kapcsolt adatbázison történő manipulációkat. Bonthatjuk például a kapcsolatot a következővel:

```
$kapcsolatom->disconnect();
```

Emlékezzünk arra, hogy PHP-ban a `->` segítségével hívhatjuk meg egy „objektum" egy függvényét.

9.7.5. SQL-utasítások végrehajtása

Minden SQL-utasításra „lekérdezésként" hivatkozhatunk, és azzal a `query` függvénnyel hajthatjuk végre azokat, amelynek az utasítás az egyik argumentuma, és amelyet a kapcsolati változón keresztül érhetünk el.

9.30. példa. Ismételjük meg a 9.24. példa beszúró utasítását, amelyben Denzel Washingtont és az *Emlékezz a titánokra!* című filmjét szúrtuk be a `SzerepelBenne` táblába. Tegyük fel, hogy a `$kapcsolatom` már csatlakoztatva van a filmadatbázisunkhoz, ekkor egyszerűen írhatjuk a következőt:

```
$eredmény = $kapcsolatom->query("INSERT INTO SzerepelBenne
    VALUES(" . "'Denzel Washington',
        2000, 'Emlékezz a titánokra!')");
```

Megjegyezzük, hogy a pont által összefűzött két karaktersor alkotja a lekérdezést. Csak azért vágtuk két sorra a lekérdezést, mert az átláthatóság kedvéért jobb két sorba tenni.

A `$eredmény` változóba egy hibakódot kapunk, ha a beszúró utasítás sikertelen volt. Amennyiben viszont a „lekérdezés" egy valódi SQL-lekérdezés, akkor a `$eredmény` az eredménysorokra hivatkozó sormutató lesz (lásd 9.7.6. alfejezet). □

A PHP – a 9.7.7. alfejezetnek megfelelően – az SQL-paraméterek használatát is megengedi kérdőjelek között. A macskakörmök közé tett változó használata viszont egy másik egyszerű, a felhasználó inputjától függő módszert kínál az SQL-utasítások végrehajtására. Mivel a PHP-t weblapokon belül használják, ezért vannak beépített módszerek a HTML lehetőségeinek használatára.

Gyakran a weblapot használó felhasználótól származnak az információk, amelyeket egy űrlapon keresztül kérünk be, majd a válaszokat „visszaküldjük" neki (POST). A PHP erre egy asszociatív `$_POST` nevű tömböt biztosít, amelyben a felhasználó által megadott összes adat szerepel. A kulcsokat itt az űrlap elemei szolgáltatják, a hozzárendelt értékeket pedig a felhasználó által az űrlapon megadott adatok jelentik.

9.31. példa. Tegyük fel, hogy a felhasználótól egy űrlapon keresztül bekértük a `filmcím`, `év` és `színészNév` elemeket. Ez a három érték együttesen a `SzerepelBenne` tábla egy sorának beszúrását alkothatják. Az alábbi utasítás a korábban említett űrlap három elemének értékét kapja meg:

```
$eredmény = $kapcsolatom->query("INSERT INTO SzerepelBenne
    VALUES($_POST['filmcím'], $_POST['év'],
        $_POST['színészNév'])");
```

Mivel a lekérdezés argumentuma egy macskakörmök közötti karaktersor, ezért a PHP kiértékeli a `$_POST['filmcím']` típusú kifejezéseket, és beteszi a kapott értéket a megfelelő helyre. □

9.7.6. A PHP sormutató műveletei

Mikor a `query` függvény argumentuma egy valódi lekérdezés, akkor egy objektummal tér vissza, azaz a soroknak egy listájával. Minden egyes sor egy olyan numerikus tömb lesz, amelynek az indexelése – nullától kezdve – természetes számokkal történik. A legalapvetőbb függvény, amit a kapott objektumra alkalmazhatunk, a `fetchRow()`, amely vagy a következő sorral, vagy a 0 (hamis) értékkel (ha nincs több sor) tér vissza.

```
1) $bevételek = $kapcsolatom->query("SELECT nettóBevétel
       FROM GyártásIrányító");
2) while ($sor = $bevételek->fetchRow()) {
3)     $bevétel = $sor[0];
          // a $bevétel értékének feldolgozása
      }
```

9.23. ábra. A nettó bevételek megkeresése és feldolgozása PHP-ban

9.32. példa. A 9.23. ábrán látható PHP-kód ekvivalens a 9.23., illetve 9.25. példákban szereplő JDBC-kóddal. Itt is feltételeztük, hogy a `$kapcsolatom` már egy elérhető kapcsolatot takar.

Az 1. sorban átadjuk a `$kapcsolatom` kapcsolatnak a lekérdezést, és az eredményobjektumot értékül adjuk a `$bevételek` változónak. Ezután elkezdünk egy ciklust, amelyben minden menetben bekérjük az eredmény egy sorát, amelyet a `$sor` nevű változónak adunk értékül. Ez a változó gyakorlatilag egy egy-hosszú tömb lesz, amelynek az egyetlen komponense a `nettóBevétel` oszlop értéke lesz. Mint ahogyan a C-ben, itt is a `fetchRow()` által visszaadott érték lesz a while ciklus feltétele. Ezáltal, ha nincs már több sor, akkor ez az érték nulla lesz, és így a ciklus is befejeződik. A 3. sorban a sor első (és egyetlen) komponensét értékül adjuk a `$bevétel` változónak. Ennek az értéknek a feldolgozását nem részleteztük. □

9.7.7. Dinamikus SQL a PHP-ban

Csakúgy, mint a JDBC, a PHP is megengedi a kérdőjelek használatát a lekérdezésekben. Ezek a kérdőjelek az olyan értékeknek tartanak fenn helyet, amelyek később, az utasítás végrehajtása alatt kerülnek meghatározásra. Ennek a menete a következőképpen zajlik.

A `prepare` és az `execute` függvények is alkalmazhatók egy kapcsolatra. Ezek nagyon hasonlítanak az azonos nevű függvényekhez, amelyeket a 9.3.9. alfejezetben, illetve máshol is tárgyaltunk. A `prepare` függvény egyetlen argumentuma egy SQL-utasítás, és az utasítás előkészített változatával tér vissza. Az `execute` függvénynek két argumentuma van: az előkészített utasítás és az utasításban szereplő kérdőjeleknek megfelelő helyettesítési értékeknek a tömbje. Ha csak

egy kérdőjel van, akkor ez utóbbi inkább egy egyszerű változó, mintsem egy tömb.

9.33. példa. Tekintsük újra a 9.26. példában szereplő problémát, amelyben több név-cím pár Stúdió relációba történő beszúrására készültünk fel. Kezdetként készítsük el az utasítás paraméteres változatát az alábbival:

```
$előkészítettLekérd = $kapcsolatom->query("INSERT INTO
    Stúdió( név, ". "cím) VALUES(?,?)");
```

Most már a $előkészítettLekérd egy előkészített lekérdezés lesz, amelyet így a stúdió nevét és címét tartalmazó kettő hosszú tömbbel együtt használhatunk az execute függvény argumentumaként. Végrehajthatjuk például a következő utasításokat:

```
$argumentumok = array('MGM', 'Los Angeles');
$eredmény = $kapcsolatom->execute($előkészítettLekérd,
    $argumentumok);
```

Ennek az eljárásnak az előnye ugyanaz, mint a dinamikus SQL-implementációinak. Ha ezen a módon több sort is beszúrunk, akkor csak egyszer kell előkészítenünk a beszúró utasítást, amit ezután többször is végrehajthatunk. □

9.7.8. Feladatok

9.7.1. feladat. Oldjuk meg újra a 9.3.1. feladatot, de most PHP-val.

9.7.2. feladat. Oldjuk meg újra a 9.3.2. feladatot, de most PHP-val.

! **9.7.3. feladat.** A 9.31. példában kiaknáztuk azt a PHP-lehetőséget, amely a macskakörmök közötti karaktersorokban a változó használatát támogatja. Mennyire szükséges ez a lehetőség? JDBC-ben használhatunk hasonlót? Ha a válasz igen, akkor hogyan tehetjük ezt meg?

9.8. Összefoglalás

✦ *Háromrétegű architektúrák:* A nagy mennyiségű felhasználói interakciót támogató, webes elérésű, nagyméretű adatbázis-telepítések általában három szintet különböztetnek meg a folyamatokra nézve: webszerverek, alkalmazásszerverek, illetve adatbázisszerverek. Egy szinten belül több folyamat is lehet aktív, ezek a folyamatok egy vagy több processzorra lehetnek szétosztva.

- *Kliens-szerverrendszerek:* A szabvány szerint egy SQL-kliens hozzákapcsolódik egy SQL-szerverhez, kettejük között létrehozva egy kapcsolatot, valamint egy munkafázist (műveletek sorozatát). Egy munkafázis alatt általában egy modul kerül végrehajtásra, és a végrehajtás alatt álló modult SQL-ágensnek nevezzük.

- *Adatbázis-környezet:* Egy SQL-adatbázis-kezelő telepítésekor kialakul egy új SQL-környezet. Egy környezeten belül az egyes adatbáziselemek (pl. a relációk) csoportosítva lesznek (adatbázis)sémákba, katalógusokba és klaszterekbe. A katalógus sémák gyűjteménye, a klaszter pedig az elemeknek az a legbővebb gyűjteménye, amelyet még egy felhasználó elérhet.

- *A típuseltérés problémája:* Az SQL adatmodellje jelentősen különbözik a hagyományos programozási nyelvek által támogatott adatmodellektől. Ezért az SQL és a befogadó nyelven írt programok közötti adatcsere olyan közös elérésű (osztott) változókon keresztül történhet, amelyek a program SQL nyelvű részeiben a sorok komponenseinek felelnek meg.

- *Beágyazott SQL:* Az SQL-lekérdezések és -módosítások kifejezésére egy általános lekérdező felület használatánál gyakran hatékonyabb, ha hagyományos nyelven írunk programokat, és azokba ágyazzuk be az SQL-utasításokat. Egy előfordító fordítja le a beágyazott SQL-utasításokat a befogadó nyelv megfelelő függvényhívásaira.

- *Kurzorok:* A kurzor egy olyan SQL-változó, amely egy reláció egy sorára hivatkozik. Az SQL és a befogadó nyelven megírt program közötti adatcsere úgy történik, hogy a kurzorral egyenként végigmegyünk egy reláció sorain, az egyes sorok tartalmát beolvassuk közös elérésű változókba, és a befogadó nyelven megírt programmal feldolgozzuk azokat.

- *Dinamikus SQL:* Az SQL-utasításoknak befogadó nyelvű programba való ágyazása helyett lehetőség van arra, hogy a nyelv karakterlánc-változóiban hozzunk létre SQL-utasításokat. Ezeket az SQL-rendszer fogja értelmezni és végrehajtani.

- *Tartósan tárolt modulok:* Az adatbázisséma részeként létrehozhatunk eljárásokat és függvényeket tartalmazó gyűjteményeket. Ezeket egy speciális nyelven írjuk meg, amelyben megtalálhatók az ismert vezérlési szerkezetek és az SQL-utasítások is.

- *Hívásszintű felület:* Van egy szabványos függvénykönyvtár, amelyet SQL/CLI-nek vagy ODBC-nek hívunk, amelyet bármely C-programhoz hozzá lehet szerkeszteni. Ezek a függvények a beágyazott SQL-hez hasonló lehetőségeket nyújtanak, de nincs szükség a használatukhoz előfeldolgozóra.

- *JDBC:* A Java-adatbázis-kapcsolat (Java Database Connectivity) hasonló a CLI-hez, egy Java-osztályok kollekciója, amelynek a segítségével csatlakozhatunk egy adatbázishoz.

- *PHP:* Egy másik népszerű rendszer a hívásszintű felület megvalósítására a PHP. Ezt a nyelvet a HTML-dokumentumokba ágyazva találhatjuk meg. Ez a nyelv lehetővé teszi a weblapok és adatbázisok közötti kommunikációt.

9.9. Irodalomjegyzék

A PSM-szabvány leírása [4]-ben található, és [5] egy átfogó könyv ugyanerről a témáról. Az Oracle által támogatott PSM-verziót PL/SQL-nek nevezik, és [2] ennek egy összefoglaló leírását tartalmazza. Az SQL-szerver egyik verziója az ún. tranzakciós SQL ([6]). [1] az IBM SQL PL környezetének leírásával foglalkozik.

[3] a JDBC népszerű referenciája, [7] pedig hasonló szintű leírás a PHP-ról, amelyet eredetileg a könyv egyik szerzője, R. Lerdorf fejlesztett ki.

[1] D. Bradstock et al., *DB2 SQL Procedure Language for Linux, Unix, and Windows*, IBM Press, 2005.

[2] Y.-M. Chang et al., „Using Oracle PL/SQL"
`http://infolab.stanford.edu/~ullman/fcdb/oracle/or-plsql.html`

[3] M. Fisher, J. Ellis, J. Bruce, *JDBC API Tutorial and Reference*, Prentice-Hall, Upper Saddle River, NJ, 2003.

[4] ISO/IEC Report 9075-4, 2003.

[5] J. Melton, *Understanding SQL's Stored Procedures: A Complete Guide to SQL/PSM*, Morgan-Kaufmann, San Francisco, 1998.

[6] Microsoft Corp., „Transact-SQL Reference"
`http://msdn2.microsoft.com/en-us/library/ms189826.aspx`

[7] K. Tatroe, R. Lerdorf, P. MacIntyre, *Programming PHP*, O'Reilly Media, Cambridge, MA, 2006.

10. fejezet

Haladó témák a relációs adatbázisok tárgykörében

A jelen fejezet által érintett témák főként adatbázis-programozók érdeklődését elégítik ki. A tárgyalást az SQL-szabványban szereplő, az adatbáziselemekhez történő hozzáféréseket meghatározó jogosultságok áttekintésével kezdjük. Ezután áttekintjük az SQL egy olyan kiterjesztését, amely megengedi a rekurzív programozást az SQL-ben, azaz olyan lekérdezések írását, amelyek saját eredményeiket is használják. Ezután pedig áttekintjük az objektumrelációs modellt, és azt is megvizsgáljuk, hogyan van implementálva az SQL-szabványban.

A fejezet fennmaradó részében pedig az OLAP (On-Line Analitic Processing – Online Analitikus Feldolgozás) témakörével foglalkozunk. Az OLAP olyan természetű összetett lekérdezéseket takar, amelyeknek a végrehajtása szignifikáns futási időt eredményez. Mivel végrehajtásuk nagyon költséges, ezért a hatékonyabb kezelésükre kidolgoztak néhány technológiát. Az egyik fontos irányvonala ezen technikáknak egy olyan, „adatkockának" nevezett reláció implementálása, amely lényegesen eltér a relációk konvencionális sorok multihalmaza típusú SQL-megközelítéstől.

10.1. Biztonság és felhasználói jogosultságok SQL-ben

Az SQL kiköti az *engedélyazonosítók* létezését, amelyek lényegében felhasználói nevek. Az engedélyazonosítókat fel lehet ruházni különféle jogokkal hasonlóan ahhoz, ahogyan ezt például egy operációs rendszer fájlrendszerében is megtehetjük. Van egy speciális engedélyazonosító, a `PUBLIC`, amelyet bármelyik felhasználó használhat, azaz a mögötte levő jogosultsággal minden felhasználó fel van ruházva. Egy UNIX-szerű fájlrendszerhez hasonlítva az alábbi analógiát szokás felhozni: egy fájlrendszerben általában értelmezve van az írás, olvasás és program-végrehajtási jogosultság. E három jogosultság megléte elegendő is és

indokolt is, mivel a UNIX operációs rendszerben általában minden erőforrást fájlokkal reprezentálnak, és ez a három művelet jól jellemzi azt, amit egy-egy fájllal tenni szoktak. Azonban az adatbázisok lényegesen bonyolultabbak a fájlrendszereknél, ezért az SQL-szabványban definiált jogosultságok is ennek megfelelően összetettebbek.

Ebben a részben megismerjük, milyen jogosultságokat biztosít az SQL az adatbáziselemeken, hogyan ruházhatunk fel felhasználókat (pontosabban engedélyazonosítókat) különféle jogosultságokkal, illetve hogyan vonhatunk vissza korábban kiadott jogosultságokat.

10.1.1. Jogosultságok

Az SQL kilencféle jogosultságot definiál: `SELECT`, `INSERT`, `DELETE`, `UPDATE`, `REFERENCES`, `USAGE`, `TRIGGER`, `EXECUTE` és `UNDER`. Ezek közül az első négy relációkra vonatkozik: alaptáblákra vagy nézettáblákra. Amint erre a nevükből is következtethetünk, az illető jogosultság tulajdonosának joga van egy tábla lekérdezésére (`SELECT FROM`), új sor beszúrására, sor törlésére és sor módosítására.

Egy SQL-utasítást tartalmazó modul nem hajtható végre a modulban levő SQL-utasítások végrehajtásához szükséges jogosultságok hiányában. Például egy `SELECT-FROM-WHERE` utasítás csak akkor hajtható végre, ha az összes benne hivatkozott táblára van `SELECT` jogosultsága. Hamarosan látni fogjuk, hogyan ruházhatók fel a modulok különféle jogosultságokkal. A `SELECT`, az `INSERT` és az `UPDATE` jogosultságokhoz tartozhat egy attribútumlista is, például: `SELECT(név, cím)`. Az ilyen esetekben csak a megadott attribútumok használhatók, hozzáadhatók vagy módosíthatók a kiválasztás, a beszúrás vagy a módosítás során. Meg kell jegyezni, hogy az ilyen típusú jogosultságok egy konkrét relációra vonatkoznak, ezért a megadásuknál már azt is meg kell fogalmaznunk, hogy melyik reláció `név` és `cím` attribútumaihoz tartoznak.

A `REFERENCES` jogosultság lehetővé tesz egy adott relációra történő hivatkozást egy épségi megszorítási feltételben. E megszorítások a 7. fejezetben megismert lehetséges megszorítások lehetnek: például önálló megszorítások, sor- vagy attribútumalapú megszorítások, vagy hivatkozási épséget ellenőrző megszorítások. A `REFERENCES` jogosultságnál is megadható egy attribútumlista, ekkor viszont csak ezen attribútumokra lehet hivatkozni a megszorításokban. Egy megszorítás csak akkor ellenőrizhető, ha az ellenőrzéséhez szükséges összes adatbáziselemre megvan a `REFERENCES` jogosultság.

A `USAGE` jogosultság számos, a relációktól és megszorításoktól (lásd 9.2.2. alfejezet) eltérő adatbáziselemre alkalmazható. Ez egy adott elem saját deklaráció során történő felhasználhatóságra vonatkozó jog lesz. A `TRIGGER` jogosultság engedélyezi a triggerek definiálását egy konkrét relációhoz. Az `EXECUTE` egy olyan konkrét kódrészlet végrehajtásának a joga, mint például a PSM-eljárások vagy függvények. Végezetül pedig az `UNDER` egy konkrét típusból készített altípusok létrehozásának a jogát jelenti. A típusok témakörét a 10.4. alfejezetben tárgyaljuk részletesebben.

Triggerek és jogosultságok

A triggerekhez tartozó jogosultságok megadásának módja egy kicsit bonyolultabb az eddigieknél. Először is, ha TRIGGER jogosultságunk van egy relációhoz, akkor bármilyen, a relációhoz tartozó trigger létrehozását megkísérelhetjük. A triggerben szereplő feltételek és műveletek viszont lekérdezhetik és/vagy módosíthatják az adatbázisunk egy részét, ezért a trigger létrehozójának rendelkeznie kell az ezeknek megfelelő jogosítványokkal a végrehajtásához. Abban az esetben, mikor valaki egy olyan műveletet hajt végre, ami a trigger lefutását eredményezi, akkor nem kell rendelkeznie azon jogokkal, amelyeket a trigger feltételei és műveletei igényelnek, ugyanis a trigger a létrehozójának jogosultságaival hajtódik végre.

10.1. példa. Tekintsük például, hogy milyen jogosultságok szükségesek a 6.15. ábrán látható beszúró utasítás végrehajtásához (ugyanezt az utasítást a 10.1. ábrán is láthatjuk). Először is, mivel ez a Stúdió relációba szúr be új sorokat, a végrehajtásához szükség van a Stúdió táblán INSERT jogra. Minthogy csak a beszúrni kívánt nevet határozzuk meg, ezért elegendő mind az INSERT, mind az INSERT(név) jogosultság megadása a Stúdió reláción, ez utóbbi jog csak olyan sorok beszúrását engedélyezi, amelyeknek csak a név attribútumának értékét adjuk meg, az újonnan beszúrni kívánt sor többi attribútuma NULL értékű lesz, ami pedig megfelel a 10.1. ábrán lévő utasítás eredményének.

```
1) INSERT INTO Stúdió(név)
2)     SELECT DISTINCT stúdióNév
3)     FROM Filmek
4)     WHERE stúdióNév NOT IN
5)         (SELECT név
6)          FROM Stúdió);
```

10.1. ábra. Új stúdiók felvitele

Vegyük észre, hogy a 10.1. ábrán látható SQL-utasítás két beágyazott lekérdezést is tartalmaz a 2., illetve 5. soroktól kezdődően. Ahhoz, hogy ezeket a lekérdezéseket is kiértékelhessük, szükségünk van a rájuk vonatkozó jogosultságokra is. Azaz szükségünk lesz a SELECT jogra a FROM záradékban szereplő Filmek és Stúdió relációra is. Természetesen az, hogy a Stúdió táblára INSERT jogosultságunk van, nem vonja maga után a SELECT jogosultság birtoklását is, és a fordítottja sem igaz. Mivel mind a Stúdió, mind a Filmek relációnak csak néhány attribútumára van szükségünk, ezért elegendő lesz megadni a SELECT(stúdióNév) jogosultságot a Filmek táblán és a SELECT(név) jogosultságot a Stúdió táblán. Vagy tetszőleges, ezeket az attribútumokat tartalmazó attribútumlistával rendelkező jogosultság is megfelelne erre. □

10.1.2. Jogosultságok kialakítása

A jogosultságok odaítélésének két aspektusa létezik: kezdetben hogyan keletkeznek és hogyan adhatóak tovább a felhasználók között. Most a kezdeti jogosultságokkal foglalkozunk, de a 10.1.4. alfejezetben a jogosítványok átadására is kitérünk.

Minden SQL-elemnek – mint például a sémáknak, moduloknak – van egy tulajdonosa. Egy elem tulajdonosa minden jogosultsággal rendelkezik az illető elem felett. Az SQL-ben a tulajdonosi viszony a következő három mód valamelyikén létesül:

1. Egy séma létrehozásakor a séma és a benne levő elemek alapértelmezés szerint a létrehozó felhasználó tulajdonába kerülnek. Ennek a felhasználónak tehát minden jogosultsága megvan a létrehozott elemekre vonatkozóan.

2. Egy munkafázist kezdeményező `CONNECT` utasításnál lehetőségünk van a munkafázist kezdeményező felhasználó nevének a megadására egy `AUTHORIZATION` záradékban. Például a

```
CONNECT TO Matild-sql-szerver AS kapcs1
    AUTHORIZATION kirk;
```

   utasítás létrehoz egy kapcsolatot a `Matild-sql-szerver` adatbázisszerverrel a `kirk` nevű felhasználó nevében, a kapcsolatra a későbbiekben `kapcs1` néven hivatkozhatunk. Az SQL adatbázis-kezelő általában valamilyen módon megbizonyosodik arról, hogy a felhasználó érvényes felhasználó-e. Ezt megtehetik például egy jelszó bekérésével. A jelszót a 9.2.5. alfejezetnek megfelelő módon az `AUTHORIZATION` záradék után is megadhatjuk. Ez a megközelítés viszont biztonsági kockázatot jelenthet, hiszen a jelszó látható lesz, és így bárki megszerezheti, mikor Kirk begépeli.

3. Egy modul létrehozásakor az `AUTHORIZATION` kulcsszót követő záradékban adhatjuk meg a létrehozott modul tulajdonosát. Például az

```
AUTHORIZATION picard;
```

   záradékkal kiegészített moduldefiníciós utasítással létrehozott modul tulajdonosa `picard` nevű felhasználó lesz. Előfordulhat, hogy egy modulnak egyáltalán nincs megadva a tulajdonosa. Ezek a modulok mindenki által végrehajthatók, de a sikeres végrehajtáshoz szükség van a modulban levő SQL-utasítások végrehajtásához szükséges más forrásból származó jogosultságokra is, amelyekkel például a végrehajtást kezdeményező felhasználó rendelkezhet.

10.1.3. Jogosultságok ellenőrzése

Mint azt korábban már láthattuk, minden modulnak, sémának és munkafázisnak van egy tulajdonosa: SQL-terminológiával ezt úgy mondhatnánk, hogy mindegyikhez hozzá van rendelve egy engedélyazonosító. Egy SQL-művelet végrehajtásában általában két elem vesz részt:

1. A művelet által elért és módosított adatbáziselemek.
2. A művelet végrehajtását kezdeményező ágens.

A végrehajtó ágens a végrehajtáshoz szükséges jogokat egy bizonyos engedélyazonosító, az ún. *aktuális engedélyazonosító* alapján kapja. Az aktuális engedélyazonosító

a) vagy a modul engedélyazonosítója, ha a végrehajtás alatt álló modulnak van engedélyazonosítója;

b) vagy pedig a munkafázis-engedély azonosítója.

Egy SQL-művelet csak akkor hajtható végre, ha az aktuális engedélyazonosító biztosítja az ehhez szükséges összes jogosultságot.

10.2. példa. A jogosultságok ellenőrzési mechanizmusának áttekintéséhez tekintsük újra a 10.1. példát. Feltételezhetjük, hogy a hivatkozott `Filmek` és `Stúdió` táblák a `FilmSéma` sémában lettek létrehozva, és a séma létrehozója – így tulajdonosa is – `janeway`. A tulajdonos, `janeway` minden jogosultsággal rendelkezik a `FilmSéma` sémához tartozó táblák és adatbáziselemek felett. A tulajdonos másokat is felruházhat az említett adatbáziselemekre vonatkozó jogosultságokkal, ahogyan azt a 10.1.4. alfejezetben majd megismerjük, de most tegyük fel, hogy még nem adott másoknak semmilyen jogosultságot ezekre a táblákra. Számos mód van a 10.1. példa beszúrási műveletének végrehajtására.

1. Az említett beszúrási művelet végrehajtódhat egy olyan modulban, amelyet `janeway` hozott létre oly módon, hogy a létrehozó utasítás tartalmazta az `AUTHORIZATION janeway` záradékot. A modul végrehajtásánál az aktuális engedélyazonosító a modulhoz eltárolt engedélyazonosító lesz (ha egyáltalán van). Ez azt jelenti, hogy az SQL-utasítás végrehajtása során pontosan `janeway` jogosultságaival fut, aki – mivel `janeway` tulajdonos – minden jogosultságot birtokol a `Filmek` és `Stúdió` táblákra vonatkozóan.
2. Az említett beszúrási művelet végrehajtható egy olyan modulból is, amelynek nincs tulajdonosa. Ha ilyenkor a `janeway` azonosítójú felhasználó bejelentkezik egy `CONNECT` utasítással az `AUTHORIZATION janeway` záradékot is megadva, akkor az aktuális engedélyazonosító `janeway` lesz, az ő jogosultságaival kezd el futni az illető modul, így minden jogosultsága megvan az illető beszúrás elvégzésére.

3. Most tegyük fel, hogy a tulajdonos, `janeway` felhasználó a szóban forgó táblákra vonatkozó összes jogosultsággal felruházza az `archer` nevű másik felhasználót (vagy esetleg a `PUBLIC` azonosítójú, speciális felhasználót, amely lényegében az „összes felhasználót" jelenti). Ha most a szóban forgó beszúrási utasítás egy olyan modulban van, amely el van látva az

   ```
   AUTHORIZATION archer
   ```

   záradékkal, akkor mivel a modul futtatása `archer` felhasználó jogosultságaival történik, és neki megvannak a végrehajtáshoz szükséges jogosultságai, ezért a beszúrás sikeres lesz.

4. Mint az előző – azaz 3. – esetben, itt is tegyük fel, hogy `janeway` felhasználó az illető táblákra vonatkozó összes jogosultsággal felruházta az `archer` nevű felhasználót, de most az említett beszúrási utasítás legyen egy olyan modulban, amelynek nincs tulajdonosa. Ha ez a modul egy olyan munkafázisban lesz végrehajtva, amelynek az engedélyazonosítója egy `AUTHORIZATION archer` záradékkal lett beállítva, akkor az aktuális engedélyazonosító `archer`, ezért a végrehajtáshoz szükséges jogosultságok megvannak.

□

Számos jogosultságokkal kapcsolatos alapelvet megismerhettünk a 10.2. példában, amelyet most röviden összefoglalunk:

- A végrehajtáshoz szükséges jogosultságok rendelkezésre állnak, ha az elérni kívánt elemek és adatok tulajdonosának az engedélyazonosítója megegyezik az aktuális engedélyazonosítóval. Erre volt példa az 1. és 2. eset.

- A végrehajtáshoz szükséges jogosultságok rendelkezésre állnak, ha az aktuális engedélyazonosítóval rendelkező felhasználónak a tulajdonos jogosultságot adott az általa elérni kívánt adatbáziselemek elérésére, vagy ha a tulajdonos a `PUBLIC` nevű felhasználónak adta meg a szóban forgó jogosultságokat. Erre volt példa a 3. és 4. eset.

- Ha egy olyan modult hajtunk végre, amelynek tulajdonosa megegyezik az elérni kívánt adatok tulajdonosával, akkor megvannak a végrehajtáshoz szükséges jogosultságok. Természetesen a modul `EXECUTE` jogával ekkor is rendelkezni kell. Erre volt példa az 1. és 3. eset.

- Egy másik legális végrehajtási mód, ha egy nyilvánosnak minősített modult hajtunk végre egy olyan munkafázisban, amely munkafázisnak az aktuális engedélyazonosítója olyan felhasználóé, akinek jogosultsága van a szükséges adatelemek elérésére. Ezt szemléltettük a fenti 2. és 4. esetekben.

10.1.4. Jogosultságok megadása

Ezidáig jogosultságok szerzésének csak egyetlen módjával ismerkedtünk meg: egy elemre vonatkozóan tulajdonosi, illetve létrehozó minőségben megszerzett jogosultságokkal. Az SQL biztosítja a `GRANT` utasítást, amellyel egy felhasználó jogosultságokat ruházhat át egy másik felhasználóra – egy felhasználó az átruházott jogosultságait nem veszíti el, így itt inkább jogosultságok „lemásolásáról" beszélhetünk.

Van egy lényeges különbség a jogosultság másolása és átruházása között. Minden egyes jogosultsághoz tartozik egy *engedélyezési képesség* is, amely a jogosultságok továbbadásakor („lemásolásakor") nem kerül automatikusan továbbadásra. Ha például egy felhasználónak van `SELECT` jogosultsága a `Filmek` táblára engedélyezési képességgel együtt, egy másik felhasználónak pedig ugyanezen jogosultsága van, de engedélyezési képesség nélkül, akkor mindketten elérhetik az illető táblát SQL-lekérdezési műveletekben. A két felhasználó lehetőségei abban különböznek egymástól, hogy míg az első – engedélyezési képességgel bíró – felhasználó megadhatja e jogosultságot egy harmadik személynek (esetleg az engedélyezési képességével együtt), addig a másodikként említett – engedélyezési képességgel nem bíró – felhasználó ezt a jogosultságát nem adhatja tovább másoknak. Ha a harmadik személy is rendelkezik engedélyezési képességgel, akkor ő is adhat jogosultságot - engedélyezési képességgel vagy anélkül – egy negyedik felhasználónak, és így tovább.

Egy *engedélyező utasítás* formája az alábbi:

`GRANT` <jogosultságok listája> `ON` <adatbáziselemek>
`TO` <felhasználók nevei>

és ezt követheti még egy `WITH GRANT OPTION` rész is.

Az adatbáziselem leggyakrabban egy reláció, amely lehet akár egy tábla vagy egy nézet. Amennyiben más típusú elemünk van, akkor az elem nevét meg kell előznie az elem típusának, ilyen például az `ASSERTION`. A jogosultséglista egy vagy több jogosultságot tartalmazhat. Lehetőségünk van az `ALL PRIVILEGES` kulcsszó használatához, ami röviden az engedélyező által birtokolt, az összes adatbáziselemhez tartozó jogosultságokat fejezi ki.

Hogy ez az utasítás sikeresen végrehajtható legyen, szükség van arra, hogy az ezt végrehajtó felhasználó rendelkezzen a jogosultságokkal és a hozzájuk tartozó engedélyezési képességgel is. Az engedélyező annál általánosabb jogosultságokkal is rendelkezhet (a megfelelő engedélyező képességekkel együtt), mint amit átad. Például, átadhatja az `SELECT` vagy `INSERT(név)` jogosultságot a `Stúdió` táblára, miközben ő rendelkezik a `Stúdió` táblára vonatkozó `INSERT` jogosultsággal (engedélyező képességekkel együtt).

10.3. példa. A `FilmSéma` sématulajdonosa, `janeway` átruházza a `Stúdió` táblára vonatkozó `SELECT` és `INSERT` jogosultságát, valamint a `Filmek` táblára vonatkozó `SELECT` jogosultságát `kirk` és `picard` felhasználóknak.

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
Stúdió(név, cím, elnökAzon)
```

Sőt átadja az ezekre vonatkozó engedélyezési képességét is. Az ehhez szükséges GRANT utasítás alakja a következő:

```
GRANT SELECT, INSERT ON Stúdió TO kirk, picard
    WITH GRANT OPTION;
GRANT SELECT ON Filmek TO kirk, picard
    WITH GRANT OPTION;
```

Most ha picard tovább akarja adni az előbb megszerzett jogosultságait sisko felhasználónak, de az engedélyezési képessége nélkül, akkor ezt az alábbi utasításokkal teheti meg:

```
GRANT SELECT, INSERT ON Stúdió TO sisko;
GRANT SELECT ON Filmek TO sisko;
```

Ezek után kirk is átadja sisko-nak a Stúdió táblára vonatkozó SELECT és INSERT(név) jogosultságát, és a Filmek táblára vonatkozó SELECT jogosultságát. Ezt az alábbi utasításokkal tette meg:

```
GRANT SELECT, INSERT(név) ON Stúdió TO sisko;
GRANT SELECT ON Filmek TO sisko;
```

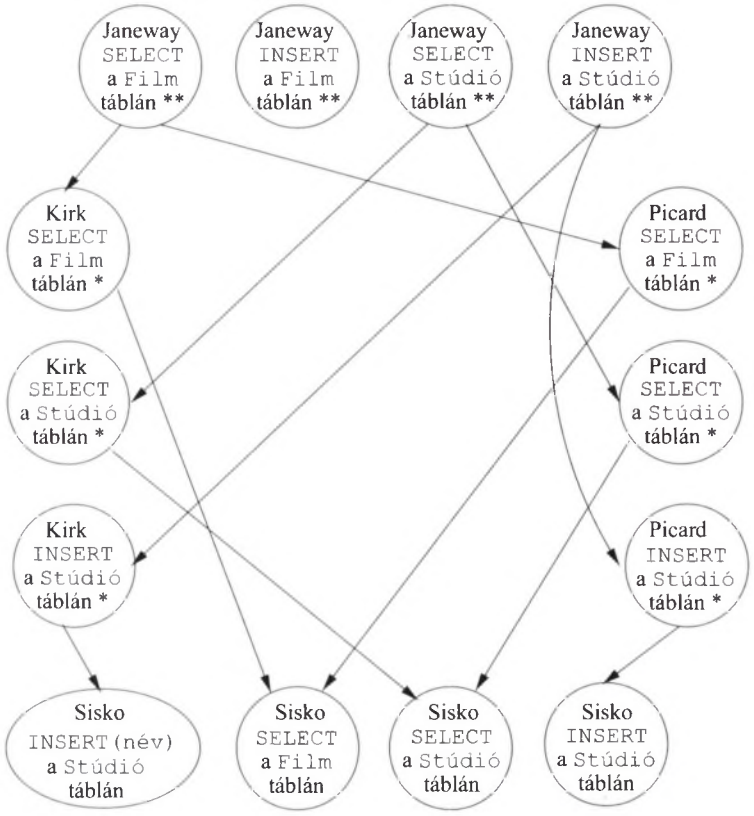
Látható, hogy sisko a Filmek és a Stúdió táblákra vonatkozó SELECT jogosultságát két felhasználótól is megkapta. Az INSERT(név) jogosultságát a Stúdió táblára szintén két helyről is megkapta: közvetlen kirk-től, valamint picard-tól is picard INSERT jogosultságának korlátozásával. □

10.1.5. Engedélyezési diagramok

Mivel a jogosultságok továbbadása során jogosultságok bonyolult rendszere jöhet létre, ezért a kiosztott jogosultságok áttekintésének segítségére kidolgoztak egy jól használható eszközt, az *engedélyezési diagram*ot. Egy SQL-rendszer egy ilyen diagramot tárol az egyes felhasználók jogosultságainak nyilvántartására, illetve annak nyilvántartására, hogy mely jogosultság mely felhasználótól származik, kitől lett átruházva (ez utóbbi különösen akkor érdekes, ha egyes jogosultságokat vissza kívánnak vonni; a 10.1.6. alfejezetetben tárgyaljuk).

Az engedélyezési diagram egy olyan gráf, amelynek csomópontjai megfeleltetésbe hozhatók egy felhasználóval és egy jogosultsággal. Ne feledjük el, hogy annak a képessége (mint például az R relációra vonatkozó SELECT), hogy valamit használhatunk és átruházhatunk, illetve hogy valamit használhatunk és nem ruházhatunk át, az két külön jogosultságot jelent. Ez a két jogosítvány két csúcsot kell jelentsen, még abban az esetben is, ha ugyanahhoz a felhasználóhoz tartozna. Hasonlóan, ha egy felhasználónak egy általánosabb, illetve egy szűkebb jogosultsága is van ugyanarra, mint például a SELECT az R-en, illetve a SELECT az $R(A)$-n, akkor is két külön csúcs reprezentálja őket.

Ha egy U felhasználó átruház egy P jogosultságot egy V felhasználónak, és ezt U egy Q jogosultsága alapján teszi (ekkor Q lehet P, vagy P-t tartalmazó általánosabb jogosultság, mindkettő engedélyezési képességgel együtt), akkor ezt a tényt a diagramon az U/Q csomóponttól a V/P csomópontig húzott vonallal jelölhetjük. Amint láthatjuk, előfordulhat, hogy bizonyos jogosultságok elvesznek egy gráfél törlésénél. Éppen emiatt választjuk ketté az olyan egymást tartalmazó jogosultságpárokat, mint például a jogosultság engedélyezési képességgel, illetve anélkül. Így ha az erősebb jogosultságot elveszítjük, attól még a gyengébb megmaradhat nekünk.



10.2. ábra. Egy engedélyezési diagram

10.4. példa. A 10.2. ábra szemlélteti a 10.3. példában megadott engedélyezési műveletek végrehajtásakor létrejött engedélyezési diagramot. A jelölési konvenciónkról megjegyezzük, hogy egy felhasználó-jogosultság pár mögé tett * karakterrel jelöljük az illető jogosultságra vonatkozó engedélyezési képességet, két csillag (azaz **) karakterrel jelöljük a tulajdonosi viszonyból származó jogosultságokat (ezek a jogosultságok nem engedélyezéssel születtek). Ez a meg-

különböztetés a jogosultságok visszavonásakor lesz majd érdekes, mint azt a 10.1.6. alfejezetben látni fogjuk. A kétcsillagos jogosultság maga után vonja az engedélyezési képességet. □

10.1.6. Jogosultságok visszavonása

A megszerzett jogosultságok bármikor visszavonhatók. Egy jogosultság visszavonásának *következetes*nek kell lennie, ami azt jelenti, hogy ha egy engedélyezési képességgel kiegészített jogosultságot vonnak vissza, akkor az illető jogosultsággal és engedélyezési képességgel más felhasználóknak átruházott jogosultságokat is mind vissza kell vonni. A jogosultság-visszavonási utasítás legegyszerűbb alakja a következő:

`REVOKE` <jogosultséglista> `ON` <adatbáziselemek listája> `FROM` <felhasználók listája>

Ezenkívül a fenti utasítást kiegészíthetjük az alábbi elemekkel:

1. A `CASCADE` kulcsszó használatával, miután a megadott jogosultságok visszavonásra kerültek, azon jogosultságokat is visszavonjuk, amelyek csak a visszavont jogosultságok miatt voltak engedélyezve. Pontosabban, ha az U felhasználó azt a P jogosítványt vonja vissza a V felhasználótól, amely az U Q jogosultságán alapult, akkor az engedélyezési diagram U/Q és V/P éle is törlődni fog. Így törölve lesznek azok a csomópontok is, amelyekbe nem vezet él tulajdonosi jogosultságokat reprezentáló csúcsokból (két csillaggal ábrázoltakból).

2. A `RESTRICT` kulcsszó használata esetén jogosultság-visszavonási utasítás nem hajtható végre, ha az előző alfejezetben leírt következetes jogosultság-visszavonási szabályok szerint olyan jogosultságok visszavonását eredményezné, amelyek tovább lettek már adva.

A `REVOKE` kulcsszó helyett írhatjuk a `REVOKE GRANT OPTION FOR` kulcsszavakat is, ami nem magának a jogosultságnak, hanem csak engedélyezési képességének a visszavonását eredményezi. Lehet, hogy módosítanunk kell egy csúcsot, egy irányított élet, vagy létre kell hoznunk egy új csúcsot ahhoz, hogy a felhasználón történt változtatásokat érzékeltethessük. A `REVOKE` utasítás ilyen alakját is követnie kell egy `CASCADE` vagy `RESTRICT` kulcsszónak.

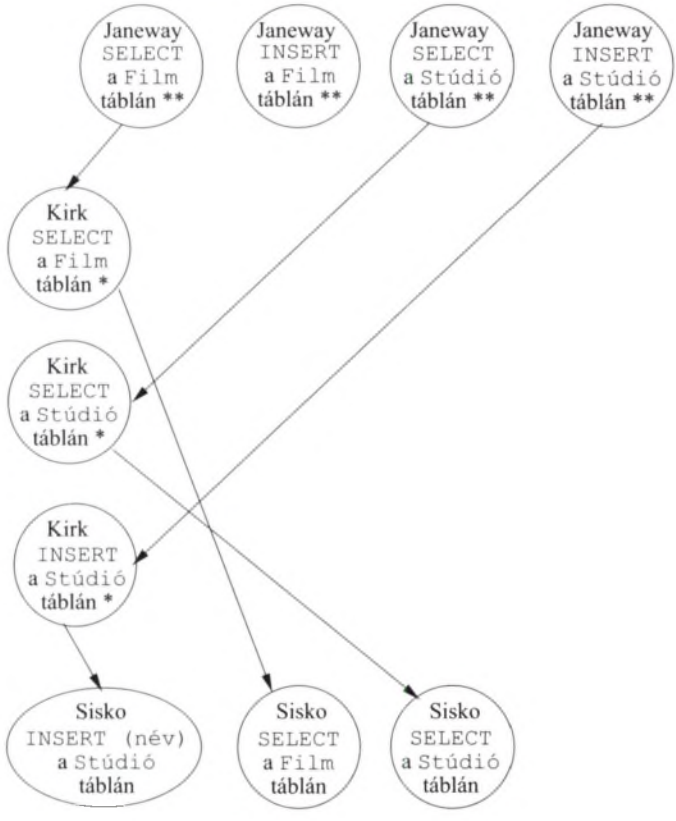
10.5. példa. A 10.3. példát folytatva tételezzük fel, hogy `janeway` visszavonja `picard`-nak átadott jogosultságait az alábbi utasításokkal:

```
REVOKE SELECT, INSERT ON Stúdió FROM picard CASCADE;
REVOKE SELECT ON Filmek FROM picard CASCADE;
```

A 10.2. ábráról töröltük `janeway` megfelelő jogosultságaiból `picard` megfelelő jogosultságaihoz vezető nyilakat. Mivel a `CASCADE` kulcsszót is megadtuk,

ezért töröltük azokat a jogosultságokat is, amelyek nem érhetők el kétcsillagos (tulajdonosi) jogosultságokat reprezentáló csúcsokból. A 10.2. ábrát megvizsgálva látható, hogy `picard` jogosultságai nem érhetők el kétcsillagos csomópontból (hiszen, ha elérhetők lennének, akkor vezetne oda út). Azt is láthatjuk, hogy `sisko INSERT` jogosultsága a `Stúdió` táblán már szintén nem érhető el. Ezért nemcsak `picard` jogosultságait kell törölni a diagramról, hanem `sisko INSERT` jogosultságát is.

Látható, hogy nem kell törölni `sisko SELECT` jogosultságát a `Filmek` és `Stúdió` táblákra, és nem kell törölni az `INSERT` jogosultságát a `Stúdió(név)` sorokra, mivel ezek elérhetők `janeway` tulajdonosi jogosultságait reprezentáló csúcsokból. A diagramon az elmondott változtatásokat elvégezve a 10.3. ábrát kapjuk. □



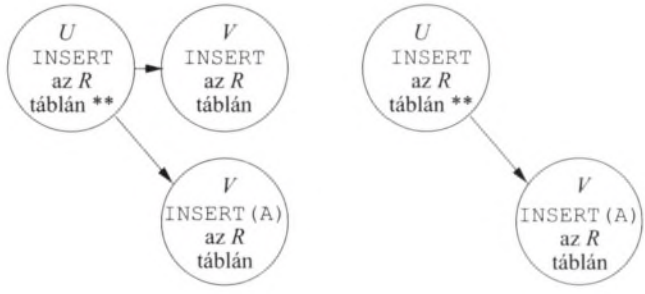
10.3. ábra. Az engedélyezési diagram `picard` jogosultságainak visszavonása után

10.6. példa. Van még néhány egyéb vonatkozás is, amelyet absztrakt példákkal fogunk szemléltetni. Először is, amikor visszavonunk egy általános p jogosultságot, akkor nem vonjuk vissza p specializált változatait. Tekintsük erre

a következő példát, melyben U felhasználó – az R reláció tulajdonosa – ad egy `INSERT` jogosultságot az R relációra V felhasználónak, és ezenkívül ad egy `INSERT(A)` jogosultságot is ugyanerre a relációra.

Lépés	*Végrehajtója*	*Tevékenység*
1.	U	`GRANT INSERT ON` R `TO` V
2.	U	`GRANT INSERT(A) ON` R `TO` V
3.	U	`REVOKE INSERT ON` R `FROM` V `RESTRICT`

Amikor U visszavonja `INSERT` jogosultságát V-től, V felhasználó `INSERT(A)` jogosultsága megmarad. A 2. és 3. lépések után az engedélyezési diagram állapotát a 10.4. ábrán szemléltetjük.



(a) A 2. lépés után (b) A 3. lépés után

10.4. ábra. Egy általánosabb jogosultság visszavonásakor a specifikusabb jogosultságok megmaradnak

Figyeljük meg, hogy a 2. lépés után két elválasztott csomópont marad V felhasználó két hasonló, de nem egyező jogosultságának ábrázolására. Szintén észrevehető, hogy a `RESTRICT` kulcsszó nem akadályozta meg a jogosultság visszavételét, mivel V nem adta tovább e jogosultságát más felhasználóknak. A V nem is adhatta volna tovább, mivel nem volt továbbadási képessége. □

10.7. példa. Most tekintsünk egy hasonló példát, ahol U átad V-nek egy $p*$ jogosultságot, amely az engedélyezési képességet is magában foglalja, ezután pedig az engedélyezési képességet visszavonja. Tegyük fel azt is, hogy az U engedélye a saját $q*$ jogosultságán keresztül történhetett meg. Ekkor viszont az $U/q*$ és $V/p*$ csúcsok közötti élet felül kell írjuk egy $U/q*$-ból V/p-be mutató éllel. Ha nem volt V/p csúcs, akkor létre kell hozni egyet. Normál körülmények között ekkor a $V/p*$ elérhetetlenné válik a p összes V-re vonatkozó engedélyével együtt. Habár lehetséges, hogy V felhasználó egy másik U-tól különböző felhasználótól is kapott $p*$ jogot, és ez esetben a $V/p*$ csúcs elérhető marad.

Az elvégzett lépések sorozata a következő:

Lépés	*Végrehajtója*	*Tevékenység*
1.	U	`GRANT` p `TO` V `WITH GRANT OPTION`
2.	V	`GRANT` p `TO` W
3.	U	`REVOKE GRANT OPTION FOR` p `FROM` V `CASCADE`

Az 1. lépésben U átadja a p jogosultságot V-nek engedélyezési képességgel. A 2. lépésben V az 1. lépésben szerzett engedélyezési képességét felhasználva átadja a p jogosultságot W-nek. Ezután a diagram a 10.5 (a) ábra szerint néz ki.

(a) A 2. lépés után (b) A 3. lépés után

10.5. ábra. Egy adott jogosultság engedélyezési képességének visszavonása nem vonja maga után az illető jogosultság visszavonását

Majd a 3. lépésben U visszavonja az engedélyezési képességet a p jogosultságra vonatkozóan V-től, de magát a jogosultságot nem vonja vissza. Mivel nincs V/p csúcsunk, ezért létrehozunk egyet. Az $U/p**$ és $V/p*$ közötti élet eltávolítjuk, és berajzolunk egy $U/p**$ és V/p közötti élet. Ezáltal a V/p és W/p csúcsok a $**$ csúcsokból elérhetetlenné váltak, ezért ezeket a csúcsokat ki is töröljük a diagramunkból. A kapott engedélyezési diagram a 10.5 (b) ábrán látható. □

10.1.7. Feladatok

10.1.1. feladat. Adjuk meg, milyen jogosultságok szükségek az alábbi lekérdezések végrehajtásához. Minden lekérdezéshez jelöljük meg mind a legspecifikusabb, mind pedig a legáltalánosabb szükséges jogosultságokat.

a) A 6.5. ábra lekérdezése.

b) A 6.7. ábra lekérdezése.

c) A 6.15. ábrán látható beszúrási művelet.

d) A 6.37. példán látható törlési művelet.

e) A 6.39. példában látható módosító utasítás.

f) A 7.3. ábrán látható ellenőrzés.

g) A 7.11. példában látható önálló megszorítás.

10.1.2. feladat. Mutassuk meg a 4. és a 6. lépés végrehajtása után kialakult engedélyezési diagramot a 10.6. ábrán leírt műveletek végrehajtásakor. Tegyük fel, hogy a p jogosultság olyan relációra vonatkozik, melynek tulajdonosa A.

Lépés	*Végrehajtója*	*Tevékenység*
1.	A	GRANT p TO B WITH GRANT OPTION
2.	A	GRANT p TO C
3.	B	GRANT p TO D WITH GRANT OPTION
4.	D	GRANT p TO B, C, E WITH GRANT OPTION
5.	B	REVOKE p FROM D CASCADE
6.	A	REVOKE p FROM C CASCADE

10.6. ábra. A 10.1.2. feladathoz tartozó műveletsorozat

10.1.3. feladat. Mutassuk meg az 5. és a 6. lépés végrehajtása után kialakult engedélyezési diagramot a 10.7. ábrán leírt műveletek végrehajtásakor. Tegyük fel, hogy a p jogosultság olyan relációra vonatkozik, melynek tulajdonosa A.

Lépés	*Végrehajtója*	*Tevékenység*
1.	A	GRANT p TO B, E WITH GRANT OPTION
2.	B	GRANT p TO C WITH GRANT OPTION
3.	C	GRANT p TO D WITH GRANT OPTION
4.	E	GRANT p TO C
5.	E	GRANT p TO D WITH GRANT OPTION
6.	A	REVOKE GRANT OPTION FOR p FROM B CASCADE

10.7. ábra. A 10.1.3. feladathoz tartozó műveletsorozat

! **10.1.4. feladat.** Mutassuk meg az alábbi lépések végrehajtása után kialakult engedélyezési diagramot. Tételezzük fel, hogy a p jogosultság egy olyan táblára vonatkozik, amelynek tulajdonosa az A felhasználó.

Lépés	*Végrehajtója*	*Tevékenység*
1.	A	GRANT p TO B WITH GRANT OPTION
2.	B	GRANT p TO B WITH GRANT OPTION
3.	A	REVOKE p FROM B CASCADE

10.2. Rekurzió az SQL-ben

Az SQL-99 szabványa tartalmazza a lekérdezések rekurzív meghatározásának lehetőségét. Létezik legalább egy nagy rendszer – az IBM DB2 –, amely implementálja az SQL-99 előírásait annak ellenére, hogy az SQL-99 szabványnak nem „lényegi" része az, hogy minden ABKR implementálja ezt a lehetőséget. Ebben a részben ezzel foglalkozunk.

10.2.1. Rekurzív relációk definiálása az SQL-ben

A `WITH` utasítás segítségével SQL-ben megfogalmazhatjuk ideiglenes relációnak a rekurzív vagy nem rekurzív definícióját. Rekurzív reláció megadásánál a relációt a `WITH` utasításon belül használhatjuk. A `WITH` utasítás egyszerű alakja:

`WITH` R `AS` <R definíciója> <R-et használó lekérdezés>

Tehát definiálunk egy R nevű ideiglenes relációt, majd használjuk egy lekérdezésben. Az ideiglenes relációk csak a `WITH` utasításhoz tartozó lekérdezéseken belül használhatók.

Általában több reláció is definiálható a `WITH` után, a definíciókat vesszővel elválasztva. A definíciók közül bármelyik lehet rekurzív. A definiált relációk lehetnek kölcsönösen rekurzívak, azaz mindegyiket néhány másik függvényében definiáljuk (akár önmagukkal is). Minden olyan relációt, amely rekurzív definícióban vesz részt, a `RECURSIVE` kulcsszó kell hogy megelőzzön. A `WITH` utasítás általános alakja a 10.8. ábrán található.

```
WITH
   [RECURSIVE] R1 AS <R1 definíciója>,
   [RECURSIVE] R2 AS <R2 definíciója>,
     ...
   [RECURSIVE] Rn AS <Rn definíciója>
<R1, R2, ..., Rn relációkat tartalmazó lekérdezés>
```

10.8. ábra. A `WITH` utasítás több ideiglenes relációra vonatkozó definíciója

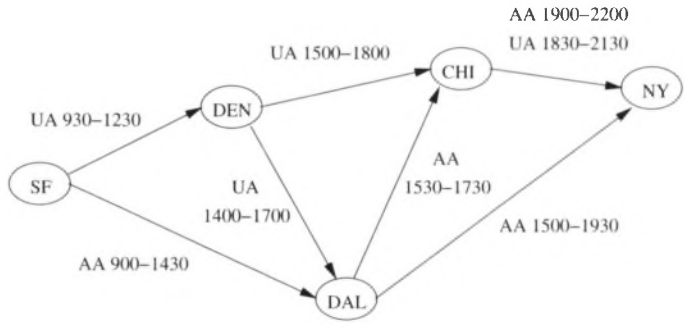
10.8. példa. Sok olyan példa található a gráfok útjainak tanulmányozása esetén, amely rekurziót használ. A 10.9. ábra egy gráfot reprezentál, amely két fiktív légitársaság (*Untried Airlines* (UA), *Arcane Airlines* (AA)) San Francisco, Denver, Dallas, Chicago és New York városok között közlekedő járatait tartalmazza. A gráfot az alábbi relációval ábrázolhatjuk:

```
Járatok(légitársaság, honnan, hova, indulás, érkezés)
```

Az ábra konkrét sorait a 10.9. ábra szemlélteti.

A legegyszerűbb rekurzív kérdés, amelyet feltehetünk, hogy „Mely (x, y) várospárokra lehetséges egy vagy több átszállással eljutni x városból y városba?". Mielőtt megírnánk a rekurzív SQL-lekérdezést, érdemes a rekurziót az 5.3. alfejezetben használt jelöléssel Datalogban is kifejezni, hiszen számos elképzelést, mint a rekurziót is, könnyebb kifejezni Datalogban, mint SQL-ben. Szükségünk lehet a korábbi részben tárgyalt terminológia áttekintésére. A következő két Datalog-szabály írja le az `Eljut(x,y)` relációt, amely már a valódi várospárokat fogja a tartalmazni:

```
1. Eljut(x,y) ← Járatok(l,x,y,i,é)
2. Eljut(x,y) ← Eljut(x,z) AND Eljut(z,y)
```



10.9. ábra. Néhány járat egy térképe

légitársaság	*honnan*	*hova*	*indulás*	*érkezés*
UA	SF	DEN	930	1230
AA	SF	DAL	900	1430
UA	DEN	CHI	1500	1800
UA	DEN	DAL	1400	1700
AA	DAL	CHI	1530	1730
AA	DAL	NY	1500	1930
AA	CHI	NY	1900	2200
UA	CHI	NY	1830	2130

10.10. ábra. A `Járatok` reláció sorai

Az első szabály azt fejezi ki, hogy az `Eljut` reláció minden olyan várospárt tartalmaz, amely városok között közvetlen járat van az elsőből a másodikba. Az l légitársaság, az i indulási idő és az $é$ érkezési idő tetszőlegesek a szabályban. A második szabály azt fogalmazza meg, hogy ha az x városból már eljutottunk z városba, és z városból eljuthatunk y városba, akkor x-ből is el tudunk jutni y-ba.

Egy rekurzív szabály kiértékeléséhez a Datalog-szabályok többszöri alkalmazására van szükségünk. Kezdetben az `Eljut` relációt üresnek feltételezzük. Az első szabály alkalmazásával a következő `Eljut` párokat kapjuk: (`SF`, `DEN`), (`SF`, `DAL`), (`DEN`, `CHI`), (`DEN`, `DAL`), (`DAL`, `CHI`), (`DAL`, `NY`) és (`CHI`, `NY`). Ez pontosan az a hét irányított él, amelyet a 10.9. ábra is tartalmaz.

A következő menetben a második szabályt alkalmazzuk rekurzív módon azért, hogy összevonhassuk az olyan élpárokat, amelyeknél az egyik végpontja a másik kezdőpontja lesz, ekkor a következőket kapjuk: (`SF`, `CHI`), (`DEN`, `NY`) és (`SF`, `NY`). A harmadik menet már az összes egy élből vagy két élből álló párokat rakja össze, és ezzel maximum négy hosszúságú irányított élből álló utakat nyerünk. A mostani diagramnál viszont ekkor már nem kapunk újabb párokat, ezért az `Eljut` reláció tíz olyan (x, y) párt fog tartalmazni, amelyekben a 10.9. ábra alapján az x-ből elérhető lesz az y.

A fenti Datalog-szabályok alapján megadható az Eljut reláció ekvivalens SQL-definíciója. Az SQL-lekérdezés képezi az Eljut definícióját egy WITH utasítás segítségével, és a WITH utasítást kibővítjük a kívánt lekérdezéssel. A példánkban az eredmény az egész Eljut reláció volt, megjegyezve, hogy a lekérdezésben feltételeket is kiköthetnénk az Eljut-ra, például azt, hogy csak a Denverből elérhető városokat kapjuk meg.

```
1)  WITH RECURSIVE Eljut(honnan, hova) AS
2)          (SELECT honnan, hova FROM Járatok)
3)      UNION
4)          (SELECT R1.honnan, R2.hova
5)           FROM Eljut R1, Eljut R2
6)           WHERE R1.hova = R2.honnan)
7)  SELECT * FROM Eljut;
```

10.11. ábra. Légi kapcsolatban lévő városokat meghatározó SQL-lekérdezés

A 10.11. ábra lekérdezése bemutatja az Eljut relációt meghatározó SQL-lekérdezést. Az 1. sor bevezeti az Eljut relációt, és a 2–6. sorok között található a reláció definíciója.

Mint ahogyan az Eljut megadásánál két Datalog-szabály volt, úgy a definíció is két lekérdezés egyesítéséből áll. A 2. sorbeli lekérdezés az unió első tagja és az első (vagy alap-) szabálynak felel meg. Azt állítja, hogy minden egyes Járatok-beli sor esetén a második és a harmadik komponens (honnan és hova) által alkotott sor megtalálható legyen az Eljut sorai között.

A 4–6. sorok közötti lekérdezés az Eljut definíciójának második (vagy rekurzív) szabályának felel meg. A két Eljut részcélt a FROM záradékban az R1 és R2 másodnév jelképezi. Az R1 első komponense megfelel a szabálybeli x-nek, és az R2 második komponense a szabálybeli y-nak. A z változónak az R1 második komponense és az R2 első komponense felel meg; figyeljük meg, hogy ezeknek a komponenseknek az egyenlőségét a 6. sor ellenőrzi le.

Végül a 7. sor írja le az egész reláció által eredményezett relációt, mely az Eljut reláció másolata. Egy lehetőség, hogy a 7. sort kicseréljük egy bonyolultabb lekérdezéssel, mint például:

```
7)  SELECT hova FROM Eljut WHERE honnan = 'DEN';
```

amely a Denverből elérhető városokat adná meg. □

10.2.2. Problémás kifejezések rekurzív SQL-ben

Az SQL-szabvány nem engedélyezi, hogy a WITH záradék után tetszőleges kölcsönösen rekurzív relációk gyűjteménye álljon. Van egy kis probléma az utóbbival, és ezért a szabvány csak a *lineáris* rekurziót támogatja. A Datalog-kifejezéseknél egy lineáris rekurzióban minden szabálynak csak egyetlen olyan részcélja lehet,

Kölcsönös rekurzió

Létezik egy gráfelméleti módszer annak ellenőrzésére, hogy két reláció vagy predikátum kölcsönösen rekurzív-e. Egy *függőségi gráf*ot kell létrehozni, melyben a csúcsok a relációknak felelnek meg (vagy predikátumoknak abban az esetben, ha Datalog-szabályokat használunk). Az A relációnak megfelelő csúcsból irányított él vezet a B relációnak megfelelő csúcsba, ha a B reláció definíciója direkt módon függ az A reláció definíciójától. Ez a Datalog esetében azt jelenti, hogy az A annak a szabálynak a törzsében található, amelynek a fejében B van. Az SQL esetében A megjelenne B definíciójában, általában a `FROM` záradékban (akár egy részkérdésben is). Ha létezik olyan irányított kör, amelyben megtalálhatók az R és S csúcsok, akkor R és S *kölcsönösen rekurzív*. A legtöbb esetben az R csúcsba hurokél vezet, azaz R rekurzívan függ önmagától.

amely kölcsönösen rekurzív a fej predikátumával. Figyeljük meg a 10.8. példában, hogy a 2. szabálynak két részcélja van azzal az `Eljut` predikátummal, amely kölcsönösen rekurzív kapcsolatban áll a fej predikátummal (hiszen a predikátum mindig kölcsönösen rekurzív viszonyban van önmagával, lásd a Kölcsönös rekurzió keretezett résznél). Azaz technikailag az ABKR visszautasíthatja a 10.11. ábrán lévő utasítás végrehajtását, miközben illeszkedik a szabványhoz.[1]

Emellett van egy fontosabb megszorítás is az SQL-rekurziókra, amelynek megsértése ahhoz vezet, hogy a lekérdezés feldolgozója nem képes értelmes módon végrehajtani a rekurziót. Egy SQL-rekurzió akkor legális, ha az R rekurzív relációdefiníciója csak olyan kölcsönösen rekurzív S relációt használ (R-et is beleértve), amelynek a használata S-ben „monoton". Egy S használat *monoton*, ha S-hez adunk egy tetszőleges sort, akkor ez azt eredményezheti, hogy R-be is bekerül egy vagy több sor, vagy hogy R változatlan marad, de soha nem eredményezheti azt, hogy R valamely sora kitörlődjön. A most következő példa azt szemlélteti, mi történhet, amikor figyelmen kívül hagyjuk a monotonitási kritériumot.

10.9. példa. Tegyük fel, hogy R egy unáris (egyattribútumos) reláció, és egyetlen sora van: a (0). Használjuk R-et EDB-relációként az alábbi Datalog-szabályra:

```
1. P(x) ← R(x) AND NOT Q(x)
2. Q(x) ← R(x) AND NOT P(x)
```

Informálisan a két szabály azt fejezi ki, hogy az x akár P-ben, akár Q-ban lehet, de mindkettőben nem. Figyeljük meg, hogy a P és a Q is kölcsönösen rekurzív.

[1] Figyeljük meg, hogy ha az 5. sorban az egyik `Eljut` használata helyett a 10.11. ábrán szereplő `Járatok`-at használnánk, akkor lineáris rekurziót kapnánk. A nem-lineáris rekurzió általában – nem feltétlenül minden esetben – átírható a neki megfelelő lineáris változatra.

Ha abból indulunk ki, hogy P és Q is üres, és alkalmazzuk egyszer a szabályokat, akkor azt kapjuk, hogy $P = \{(0)\}$ és $Q = \{(0)\}$, azaz a (0) mindkét IDB-relációba bekerül. A következő menetben, amikor a P és Q értékére alkalmazzuk a szabályokat, akkor azt kapjuk, hogy mind a kettő üres. Ez a ciklus folytatódik, ameddig akarjuk, de sosem fog a megoldáshoz konvergálni.

Valójában két „megoldás" van a Datalog-szabályokhoz:

$$\begin{array}{lll} a) & P = \{(0)\} & Q = \emptyset \\ b) & P = \emptyset & Q = \{(0)\} \end{array}$$

Nincs okunk, hogy bármelyiket is előnyben részesítsük a másikkal szemben, és az előbb bemutatott egyszerű lépéssorozat, mint a rekurzív relációk egyik kiszámítási módszere, nem konvergál egyikhez sem. Azaz arra az egyszerű kérdésre sem tudunk választ adni, hogy: „Teljesül a $P(0)$?"

```
1)  WITH
2)      RECURSIVE P(x) AS
3)              (SELECT * FROM R)
4)          EXCEPT
5)              (SELECT * FROM Q),

6)      RECURSIVE Q(x) AS
7)              (SELECT * FROM R)
8)          EXCEPT
9)              (SELECT * FROM P)

10)  SELECT * FROM P;
```

10.12. ábra. A nem monoton viselkedésű lekérdezések nem megengedettek SQL-ben

Ez a probléma nem csak a Datalogra áll fenn. A példánk két Datalog-szabálya kifejezhető rekurzív SQL-ben. A 10.12. ábra ennek egy módját szemlélteti. Ez az SQL-utasítás nem felel meg a szabvány előírásainak, ezért egyetlen ABKR sem fogja végrehajtani. □

A 10.9. példával az a probléma, hogy a 10.12. ábrán szereplő P és Q definíciója nem monoton. Vegyük például a P definícióját a 2. és 5. közötti sorokból. P a Q-tól függ, és ezzel kölcsönös rekurzív lesz ugyan, de a Q-ba történő beszúrás törlést eredményezhet R-re. Figyeljük meg, hogy ha $R = \{(0)\}$, a Q pedig üres, akkor $P = \{(0)\}$ lesz. Viszont ha a (0)-t Q-hoz adjuk, akkor kitöröljük a (0)-t P-ből. Így P definíciója nem monoton Q-ban, és a 10.12. ábrán lévő SQL-kód megsérti a szabványt.

10.10. példa. Az összesítések szintén a monotonitás elvesztéséhez vezethetnek. Tételezzük fel, hogy a következő két unáris (egy attribútummal rendelkező) P és Q relációt definiáljuk:

1. P a Q és egy R EDB-reláció egyesítése.

2. Q-nak egy sora van, amely a P elemeinek összesítését tartalmazza.

Ezeket a feltételeket kifejezhetjük egy `WITH` utasítással, azonban ez az utasítás megszegi az SQL monotonitásra vonatkozó feltételeit. A 10.13. ábra lekérdezése P értékét keresi.

```
1)  WITH
2)      RECURSIVE P(x) AS
3)                (SELECT * FROM R)
4)            UNION
5)                (SELECT * FROM Q),

6)      RECURSIVE Q(x) AS
7)            SELECT SUM(x) FROM P

8)  SELECT * FROM P;
```

10.13. ábra. Nem rétegzett lekérdezés összesítéssel, amely szabálytalan az SQL-ben

Tételezzük fel, hogy eredetileg R a (12) és a (34) sorokból áll, kezdetben P és Q üres. A 10.14. ábra foglalja össze az első hat lépésben kiszámított értékeket. Ne feledjük, hogy a relációk új értékeit az összes reláció előző lépésbeli értékei alapján számítjuk ki. Így az első lépés után P egyezni fog az R-rel, míg Q {NULL} lesz, mivel a kiszámításához P üres értékét használtuk a 7. sorból.

A második lépésben a 3–5. sorok közötti rész eredménye az

$$R \cup \{\texttt{NULL}\} = \{(12),\ (34),\ \texttt{NULL}\}$$

halmaz lesz, tehát ez lesz P új értéke, a régi érték $\{(12),\ (34)\}$ volt. Így Q új értéke a $\{(46)\}$ halmaz lesz, mivel $12 + 34 = 46$.

A harmadik lépésben $P = \{(12), (34), (46)\}$-t kapjuk a 2–5. sorok alapján. P régi értékét használjuk ($P = \{(12), (34)\}$) és a 6–7. sorok alapján $Q = \{(46)\}$ lesz újra.

Lépés	P	Q
1.	$\{(12), (34)\}$	{NULL}
2.	$\{(12), (34), \texttt{NULL}\}$	$\{(46)\}$
3.	$\{(12), (34), (46)\}$	$\{(46)\}$
4.	$\{(12), (34), (46)\}$	$\{(92)\}$
5.	$\{(12), (34), (92)\}$	$\{(92)\}$
6.	$\{(12), (34), (92)\}$	$\{(138)\}$

10.14. ábra. Iteratív kiszámítás nem monoton összesítés esetén

A negyedik lépésben P értéke ugyanaz, de Q értéke megváltozik $\{(92)\}$-re, mert $12 + 34 + 46 = 92$. Figyeljük meg, hogy Q elvesztette a (46) sort, és egy új sort kapott helyette. Tehát a (46) sor beszúrása P-be egy sor kitörlését eredményezte Q-ból (véletlenül ugyanannak a sornak a kitörlését). Ez ellenkezik a monotonitási szabállyal, amit az SQL előír, tehát az 10.13. ábra lekérdezése szabálytalan. Általában a $2i$-edik lépésben P tartalmazza a (12), (34), illetve a $(46i - 46)$ sorokat, míg Q csak a $(46i)$ sort tartalmazza. □

10.2.3. Feladatok

10.2.1. feladat. A 10.8. példa relációja:

```
Járatok(légitársaság, honnan, hova, indulás, érkezés)
```

tartalmaz az érkezési és indulási időről is információt, amit eddig nem vettünk figyelembe. Tegyük fel, hogy nemcsak az érdekel minket, hogy el tudunk-e jutni az egyik városból a másikba, hanem az is, hogy utazásunk során az átszállások is ésszerűek legyenek. Azaz, ha egynél több repülőgépet használunk, akkor az összes járatnak legalább egy órával a rá következő járat indulása előtt meg kell érkeznie. Azt is feltehetjük, hogy nincs egy napnál hosszabb utazás, így nem kell aggódnunk amiatt, hogy egy éjfél körüli érkezés esetén csak korán reggel lesz tovább csatlakozás.

a) Írjunk erre rekurziót Datalogban.

b) Írjunk erre rekurziót SQL-ben.

! **10.2.2. feladat.** A 10.8. példában a `honnan` (az angol eredetiben `frm`) az egyik attribútum neve. Miért nem használjuk inkább a `from` nevet?

10.2.3. feladat. Adott a következő reláció:

```
Folytatások(film, folytatás)
```

mely egy film azonnali folytatásait adja meg. Definiálni akarjuk még a `Sorozat` rekurzív relációt is, amelynek elemei olyan (x, y) sorok, hogy y vagy x azonnali folytatása, vagy egy folytatás folytatása stb.

a) Adjuk meg `Sorozat` definícióját rekurzív Datalog-szabályokkal.

b) Adjuk meg `Sorozat` definícióját egy rekurzív SQL-kifejezés segítségével.

c) Adjunk meg egy olyan rekurzív SQL-lekérdezést, amely azokat az (x, y) filmpárokat eredményezi, melyekre y egy nem direkt folytatása x-nek.

d) Adjunk meg egy olyan rekurzív SQL-lekérdezést, amely azokat az (x, y) filmpárkat eredményezi, amelyekre y folytatása x-nek, de nem direkt folytatása és nem egy direkt folytatás folytatása.

! *e)* Adjunk meg egy olyan rekurzív SQL-lekérdezést, amely azokat az x filmeket eredményezi, amelyeknek legalább két folytatása volt. Mindkét folytatás lehet direkt, vagy az egyik direkt, a másik pedig annak direkt folytatása.

! *f)* Adjunk meg egy rekurzív SQL-lekérdezést, amely azokat az (x, y) filmpárokat eredményezi, melyekre y folytatása x-nek, és y-nak legfeljebb egy folytatása van.

10.2.4. feladat. Adott a következő reláció:

```
Kapcsolatok(osztály, kosztály, mult)
```

amely leírja azt, hogy egy ODL-osztály hogyan viszonyul más osztályokhoz. Általában a relációnak van egy (c, d, m) sora, ha létezik egy kapcsolat a c osztályból a d osztályba. Ez a kapcsolat többértékű, ha m = `'multi'`, és egyértékű, ha m = `'single'`. A `Kapcsolatok` relációt tekinthetjük úgy, mint egy gráfot, melynek a csúcsai osztályok, és a c osztálytól a d osztályig akkor és csakis akkor létezik egy m címkéjű él, ha (c, d, m) egy él a `Kapcsolatok`-ban. Adjunk meg egy rekurzív SQL-lekérdezést, amely:

a) Azokat a (c, d) párokat adja meg, amelyekre létezik egy útvonal c-től d-ig.

b) azokat a (c, d) párokat adja meg, amelyekre létezik egy útvonal c-től d-ig, amely végig egyértékű kapcsolatokból áll.

! *c)* Azokat a (c, d) párokat adja meg, amelyekre létezik egy útvonal c-től d-ig, amelyben legalább az egyik él többértékű kapcsolatot jelképez.

d) Azokat a (c, d) párokat adja meg, amelyekre létezik egy útvonal c-től d-ig, de nem létezik köztük olyan útvonal, amely végig egyértékű kapcsolatokból áll.

! *e)* Azokat a (c, d) párokat adja meg, amelyekre létezik egy útvonal c-től d-ig, amely váltakozóan egyértékű és többértékű kapcsolatokat jelképező élekből áll.

f) Azokat a (c, d) párokat adja meg, amelyekre létezik egy útvonal c-től d-ig és d-től c-ig, amely végig egyértékű kapcsolatokból áll.

10.3. Az objektumrelációs modell

A relációs modell és az objektumorientált modell (amelyet az ODL jellemez) két fontos választási lehetőség a sok közül, amelyekre egy ABKR épülhet. Hosszú időn keresztül a relációs modell volt a meghatározó a forgalomban lévő ABKR-ek világában. Az objektumorientált ABKR-ek megpróbálkoztak betörni a piacra az 1990-es évek folyamán, de nem nyertek jelentős piaci részesedést a relációs

adatbázisok forgalmából. Ezzel szemben a relációs rendszerek forgalmazói elkezdték beépíteni az ODL-elveknek és más objektumorientált adatbázisokra tett javaslatoknak a nagy részét a saját rendszerükbe. Ennek eredményeképpen az eddig „relációsnak" nevezett ABKR-termékek nagy részét mára „objektumrelációsnak" nevezik.

Ebben a részben kiterjesztjük az absztrakt relációs modellt, hogy magába foglalhasson néhány fontos objektum relációs alapelvet is. Az ezt követő rész pedig az SQL objektumrelációkkal történő kiterjesztésével foglalkozik. A 10.3.1. alfejezetben vezetjük be az objektumrelációs fogalmakat, majd a korai megvalósításairól, a beágyazott relációkról ejtünk szót a 10.3.2. alfejezetnél. A 10.3.3. alfejezetben az ODL-típusú hivatkozások objektumrelációkra vonatkozó alkalmazását tárgyaljuk. Végül a 10.3.4. alfejezetben összehasonlítjuk az objektumrelációs és a tisztán objektumorientált megközelítéseket.

10.3.1. A relációktól az objektumrelációkig

Miközben a reláció megmaradt alapvető fogalomnak, a relációs modellt *objektumrelációs modellé* terjesztették ki a most következő lehetőségek beépítésével:

1. *Attribútumokhoz szerkesztett strukturált típusok.* Az attribútumokhoz nem csak atomi típusok rendelhetők. Az objektumrelációs rendszerek egy ODL-típusú típusrendszert támogatnak: atomi típusokból felépített típusok, illetve típuskonstruktorokkal (például a szerkezetekhez, a halmazokhoz és a multihalmazokhoz). Különösen fontos egy szerkezetekből álló multihalmaz szerkezetű típus, amely alapvetően egy reláció. Azaz egy sor egy komponensének értéke lehet egy egész reláció, ezt nevezzük beágyazott relációnak.

2. *Metódusok.* Ugyanolyanok, mint a metódusok az ODL-ben, vagy bármely más objektumorientált programozási környezetben.

3. *Sorok azonosítói.* Az objektumrelációs rendszerekben a sorok töltik be az objektum szerepét. Ezért is vált hasznossá, hogy minden egyes sor egyedi azonosítót kapjon, amely megkülönbözteti a sort a többitől még akkor is, ha kettőnek ugyanolyan komponensértékei lennének. Ez az azonosító, csakúgy mint az objektumazonosító ODL-ben, rejtett a felhasználó elől, habár van néhány eset az objektumrelációs rendszerekben, amikor a felhasználó láthatja a sorok azonosítóját.

4. *Hivatkozások.* Miközben a tiszta relációs modellnek nincs jelölése a sorokra vonatkozó hivatkozásra vagy mutatóra, az objektumrelációs rendszerekben ezt többféleképpen is megtehetjük.

A következő alfejezetekben az objektumrelációs rendszerek ezen hozzáadott lehetőségeit részleteiben áttekintjük és szemléltetjük is.

10.3.2. Beágyazott relációk

A *beágyazott relációs modell*ben a reláció attribútumaira nemcsak atomi típusokat engedünk meg, hanem a típus lehet akár egy relációséma is. Összefoglalóan itt van az attribútumok típusának és a relációk (séma)típusának egy alkalmas, rekurzív definíciója:

KIINDULÁS: Egy attribútum típusa lehet egy atomi típus (egész szám, valós szám, karaktersorozat és így tovább).

INDUKCIÓ: Egy relációnak a típusa lehet bármilyen *séma*, amely egy vagy több attribútumnevet tartalmaz, és az attribútumokhoz tartozó megengedett típusokat ad meg. Sőt egy séma is lehet egy attribútumnak a típusa.

A most következőkben általában el fogjuk hagyni az atomi típusokat azokon a helyeken, ahol ez nem okoz félreértést. Egy attribútum, amely séma, az attribútum nevével és a sémához tartozó attribútumoknak a zárójelezett listájával van jelölve. Mivel ez utóbbi attribútumok maguk is lehetnek struktúrák, ezért a zárójelezés tetszőleges mélységig be lehet ágyazva.

10.11. példa. Tervezzünk egy olyan beágyazott relációsémát a filmszínészekhez, amely tartalmaz egy `filmek` attribútumot, mint relációt, amely az adott színész által játszott filmeket fogja ábrázolni. A `filmek` attribútum sémája tartalmazza a film címét, gyártási évét és hosszát. A `Színészek` reláció sémája tartalmazza a színész nevét, címét és születési dátumát, illetve a `filmek`-ben lévő információt. Ezeken felül a `cím` attribútum is relációtípusú lesz: az `utca` és `város` attribútumokkal. Ez utóbbi relációban egy színész több címét is tárolhatjuk majd. A `Színészek` sémája tehát a következő:

```
Színészek(név, cím(utca, város), születésiDátum,
        filmek(filmcím, év, hossz))
```

A `Színészek` beágyazott reláció egy lehetséges példája a 10.15. ábrán található. Ebben a konkrét relációban két sort találhatunk: egyet Carrie Fisherhez, egyet pedig Mark Hamillhez. A komponensek értékeit rövidítettük helytakarékossági okokból, illetve a sorok szaggatott vonallal történő szétválasztását is csak kényelmi szempontok miatt tettük meg, és így nincs semmilyen jelölési jelentősége.

A Carrie Fisherhez tartozó sorban láthatjuk az ő nevét (atomi értéket), amelyet egy reláció követ a lakcím komponens értékével. Ennek a relációnak két attribútuma van, `utca` és `város`, és két sora is van a két házának megfelelően. Ezután következik a születési dátum, amely szintén egy atomi érték. Végül következik a `filmek` attribútumra vonatkozó komponens. Ezen attribútumnak, mint típusnak, relációsémája van a film címével, gyártási évével és hosszával. A Carrie Fisher sorához tartozó `filmek` komponens relációja az ő három legismertebb filmjét tartalmazza.

A második sor, Mark Hamillhez ugyanilyen komponensekkel rendelkezik. Az ő `cím` relációja csak egy sorral rendelkezik, mert a mi fikcióink szerint neki

név	*cím*		*születési-Dátum*	*filmek*		
Fisher	*utca*	*város*	9/9/99	*filmcím*	*év*	*hossz*
	Maple	H'wood		Csillagok háborúja	1977	124
	Locust	Malibu		A birodalom visszavág	1980	127
				Jedi visszatér	1983	133
Hamill	*utca*	*város*	8/8/88	*filmcím*	*év*	*hossz*
	Oak	B'wood		Csillagok háborúja	1977	124
				A birodalom visszavág	1980	127
				Jedi visszatér	1983	133

10.15. ábra. Egy beágyazott reláció a színészekre és filmjeikre

csak egy háza van. A `filmek` relációja pedig ugyanolyan, mint Carrie Fisheré, hiszen a legismertebb filmjeik véletlenül egybeesnek. Megjegyezzük, hogy ez a két reláció két különböző sorkomponens. Ezek a komponensek ugyanúgy megegyezhetnek, mint ahogyan két különböző, egész értékű komponens értéke is megegyezhet (például 124). □

10.3.3. Hivatkozások

Az a tény, hogy a *Csillagok háborújá*hoz hasonlóan néhány film több relációban (azaz a beágyazott `Színészek` reláció `filmek` attribútumának értékeiben) is előfordulhat, redundanciához fog vezetni. Valójában a 10.11. példa sémájának megfelelő beágyazott reláció nem volt BCNF-ben. Azonban a `Színészek` reláció felbontása nem fogja megszüntetni a redundanciát. Ehelyett inkább azt kell elintéznünk, hogy egy film csak egyszer szerepeljen az összes `filmek` reláció összes sorában.

A probléma megoldása az, hogy az objektumrelációknak rendelkezni kell azzal a lehetőséggel, hogy inkább hivatkozhassunk egy t sorból egy s sorra, minthogy az s egyben benne legyen a t-ben. Ezért a modellünkhöz kell adni egy újabb indukciós szabályt: egy attribútum típusa lehet egy adott sémával rendelkező sorra történő hivatkozás vagy adott sémával rendelkező sorokra történő hivatkozási halmaz is.

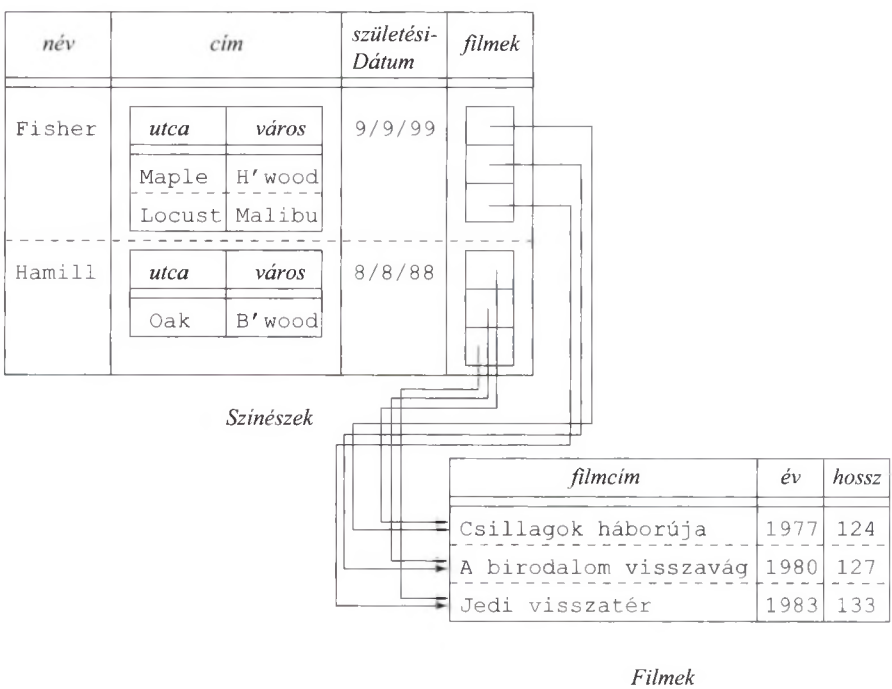
Ha egy A attribútum típusa egy adott R nevű relációsémájú sorra vonatkozó hivatkozás, akkor az A attribútum $A(*R)$ alakban jelenik meg egy sémában. Figyeljük meg, hogy ez az eset analóg azzal, mintha A egy R típusú ODL-kapcsolat lenne, azaz egy egyszerű R típusú objektumhoz kapcsolódik. Hasonlóan, ha egy A attribútum típusa egy R séma soraira vonatkozó hivatkozási halmaz, akkor

az A attribútum $A(\{*R\})$ alakban jelenik meg egy sémában. Ez az eset hasonlít egy olyan A ODL-kapcsolathoz, amelynek típusa `Set<`R`>`.

10.12. példa. Az egyik megfelelő módszer a 10.15. ábrán lévő redundancia helyrehozására az, ha két relációt használunk: az egyiket a színészekhez, a másikat a filmekhez. Ebben a példában egy `Filmek` relációt érdemes használnunk, amely egy megszokott reláció lesz, amelynek ugyanaz lesz a sémája, mint a 10.11. példa `filmek` attribútumának. A `Színészek` új sémája nagyon hasonló az előző példa `Színészek` beágyazott relációjához, de a `filmek` attribútum típusa egy, a `Filmek` soraira vonatkozó hivatkozási halmaz típus lesz. Így a két reláció sémája a következő lesz:

```
Filmek(filmcím, év, hossz)
Színészek(név, cím(utca, város), születésiDátum,
        filmek({*Filmek}))
```

A 10.15. ábrán látható adat új sémához alakított változatát a 10.16. ábra szemlélteti. Figyeljük meg, hogy mivel minden egyes filmnek csak egyetlen sora lesz, mégis több hivatkozás lehet ezekre. Így a 10.11. példa sémájában lévő redundanciát kiküszöböltük. □



10.16. ábra. Egy attribútum értékeinek hivatkozási halmaza

10.3.4. Az objektumorientált és az objektumrelációs változatok összehasonlítása

Az ODL-lel tipizált objektumorientált adatmodell és a most áttekintett objektumrelációs adatmodell nagyon hasonlóak. Az összehasonlítás szembeötlő pontjai a következők:

Objektumok és sorok

Egy objektum értéke valójában egy komponensekkel rendelkező struktúra az attribútumaira és kapcsolataira nézve. Ugyan nincs meghatározva az ODL-szabványban, hogyan kell ábrázolni a kapcsolatokat, de feltehetjük, hogy egy objektum a vele kapcsolatban lévő objektumokkal valamilyen hivatkozási gyűjteményen keresztül van összekapcsolva. Egy sor olyan lesz, mint egy struktúra, de a megszokott relációs modellekben csak az attribútumaihoz tartozhatnak komponensek. A kapcsolatokat egy másik relációban szereplő sorral ábrázolhatjuk, mint ahogyan a 4.5.2. alfejezetben is javasoltuk. Az objektumrelációs modell viszont azzal, hogy megengedték, hogy a sorok egy komponense hivatkozások halmaza is lehessen, egyúttal megengedi a kapcsolatok direkt beépítését az olyan sorokba, amelyek egy „objektumot" vagy entitást ábrázolnak.

Metódusok

Nem tárgyaltuk a relációséma részeként definiált metódusok használatát. A gyakorlatban viszont az SQL-99 és a megvalósított objektumrelációs elvek egy osztályhoz vagy egy típushoz egy ODL-éhez hasonló metódusdeklarációt és definíciót tesznek lehetővé.

Típusrendszerek

Az objektumorientált és az objektumrelációs típusrendszerek nagyon hasonlóak. Mind a kettő elemi (atomi) típusokra épül, és új típusokat lehet velük létrehozni struktúra-, illetve gyűjteménytípus-konstruktorok segítségével. A gyűjteménytípusok sokszínűek, de mindegyik változatuk tartalmaz multihalmazt vagy halmazt. Sőt a struktúrákból képzett halmaz- (vagy multihalmaz-) típus speciális szerepet kap mindkét modellben. Ez pedig ODL-ben az osztályok típusát jelenti. Az objektumrelációs modellben pedig a relációk típusát adja.

Hivatkozások és objektumazonosítók

A tisztán objektumorientált rendszerek objektumazonosítókat használnak, amelyek teljesen rejtve vannak a felhasználók elől, azaz így le se kérdezhetők és meg se jeleníthetők. Az objektumrelációs modellben a hivatkozások is lehetnek egy típus részei, és ezért néhány esetben elképzelhető, hogy a felhasználó látni fogja ezeket az értékeket, sőt meg is jegyezheti ezeket a későbbi használat céljából.

A nézőpontunktól függően úgy tekinthetünk ezekre a helyzetekre, mint valamire, ami egy súlyos rendszerhibától egészen egy zseniális ötleten keresztül bármi lehet, de gyakorlatban általában nagyon kicsi a különbség.

Kompatibilitás visszafelé

A kicsiny eltérés a két modell alapvető környezeteiben érdekessé teszi azt a kérdést, hogy az objektumrelációs rendszerek miért váltak dominánssá a tisztán objektumorientált rendszerekkel szemben a piacon. A válasz szerintünk a következő lesz. Mivel a relációs ABKR-ek átalakultak objektumrelációs ABKR-ekké, ezért a forgalmazók nagy figyelmet fordítottak a visszafele kompatibilitásra, azaz arra, hogy az újabb verziójú rendszer továbbra is futtassa a régi kódot, és elfogadja ugyanazt a sémát. A felhasználónak ne kelljen alkalmazkodnia az objektumorientált környezetekhez. Másrészt egy tisztán objektumorientált rendszerre való áttéréskor szükség lehet telepítésekre az átíráshoz, illetve alapos újraszervezéshez. Ezért bármilyen versenyképes előnyei is voltak az objektumorientált adatbázisrendszereknek, mindez nem volt elegendő ahhoz, hogy sokan változtassanak.

10.3.5. Feladatok

10.3.1. feladat. A beágyazott relációknál használt jelölés felhasználásával és a relációkhoz tartozó hivatkozások segítségével adjunk egy vagy több relációsémát, amely az alábbiakban szereplő információt tartalmazza. Minden esetben fontoljuk meg, hogy milyen attribútumokat tartalmazhat a reláció, de ne nagyon térjen el az általunk használt filmes példa attribútumaitól. Azt is mutassuk meg, ha a séma redundanciát mutat. Ha redundancia van, akkor mondjuk meg azt is, hogyan lehetne ezt kiküszöbölni.

a) A filmeket a filmhez tartozó színészekkel együtt a filmek és színészek összes megszokott attribútumának szerepeltetésével.

! *b)* A stúdiókat az általuk készített filmekkel és a filmekben szereplő színészekkel együtt a stúdiók, filmek és színészek összes attribútumának szerepeltetésével.

c) A filmeket a stúdiójukkal és a színészeikkel együtt a filmek, stúdiók és színészek összes megszokott attribútumának szerepeltetésével.

10.3.2. feladat. Ábrázoljuk a 4.1.1. feladat banki adatait az ebben az alfejezetben bemutatott objektumrelációs modellben. Bizonyosodjunk meg a következőkről: adott ügyfél sorából legyen könnyű számláinak elérése, és fordítva, egy számla sorából legyen könnyű a számlát birtokló ügyfél (ügyfelek) elérése. Próbáljuk meg elkerülni a redundanciát!

! **10.3.3. feladat.** Ha a 10.3.2. feladatban szereplő adatokat úgy módosítanánk, hogy egy számla csak egy ügyfélhez tartozhat, mint a 4.1.2 (a) feladatban, akkor hogyan lehetne egyszerűbben megválaszolni a 10.3.2. feladatot?

! **10.3.4. feladat.** A 4.1.3. feladatban szereplő játékosokat, csapatokat és szurkolókat fogalmazzuk át az objektumrelációs modellnek megfelelő formába.

! **10.3.5. feladat.** A 4.1.6. feladatban szereplő leszármazási rendet fogalmazzuk át az objektumrelációs modellnek megfelelő formába.

10.4. Felhasználó által definiált típusok SQL-ben

Most megnézzük, hogyan valósítja meg az SQL-99 a 10.3. alfejezetben bemutatott objektumrelációs lehetőségek nagy részét. SQL-ben a központi kiterjesztés a *felhasználói típus* vagy röviden UDT (user-defined type), amely a relációs modellt objektumrelációssá teszi. Az UDT kétféleképpen használható, lehet:

1. egy tábla típusa vagy
2. valamilyen táblában szereplő attribútum típusa.

10.4.1. Típusdefiníció SQL-ben

Az SQL-99 szabvány több módszert is lehetővé tesz a programozó részére az UDT megadására. A legegyszerűbb egy létező típusnak az átnevezése.

`CREATE TYPE` T `AS` <primitív típus>;

Ez egy olyan primitív típust nevez át, mit például az `INTEGER`. Az lehet a célunk ezzel, hogy megelőzzük az olyan hibákat, amelyek a logikailag össze nem hasonlítható, illetve fel nem cserélhető értékek közti típuskényszerítésnél merülnek fel (még ha ugyanaz is a két érték primitív típusa). Ezt a célt teszi világosabbá a következő példa.

10.13. példa. A bemutatott filmes példánkban több `INTEGER` típusú attribútum is van. Ilyen például a filmek hossza (`Filmek.hossz`) vagy a gyártásirányítók producerazonosítója (`GyártásIrányító.producerAzon`). A `producerAzon` értékének és a `Stúdió azonosító` értékének összehasonlítása értelmes lesz, és így vehetnénk a két attribútum egy tetszőleges értékét, és ezt az értéket letárolhatnánk egy sor másik attribútumának értékeként. Viszont a film hosszának a gyártásirányító azonosító számával történő összehasonlítása nem lesz értelmezhető, vagy másként a `Filmek` táblából származó `hossz` érték nem tárolható le a `GyártásIrányító` egy sorának `producerAzon` attribútuma értékeként.

Ha típusokat hozunk létre, akkor meghatározhatunk egy `AzonosítóTípus` típust a reláció deklarációjában szereplő `INTEGER` helyett az `azonosító` és `producerAzon` értékekre. Sőt megadhatunk a `Filmek` deklarációjában a `hossz` attribútumra egy `HosszTípus` típust is. Ez a következő lesz:

```
CREATE TYPE AzonosítóTípus AS INTEGER;
CREATE TYPE HosszTípus AS INTEGER;
```

Egy objektumrelációs ABKR ekkor nem értelmezi a két típus egymással történő összehasonlításának kísérletét, sőt még azt sem, hogy az egyik típus értékét a másik értékeként használjuk. □

SQL-ben az UDT megadásának egy hatékonyabb formája néhány különbségtől eltekintve ugyanolyan, mint az ODL-osztályok megadási módja volt. Először is egy felhasználói típussal adott reláció kulcsának deklarálása a tábla definíciójának a része, nem pedig a típusé. Így SQL-ben több reláció deklarálható ugyanazzal a (felhasználói) típussal, de ezeknek a relációknak a kulcsai és más megszorításai különböznek. Másodszor, a kapcsolatot nem tulajdonságként kezeli az SQL. Egy kapcsolatot a 4.10.5. alfejezetben tárgyalt módon egy külön relációban lehet leírni. Másik lehetőség a hivatkozások használata, amelyet majd a 10.4.5. alfejezetben látunk. Az UDT definíciójának alakja az alábbi:

CREATE TYPE T AS (<attribútumdeklarációk>);

10.14. példa. A 10.17. ábrán látható két típus: a `CímTípus` és a `SzínészTípus`. Egy `CímTípus` típusú sor két komponensből áll, attribútumai pedig `utca`, illetve `város`. Ezen komponensek 50, illetve 20 hosszú karakterláncok. Egy `SzínészTípus` típusú sornak szintén két összetevője van. Az első a `név` attribútum, amely 30 hosszú karakterlánc típusú. A második attribútum a `cím`, ennek típusa önmaga a `CímTípus` (UDT), vagyis egy `utca` és `város` komponensekből álló sor. □

```
CREATE TYPE CímTípus AS (
    utca    CHAR(50),
    város   CHAR(20)
);

CREATE TYPE SzínészTípus AS (
    név     CHAR(30),
    cím     CímTípus
);
```

10.17. ábra. Két típus definíciója

10.4.2. Metódusok megadása a felhasználói típusban

Egy metódus deklarációja hasonló módon történik, mint egy függvénynek a bevezetése PSM-ben (lásd 9.4.1. alfejezet). A metódusok nem analógok a PSM-eljárásokkal, ugyanis minden metódus valamilyen típusú értékkel kell visszatérjen. Miközben a PSM-ben a függvény deklarációja és definíciója együtt szerepelnek, addig a metódusnak kell egy deklaráció, amelyet a `CREATE TYPE` utasításban egy attribútumokat tartalmazó zárójelezett lista kell kövessen, illetve egy elkülönített definíció is kell a `CREATE METHOD` utasításban. A metódus kódja

nem feltétlenül PSM-ben íródott, de elképzelhető ez is. Például a metódus törzse lehet Java-kód, amely JDBC-szintaxist használ az adatbázis-hozzáférésre.

A metódus deklarációja úgy néz ki, mint egy PSM-függvény deklarációja, csak a `CREATE FUNCTION` helyett a `METHOD` kulcsszót használjuk. Bár az SQL-ben írt metódusoknak általában nincs argumentuma, mégis alkalmazhatók sorokra csakúgy, mint az ODL-metódusok objektumokra. Ha szükséges, az eljárás definíciójában a `SELF`-fel hivatkozhatunk erre a sorra.

10.15. példa. Egészítsük ki a 10.17. ábrán lévő `CímTípus` típus definícióját egy `házSzám` eljárással, amely a ház címének erre szánt részét szedi ki az `utca` komponensből. Ha például az `utca` komponens értéke `'123 Maple St.'`, akkor a `házSzám` `'123'`-mal tér vissza. A `házSzám` valódi működési módja nem látható a deklarációban, mivel a részletek a definícióban szerepelnek. A módosított típusdefiníció a 10.18. ábrán található.

```
CREATE TYPE CímTípus AS (
    utca      CHAR(50),
    város     CHAR(20)
    )
    METHOD házSzám() RETURNS CHAR(10);
```

10.18. ábra. Egy metódusdeklaráció hozzáadása egy UDT-hez

Láthatjuk, hogy a `METHOD` kulcsszó után az eljárás neve és argumentumainak és típusainak listája (zárójelezett) szerepel. Jelen esetben nincs argumentum, de a zárójelek ilyenkor is kellenek. Ha lennének argumentumok, akkor a típusaik követnék azokat, mint például az `(a INT, b CHAR(5))`. □

10.4.3. Metódusdefiníciók

Külön definiáljuk a metódust. Egy metódus definíciójának egyszerű alakja a következőkből tevődik össze:

`CREATE METHOD` <a metódus neve, az argumentumok és a visszatérési érték>
`FOR` <UDT neve>
<a metódus törzse>

Azaz a `FOR` záradékban szerepel az UDT, amelyre a metódust megadjuk. A metódus definíciója nem kell hogy a hozzá tartozó típus definíciójával egy helyen vagy annak részeként szerepeljen.

10.16. példa. A 10.15. példában szereplő `házSzám` eljárást definiálhatjuk a 10.19. ábrán lévő módon. A metódus törzsét kihagytuk, mivel a tervezett `cím` karakterlánc darabolásának megvalósítása nem triviális még egy általános befogadó nyelv használata esetén sem. □

```
CREATE METHOD házSzám() RETURNS CHAR(10)
FOR CímTípus
BEGIN
    ...
END;
```

10.19. ábra. Egy metódusdefiníció

10.4.4. Relációk deklarálása UDT felhasználásával

Miután deklaráltunk egy típust, létrehozhatunk egy vagy több relációt, amelyek a típusnak megfelelő típusú sorokat tartalmaznak. Egy ilyen relációt a 2.3.3. alfejezetben leírtakhoz hasonlóan hozhatunk létre azzal a különbséggel, hogy az attribútumok és típusuk felsorolása helyett az `OF` záradékot és az UDT-típus nevét használjuk. Azaz, a `CREATE TABLE` utasítás egy választható alakja UDT felhasználásával a következő:

`CREATE TABLE` <táblanév> `OF` <UDT neve>
(<elemek listája>);

A zárójelezett elemek listája tartalmazhat: kulcsot, idegen kulcsokat és a soralapú megszorításokat. Megjegyezzük viszont, hogy ezek az elemek a megadott táblára lesznek érvényesek, nem pedig az UDT-re. Így több táblának a sortípusa is lehet ugyanaz az UDT, miközben ezeknek a tábláknak lehetnek különböző megszorításaik, sőt akár különböző kulcsaik is. Ha nincsenek se megszorítások, se kulcsok a táblához, akkor a zárójelezett rész elhagyható.

10.17. példa. Deklarálhatjuk a `FilmSzínész` nevű relációt úgy, hogy `SzínészTípus` típusú sorokat tartalmazzon:

```
CREATE TABLE FilmSzínész OF SzínészTípus (
    PRIMARY KEY (név)
);
```

Az utasítás azt eredményezi, hogy a `FilmSzínész` táblának két attribútuma lesz, a `név` és a `cím`. Az első attribútum, a `név`, egy szokványos karakterlánc, míg a második – `cím` – típusa maga is egy UDT-típus, nevezetesen `CímTípus` típusú. A `név` attribútum a reláció kulcsa lesz, ezért nem lehet két azonos névvel rendelkező sora. □

10.4.5. Hivatkozások

Az objektumorientált nyelvek objektumazonosági tulajdonságát az SQL a *hivatkozás* (reference) fogalmának bevezetésével oldja meg. Az UDT-típusú tábláknak lehet egy *hivatkozási oszlop*a, amely az azonosítást szolgálja. Ez az oszlop lehet például a tábla elsődleges kulcsa (ha egyáltalán létezik), vagy egy

olyan oszlop, amelynek az értékeit az ABKR generálja, és egyediségüket megőrzi. Amíg nem láttuk, hogyan kell a hivatkozásokat használni, addig gondot okozhat egy hivatkozási oszlop definíciója.

Ha hivatkozni akarunk egy hivatkozási oszloppal rendelkező tábla soraira, akkor egy attribútumtípusa lehet egy hivatkozás egy másik típusra. Ha T egy UDT, akkor `REF`(T) egy T típusú sorra történő hivatkozásnak a típusa. Sőt a hivatkozásnak lehet *hatásköre* is, mégpedig azon reláció neve, amelynek soraira hivatkozik. Azaz egy A attribútum, amelynek értékei az R (R egy T UDT-típusú tábla) reláció soraira hivatkoznak, a következőképpen írható le:

`A REF(`T`) SCOPE` R

Ha nincs hatáskör definiálva, akkor minden T típusú relációra hivatkozhatunk.

```
CREATE TYPE SzínészTípus AS (
    név       CHAR(30),
    cím       CímTípus,
    legjobbFilm REF(FilmTípus) SCOPE Filmek
);
```

10.20. ábra. Egy „legjobb film" hivatkozás hozzáadása a `SzínészTípus` típushoz

10.18. példa. A `FilmSzínész` relációban rögzítsük az összes színészre az adott színész legjobb filmjeit. Tegyük fel, hogy meghatároztunk egy megfelelő `Filmek` relációt, illetve hogy ennek az UDT-típusa a `FilmTípus`. A későbbiekben a 10.21. ábrán definiáljuk a `FilmTípus`-t és a `Filmek`-et is. A `SzínészTípus` most következő, új definíciója már tartalmazza a `legjobbFilm` attribútumot, amely egy filmre vonatkozó hivatkozást takar. Most pedig, ha a `FilmSzínész` relációhoz a 10.20. ábra UDT-definícióját használjuk, akkor minden színészsornak van egy komponense, amely a színész legjobb filmjéhez tartozó `Filmek` relációbeli sorra hivatkozik. □

10.4.6. Objektumazonosítók létrehozása a táblákhoz

A tábla egy sorára való hivatkozáshoz, mint például a 10.18. példa `Filmek` relációjánál, a tábla sorainak rendelkeznie kell egy „objektumazonosítóval". Az ilyen táblákat *hivatkozhatónak (referenceable)* nevezzük. Egy olyan `CREATE TABLE` utasítást, amelyben a tábla típusa egy UDT-típus (mint a 10.4.4. alfejezetben), kiegészíthetjük az alábbi alakú záradékkal:

`REF IS` <attribútumnév> <hogyan generált>

Az attribútumnév annak az oszlopnak a neve, amely a sorokra vonatkozó objektumazonosítást segíti elő. A „hogyan generált" záradék általában a következők egyike:

1. SYSTEM GENERATED, amelynek a jelentése, hogy az ABKR biztosítja minden sorra, hogy az értékek egyediek legyenek a megfelelő oszlopra nézve.

2. DERIVED, amelynek a jelentése, hogy az ABKR a reláció elsődleges kulcsát használja arra, hogy az értékek egyediek legyenek a megfelelő oszlopra nézve.

10.19. példa. A 10.21. ábrán láthatjuk, hogyan definiálhatjuk a FilmTípus UDT-t, illetve a Filmek relációt úgy, hogy hivatkozható legyen. Az UDT-t az 1–4. sorok írják le. Majd az 5–7. sorok felhasználják ezt a típust a Filmek reláció definíciójánál. Megjegyezzük, hogy a 7. sor miatt a filmcím és az év együtt lesz a Filmek reláció kulcsa.

```
1)  CREATE TYPE FilmTípus AS (
2)      filmcím CHAR(30),
3)      év      INTEGER,
4)      műfaj   CHAR(10)
    );

5)  CREATE TABLE Filmek OF FilmTípus (
6)      REF IS filmAzon SYSTEM GENERATED,
7)      PRIMARY KEY (filmcím, év)
    );
```

10.21. ábra. Egy hivatkozható tábla létrehozása

A 6. sor azt is kifejezi, hogy az ABKR lesz felelős a filmAzon értékeinek generálásáért minden olyan esetben, mikor új sor szúródik be a Filmek táblába. Ha SYSTEM GENERATED helyett DERIVED kulcsszót írunk, akkor egy új sor filmAzon értékét az elsődleges kulcs filmcím és év attribútumai alapján végzett számításokkal képzi a rendszer ezen sor alapján. □

10.20. példa. Most pedig nézzük meg, hogy a filmek és szereplők közötti sok-sok kapcsolatot hogyan ábrázolhatnánk hivatkozások segítségével. Korábban már ábrázoltuk ezt a kapcsolatot egy olyan reláció segítségével, mint a SzerepelBenne, amely a Filmek és FilmSzínész kulcsaiból álló sorokat tartalmaz. Egy másik lehetőség, hogy a SzerepelBenne definíciójában szerepeljenek a hivatkozások az előbbi két reláció soraira.

Először a FilmSzínész relációt újra kell definiálnunk, hogy hivatkozható legyen, azaz:

```
CREATE TABLE FilmSzínész OF SzínészTípus (
    REF IS színészAzon SYSTEM GENERATED,
    PRIMARY KEY (név)
);
```

Ekkor a SzerepelBenne reláció megfogalmazható két attribútummal, amelyek hivatkozások. Az egyik a filmsorokra, a másik pedig a színészsorokra hivatkozik. Az alábbi egy direkt definíciója ennek a relációnak:

```
CREATE TABLE SzerepelBenne (
    színész REF(SzínészTípus) SCOPE FilmSzínész,
    film    REF(FilmTípus) SCOPE Filmek
);
```

A fentiek alapján akár megfogalmazhatnánk egy UDT-t is, majd a SzerepelBenne relációt ilyen típusú relációként deklarálhatnánk. □

10.4.7. Feladatok

10.4.1. feladat. Az általunk használt filmpéldára válasszunk az összes reláció minden egyes attribútumához egy típusnevet. Válasszuk ugyanazt az UDT-típust az attribútumokra, ha az értékeik ésszerűen összehasonlíthatók vagy felcserélhetők, illetve adjunk különböző UDT-típust nekik, ha az értékeik nem összehasonlíthatóak, illetve nem felcserélhetők.

10.4.2. feladat. Deklaráljuk az alábbi típusoknak megfelelő típusokat:

a) NévTípus, amely tartalmazza a családi nevet, utónevet és valamilyen megszólítást.

b) SzemélyTípus, amely tartalmazza a személy nevét, valamint a hivatkozásokat azokra a személyekre, akik az ő apjuk és anyjuk. A deklarációban használjuk az *a)* részben definiált típust.

c) HázasságTípus, amely tartalmazza a házasság dátumát, valamint a férjre és a feleségre történő hivatkozásokat.

10.4.3. feladat. Tervezzük át a 2.4.1. feladatban szereplő termékek adatbázissémáját. Használjunk típusokat és hivatkozás típusú attribútumokat ott, ahol helyénvaló. A PC, Laptop és Nyomtató relációkban a modell attribútum hivatkozás legyen a modellnek megfelelő Termék sorra.

! **10.4.4. feladat.** A 10.4.3. feladatban azt indítványoztuk, hogy a modellszámok helyett a PC, Laptop és Nyomtató táblákban a megfelelő Termék sorokra történő hivatkozások szerepeljenek. Megtehetjük-e azt, hogy a Termék tábla modell attribútumából olyan hivatkozást készítünk, amely ennek a relációnak az adott típusú terméket reprezentáló sorára mutat? Ha igen, akkor miért, ha nem, akkor miért nem?

10.4.5. feladat. Tervezzük át a 2.4.3. feladatban szereplő csatahajók adatbázissémáját. Használjunk típusokat és hivatkozás típusú attribútumokat ott, ahol helyénvaló. Keressük meg a sok-egy kapcsolatokat, és próbáljuk meg azokat hivatkozás típusú attribútumokkal reprezentálni.

10.5. Műveletek objektumrelációs adatokkal

Az előző fejezetek során tárgyalt összes SQL-utasítást vagy egy UDT által definiált táblára, vagy olyan táblára fogalmaztuk meg, amely néhány attribútumának típusa UDT volt. Használhatunk viszont még néhány teljesen új műveletet is, mint például a hivatkozáskövetést. Néhány jól ismert operátor azonban új szintaxist von maga után, főként azok, amelyek UDT-típussal rendelkező oszlopokhoz férnek hozzá vagy módosítják azokat.

10.5.1. Hivatkozások követése

Legyen x egy `REF(`T`)` típusú érték. Ekkor az x egy T típusú t sorra hivatkozik. Megkaphatjuk magát a t sort vagy a t sor egy komponensét az alábbiakkal:

1. A `->` operátornak ugyanaz a jelentése, mint az ugyanilyen C-beli operátornak. Azaz, ha az x egy t sor referenciája, és a a t attribútuma, akkor `x->a` lesz az a attribútum értéke a t sorban.

2. A `DEREF` operátor egy hivatkozásra alkalmazható, és a hivatkozott sort szolgáltatja.

10.21. példa. Használjuk a 10.20. példában használt `SzerepelBenne` relációt, hogy megtaláljuk Brad Pitt filmjeit. A séma az alábbi volt:

```
SzerepelBenne(színész, film)
```

ahol a `színész` és a `film` a `FilmSzínész`, illetve `Filmek` soraira hivatkozott. A lehetséges lekérdezés:

```
1)  SELECT DEREF(film)
2)  FROM SzerepelBenne
3)  WHERE színész->név = 'Brad Pitt';
```

A 3. sorban szereplő `színész->név` kifejezés visszaadja a `FilmSzínész`-beli azon `név` komponenst, amely hivatkozik valamely `SzerepelBenne`-ben lévő sor `színész` komponensére. Azaz a `WHERE` záradék azonosítja azokat a `SzerepelBenne` sorokat, amelyeknek a `színész` komponense a Brad Pitthez tartozó `FilmSzínész`-beli sorokra hivatkozik. Ezek után az 1. sor a `film` komponens által hivatkozott sorokból előállítja a `film` sorokat. Mind a három filmattribútum szerepelni fog a kiíratásban, azaz: a `filmcím`, az `év` és a `műfaj` attribútumok.

Figyeljük meg, hogy az első sort lecserélhetnénk az alábbira:

```
1)  SELECT  film
```

Ha viszont ezt tesszük, akkor egy, a felhasználó számára értelmezhetetlen rendszer által generált listát kaphatunk, ahol a listatagok a sorok belső, egyedi azonosítójaként szolgálnak. □

10.5.2. UDT-vel rendelkező sorok attribútumainak lekérdezése

Amikor egy relációt valamilyen UDT-típussal definiálunk, akkor a sorokat úgy kell tekintenünk, mint egyszerű objektumokat, és nem mint az UDT attribútumaira hivatkozó komponensek listáját. Ennek megértéséhez tekintsük a `Filmek` reláció 10.21. ábrán szereplő definícióját. Ennek a relációnak a típusa UDT `FilmTípus`, amelynek három attribútuma van: `filmcím`, `év` és `műfaj`. Egy `Filmek` relációbeli t sornak viszont csak egy komponense van, nem pedig három. Ez a komponens pedig maga az objektum.

Ha az objektumban „mélyebbre ásunk", akkor kiszedhetjük a `FilmTípus` típusból a három attribútumának értékét, valamint használhatjuk a típuson értelmezett metódusokat is. Ezeket az attribútumokat viszont helyesen el kell tudjuk érni, mivel ezek nem magának a sornak az attribútumai. Minden UDT-nek az összes attribútuma rendelkezik egy implicit módon definiált, úgynevezett *megfigyelő (observer) metódus*sal, amelynek neve egy x attribútumra $x()$. Ugyanúgy alkalmazhatjuk ezt a metódust, mint ahogyan bármelyik UDT-metódust, azaz egy ponttal hozzácsatoljuk egy olyan kifejezéshez, amely azonos egy ugyanilyen típusú objektummal. Tehát, ha t T típusú változó, és x a T attribútuma, akkor $t.x()$ a t-vel jelölt sorban (objektumban) lévő értéke x-nek.

10.22. példa. Keressük meg a 10.21. ábrán szereplő `Filmek` relációból a *King Kong* című filmhez tartozó gyártási év(ek)et. A következő út alkalmas erre:

```
SELECT m.év()
FROM Filmek m
WHERE m.filmcím() = 'King Kong';
```

Annak ellenére, hogy az m sorváltozóra itt nincs szükség, mégis kell egy változó, amelynek az értéke egy `FilmTípus` (a `Filmek` relációhoz tartozó UDT) típusú objektum. A `WHERE` záradék feltétele összehasonlítja a `'King Kong'` konstanst és az `m.filmcím()` értékét. Ez utóbbi a `FilmTípus` `filmcím` attribútumának megfigyelő eljárása. Hasonlóan a `SELECT` záradékban szereplő `m.év()` által kifejezett értékhez, ahol a kifejezés az m objektum `év` komponenshez tartozó megfigyelő metódusát alkalmazza. □

Gyakorlatban az objektumrelációs ABKR-ekben nem használják a metódusszintaxist egy objektum egy attribútumához történő hozzáférés esetén. Ehelyett eldobják a zárójeleket, és az alábbi példában bemutatott módszert alkalmazzák. Például a 10.22. példa felírása ekkor:

```
SELECT m.év
FROM Filmek m
WHERE m.filmcím = 'King Kong';
```

Az m sorváltozó viszont itt is kelleni fog.

A pont művelet metódusokra használható és arra is, hogy megtaláljuk az objektum attribútumainak értékeit. Ezeknél a metódusoknál még akkor is kell zárójelet használni, ha nincs argumentumuk.

10.23. példa. Tegyük fel, hogy a `FilmSzínész` relációt a `SzínészTípus` UDT segítségével adtuk meg, ami – a 10.14. példát felidézve – rendelkezett egy `CímTípus` típusú `cím` attribútummal. Ez utóbbi típusnak volt egy `házSzám()` eljárása, amely kiszedi a házszámot egy `CímTípus` típusú objektumból (lásd 10.15. példa). Ekkor a következő lekérdezés kiszedi a `cím` komponenst az *sz* `SzínészTípus` típusú objektumból, majd erre a `CímTípus` típusú objektumra alkalmazza a `házSzám()` eljárást.

```
SELECT MAX(sz.cím.házSzám())
FROM FilmSzínész sz
```

A visszakapott eredmény a legnagyobb házszám lesz a színészek házszámai közül. □

10.5.3. Generáló és változtató függvények

A hozzáférési metóduson kívül két olyan, az UDT-definíció során automatikusan létrehozott metódus létezik, amelyek használhatók egy UDT-nek megfelelő adat létrehozásához, illetve egy UDT típusú objektum valamely komponensének módosításához. Ezek pedig a következők:

1. A *generáló metódus (generator method).* Ezen metódus rendelkezik a típus nevével, viszont argumentummal nem. Egy másik nem megszokott tulajdonsága az, hogy objektum nélkül is meghívható. Azaz, ha T egy UDT, akkor $T()$ egy olyan T típusú objektummal tér vissza, amelynek az egyes komponensei nem rendelkeznek értékkel.

2. *Változtató metódus (mutator method).* A T UDT minden x attribútumára létezik egy $x(v)$ változtató metódus, amely megváltoztatja azon objektum x attribútumának értékét v-re, amelyre alkalmaztuk. Megjegyezzük, hogy egy attribútumra alkalmazott változtató, illetve hozzáférési metódus is rendelkezik az attribútum nevével, de a változtatónak van argumentuma.

10.24. példa. Írjunk olyan PSM-eljárást, amelynek egy utca, egy város és egy név lesz az argumentuma, illetve amely a megfelelő generáló és változtató függvények hívásaival ezen három értékből képzett objektumot szúrja be a `FilmSzínész` relációba (amelynek a 10.17. példa alapján `SzínészTípus` a típusa). Idézzük fel a 10.14. példából, hogy a `SzínészTípus` objektumainak van egy `név` komponense, amely egy karakterlánc, illetve egy `cím` komponense, amely egy `CímTípus` típusú objektum. A `SzínészBeszúr` metódust a 10.22. ábrán szemléltetjük.

A 2–4. sorokban bevezetjük az u, v és n argumentumokat, amelyek a megfelelő utca, város és név értékeket szolgáltatják majd. Az 5., illetve 6. sorban két

```
1)  CREATE PROCEDURE SzínészBeszúr(
2)      IN u CHAR(50),
3)      IN v CHAR(20),
4)      IN n CHAR(30)
    )
5)  DECLARE újCím CímTípus;
6)  DECLARE újSzínész SzínészTípus;

    BEGIN
7)      SET újCím = CímTípus();
8)      SET újSzínész = SzínészTípus();
9)      újCím.utca(u);
10)     újCím.város(v);
11)     újSzínész.név(n);
12)     újSzínész.cím(újCím);
13)     INSERT INTO FilmSzínész VALUES(újSzínész);
    END;
```

10.22. ábra. SzínészTípus objektumok létrehozása és tárolása

lokális változót deklarálunk. Mindkettő az objektum típusában szereplő egyik UDT-t jelenti, amely benne van a `FilmSzínész` relációban. A 7. és 8. sorban üres objektumokat hozunk létre ezekhez a típusokhoz.

A 9–10. sorok valódi értéket adnak az `újCím` objektumnak. Az értéket az eljárás argumentumaiból képezzük egy utcából és egy városból. A 11. sor ehhez hasonlóan az `újSzínész` objektum `név` komponensének értékét beállítja n argumentum értékére. Ezután a 12. sor veszi a teljes `újCím` objektumot, és ennek értékét átadja az `újSzínész cím` komponensének értékéül. Végül a 13. sor beszúrja az így létrehozott objektumot a `FilmSzínész` relációba. Megjegyezzük, hogy mint mindig, egy UDT-típussal adott reláció csak egyetlen komponenssel rendelkezik, még akkor is, ha a komponensnek több attribútuma van, mint jelen példánk esetében a `név` és a `cím`.

A `FilmSzínész` táblába beszúrhatunk például az alábbi `SzínészBeszúr` eljáráshívással:

```
CALL SzínészBeszúr('345 Spruce u.',
    'Glendale', 'Gwyneth Paltrow');
```

□

Egy UDT által adott relációba történő beszúrás jóval egyszerűbb, ha az ABKR végzi, vagy ha létrehozunk egy olyan generáló függvényt, amely átveszi az UDT attribútumainak megfelelő értékeket, és egy megfelelő objektumot ad vissza. Ha például lenne egy `CímTípus(u,v)`, illetve egy `SzínészTípus(n,c)` függvényünk, amelyek a jelzett típus objektumával térnének vissza, akkor a

10.24. példa végén szereplő beszúrást megtehetnénk egy általános alakú INSERT utasításként:

```
INSERT INTO FilmSzínész VALUES(
    SzínészTípus('Gwyneth Paltrow',
        CímTípus('345 Spruce u.', 'Glendale')));
```

10.5.4. Rendezési viszonyok UDT-ken

Az UDT objektumai természetükből fakadóan elvontak abban az értelemben, hogy nincs lehetőség sem két azonos UDT-vel rendelkező objektum összehasonlítására, sem arra, hogy megvizsgáljuk az „egyenlőségüket", vagy azt, hogy valamelyik nagyobb-e a másiknál. Még két azonos komponensekkel rendelkező objektumot sem tekinthetünk egyenlőnek, hacsak nem jelezzük a rendszernek, hogy tekintse azokat annak. Ehhez hasonlóan arra sincs egyértelmű módszer, hogy az UDT-vel definiált reláció sorait rendezhessük, kivéve, ha definiálunk egy olyan függvényt, amely megállapítja az adott UDT típusú objektumok elsőbbségét egymással szemben.

Már több SQL-műveletet láttunk, amelyek vagy egyenlőségi vizsgálatot, vagy mind egyenlőség, mind „kisebb mint" vizsgálatot is igényeltek. Az ismétlődéseket például nem szüntethetjük meg, ha nem tudjuk megmondani, hogy két sor mikor egyenlő. Nem csoportosíthatunk UDT-típusú attribútumok szerint az adott UDT-n értelmezett egyenlőségi vizsgálat hiányában. Nem használhatjuk sem az ORDER BY záradékot, sem az összehasonlítást (mint a $<$) a WHERE záradékban, hacsak nem tudunk bármely két elemet összehasonlítani.

Egy rendezés vagy összehasonlítás megadásához az SQL megengedi egy CREATE ORDERING utasítás kiadását bármilyen UDT-re. Ez az utasítás többféle alakban is előfordulhat, mi viszont csak a két legegyszerűbb lehetőséget mutatjuk meg:

1. A következő utasítás azt fejezi ki, hogy a T UDT két tagját egyenlőnek tekintjük, ha az összes megfelelő komponensük megegyezik. Ekkor a $<$ nincs definiálva a T UDT objektumaira nézve.

   CREATE ORDERING FOR T EQUALS ONLY BY STATE;

2. A most következő utasítás pedig azt jelenti, hogy a hat összehasonlító (<, <=, >, >=, = és <>) operátor bármelyike alkalmazható a T UDT objektumaira.

   CREATE ORDERING FOR T
   ORDERING FULL BY RELATIVE WITH F;

   Az x_1 és x_2 objektumok összehasonlításához az objektumokra alkalmazott F függvényt használjuk. Ezt a függvényt úgy kell megfogalmazni, hogy

ha $F(x_1, x_2) < 0$, akkor arra tudjunk következtetni, hogy $x_1 < x_2$. Tehát $F(x_1, x_2) = 0$ azt jelenti, hogy $x_1 = x_2$, $F(x_1, x_2) > 0$ pedig azt, hogy $x_1 > x_2$. Ha az `ORDERING FULL`-t kicserélnénk az `EQUALS ONLY` kulcsszóra, akkor $F(x_1, x_2) = 0$ jelzi, hogy $x_1 = x_2$, miközben bármilyen másik $F(x_1, x_2)$ értékből következik, hogy $x_1 \neq x_2$. Ebben az esetben a $<$ összehasonlítás nem lehetséges.

10.25. példa. Tekintsünk egy lehetséges rendezést a 10.14. példában szereplő `SzínészTípus` UDT-re. Ha csak az egyenlőséget akarjuk értelmezni az ilyen UDT-objektumokra, akkor deklaráljuk a következőt:

```
CREATE ORDERING FOR SzínészTípus EQUALS ONLY BY STATE;
```

A fenti utasítás alapján két `SzínészTípus` objektum akkor és csak akkor egyenlő, ha a neveik (a névattribútumok értékei) mint karakterláncok megegyeznek, és a címeik (a címattribútumok értékei) ugyanazt a `CímTípus` UDT-objektumot adják.

A probléma az, hogy egy `CímTípus`-ú objektum nem feltétlenül egyenlő önmagával, hacsak nem definiálunk a `CímTípus`-ra is egy rendezést. Azaz legalább egy egyenlőségvizsgálatot létre kell hoznunk a `CímTípus`-ra. Ennek egy egyszerű módja, hogy két `CímTípus` akkor és csak akkor legyen egyenlő, ha az utcáik és a városaik is megegyeznek, mint füzérek. Ezt a következőképpen érhetjük el:

```
CREATE ORDERING FOR CímTípus EQUALS ONLY BY STATE;
```

Egy másik lehetőségként egy teljes rendezést definiálhatunk a `CímTípus` objektumain. Egy ésszerű rendezés lehet, ha először a címeket városok szerint ábécé sorrendbe, majd az ugyanazon városban lévő címeket utcák szerinti ábécé sorrendbe rendezzük. Ehhez definiálnunk kell egy `CímKEN` függvényt, amely két argumentumot vár, és pozitív, nulla vagy negatív értékkel tér vissza annak függvényében, hogy az első cím nagyobb, egyenlő vagy kisebb a másodiknál. Deklaráljuk az alábbit:

```
CREATE ORDERING FOR CímTípus
ORDERING FULL BY RELATIVE WITH CímKEN;
```

A `CímKEN` függvény a 10.23. ábrán található. Megjegyezzük, hogy ha elérünk a 7. sorba, akkor a két `város` komponense biztos meg fog egyezni, tehát az `utca` komponenseket is összehasonlítjuk. Hasonlóan, ha a 9. sornál járunk, akkor csak az a lehetőség maradt, hogy a városok megegyeznek, de az első argumentumhoz tartozó utcanév ábécé sorrendben megelőzi a másodikhoz tartozó utcanevet. □

Gyakorlatban a forgalomban lévő ABKR-eknek van saját módszerük arra, hogyan engedjék meg a felhasználónak az UDT-n történő összehasonlítás definiálását. A fent említett két megközelítésen kívül még van néhány elérhető lehetőség:

```
1)  CREATE FUNCTION CímKEN(
2)      x1 CímTípus,
3)      x2 CímTípus
4)  ) RETURNS INTEGER

5)  IF x1.város() < x2.város() THEN RETURN(-1)
6)  ELSEIF x1.város() > x2.város() THEN RETURN(1)
7)  ELSEIF x1.utca() < x2.utca() THEN RETURN(-1)
8)  ELSEIF x1.utca() = x2.utca() THEN RETURN(0)
9)  ELSE RETURN(1)
    END IF;
```

10.23. ábra. A címobjektumok összehasonlító függvénye

a) Szigorú objektumegyenlőség. Két objektum akkor és csak akkor egyenlő, ha ugyanazok az objektumok.

b) Eljárással definiált egyenlőség. Egy függvényt alkalmazunk a két objektumra, amely igaz vagy hamis értékkel tér vissza attól függően, hogy egyenlőnek tekinthetőek-e vagy sem.

c) Eljárással definiált leképezés. Egy függvényt alkalmazunk egy objektumra, és egy valós számot kapunk vissza. Az objektumok összehasonlítását ezekkel a valós számokkal végezzük el.

10.5.5. Feladatok

10.5.1. feladat. A 10.20. példában szereplő `SzerepelBenne` relációt és a `SzerepelBenne` relációból elérhető `Filmek` és `FilmSzínész` relációk felhasználásával írjuk meg a következő lekérdezéseket:

a) Határozzuk meg a *Dogma* című film szereplőit.

! *b)* Határozzuk meg az összes olyan film címét és gyártási évét, amelyben a szereplő színészek legalább egyike Malibun él.

c) Határozzuk meg az összes filmet (`FilmTípus` típusú objektumokat), amelyekben szerepel Melanie Griffith.

! *d)* Keressük meg a legalább öt színésszel készített filmeket (cím és gyártási év).

10.5.2. feladat. A 10.4.3. feladat adatbázissémáját felhasználva írjunk lekérdezéseket az alábbiakhoz. Ne felejtsünk hivatkozást használni ott, ahol az célszerű.

a) Határozzuk meg a 60 gigabájtnál nagyobb méretű merevlemezzel rendelkező PC-k gyártóit.

b) Határozzuk meg a lézernyomtatók gyártóit.

! *c)* Készítsünk táblázatot, amely minden laptopmodellhez megadja annak a laptopnak a modellszámát, amelyiket ugyanaz a gyártó gyárt, és ezek közül a legnagyobb sebességű.

10.5.3. feladat. Használjuk a 10.4.5. feladat sémáját, és fogalmazzuk meg az alábbi lekérdezéseket. Ahol lehet, próbáljuk meg a hivatkozásokat használni és elkerülni az összekapcsolásokat (vagyis alkérdések, illetve egynél több sorváltozó használatát a `FROM` záradékban).

a) Keressük meg a 35 000 tonnánál nagyobb vízkiszorítású hajókat.

b) Keressük meg azokat a csatákat, amelyekben legalább egy hajó elsüllyedt.

! *c)* Keressük meg azokat a hajóosztályokat, amelyekbe 1930 után felavatott hajók tartoztak.

!! *d)* Keressük meg azokat a csatákat, amelyekben legalább egy amerikai hajó megsérült.

10.5.4. feladat. Feltéve, hogy a 10.23. ábrán szereplő `CímKEN` függvény felhasználható, írjunk egy megfelelő függvényt a `SzínészTípus` típusú objektumok összehasonlítására, illetve deklaráljuk úgy ezt a függvényt, hogy a `SzínészTípus` objektumok rendezésének alapja legyen.

! **10.5.5. feladat.** Írjunk olyan eljárást, amelynek az argumentuma egy színésznek a neve, és kitörli a `SzerepelBenne` és `FilmSzínész` táblából az összes olyan sort, melyben az adott színész szerepel.

10.6. Online analitikus feldolgozás

Az adatbázisok fontos alkalmazási területe az adatok vizsgálata minták vagy trendek meghatározásának céljából. Ezt a tevékenységet nevezzük OLAP-nak (On-Line Analitic Processing – Online Analitikus Feldolgozás). Általában olyan, meglehetősen összetett lekérdezéseket takarnak, amelyek egy vagy több összesítést tartalmaznak. Ezen előbb említett lekérdezéseket gyakran OLAP- vagy *döntéstámogató lekérdezés*ként emlegetik. A 10.6.2. alfejezetben adunk néhány felhasználási példát is. Egy gyakori példa lehet az, amikor egy társaság azokat a termékeit akarja meghatározni, amelyek eladásai jelentős mértékben növekedtek vagy csökkentek.

A döntéstámogató lekérdezések általában nagy mennyiségű adattal dolgoznak még akkor is, ha a lekérdezés eredménye kicsi. Szemben az olyan általános adatbázis-operációkkal, mint a bankbetétek vagy a járatfoglalások, amelyek csak az adatbázis egy kisebb részét érintik, az ilyen operációkat gyakran

OLTP-nek (On-Line Transaction Processing – Online Tranzakciós Feldolgozás) nevezzük.

Az ABKR-ek egyik aktuális irányvonala az OLAP-lekérdezések támogatása. Például mostanában a rendszerek gyakran támogatják valamilyen formában az „adatkockákat". Az ilyen rendszerek szerkezetét a 10.7. alfejezetben vizsgáljuk majd.

10.6.1. Az OLAP és az adattárházak

Egy általános OLAP-alkalmazás a központi adatbázis egy elkülönített másolatán helyezkedik el, ezt nevezzük *adattárház*nak. Több különböző adatbázisból származó adatokat egy adattárházba integrálják. Az egyik elterjedt forgatókönyv, hogy az adattárházakat csak éjjelente frissítik, így az analízis egy rögzített másolaton futhat a nap során. Vagyis az adattárház adatai már 24 óra alatt elavulnak, ami az OLAP-lekérdezések eredményének az időszerűségét erősen korlátozza, de ez az „időeltolódás" sok döntéstámogató alkalmazásban még a tűréshatáron belül marad.

Különböző okai vannak annak, hogy a tárházak miért játszanak olyan fontos szerepet az OLAP-alkalmazásokban. Egyrészt lehetséges, hogy az adataink kezdetben számos különböző adatbázisban vannak szétszórva, vagyis ahhoz, hogy OLAP-lekérdezéseket tudjunk végrehajtani, egy adattárházat kell létrehoznunk, amelyben megfelelően tudjuk szervezni és központosítani az egyesített adatokat. Másrészt viszont legtöbbször fontosabb annak a ténye, hogy az OLAP-lekérdezések végrehajtása – mivel ezek összetettek és az adatbázis nagy részét érintik – túl sok időt vesz igénybe az olyan tranzakciókezelő rendszerekben, amelyektől nagy teljesítményt várunk el. Idézzük fel a 6.6. alfejezetben tárgyalt sorolható tranzakciókat. Egy, az adatbázis nagyobb részét érintő hosszú tranzakció más tranzakciókkal történő soros végrehajtása esetén leállhatnának a megszokott OLAP-műveletek, és ez nem feltétlenül tolerálható. Például nem kellene megengedni az új kereskedelmi adatok felvitelét, amikor egy konkurens OLAP-lekérdezés éppen a kereskedési átlagot számítja.

10.6.2. OLAP-alkalmazások

Az OLAP-alkalmazások általában fogyasztásra vonatkozó adatokból összeállított tárházat használnak. Nagyobb üzlethálózatok terabájtnyi információt halmoznak fel arról, hogy melyik üzletükben melyik cikkből mennyit adtak el. A cégre váró problémák vagy nagy lehetőségek előrejelzésében nagy szerepe lehet az olyan lekérdezéseknek, amelyek a fogyasztási adatok csoportosítása, összesítése alapján a valamilyen szempontból jelentős csoportokat azonosítani tudják.

10.26. példa. Tegyük fel, hogy a Földi Malac Autókereskedés nevű cég létrehoz egy adattárházat, amely alapján elemezni tudja majd a gépkocsik iránti keresletet. A tárház sémája a következő:

```
Eladások(sorszám, dátum, forgalmazó, ár)
Autók(sorszám, modell, szín)
Forgalmazók(név, város, állam, tel)
```

Egy jellemző döntéstámogató lekérdezés a 2006. április 1-jén vagy azóta történt értékesítések alapján megállapíthatná, hogyan változik az eladott járművek átlagára államonként. Egy ilyen lekérdezést mutatunk be a 10.24. ábrán.

```
SELECT állam, AVG(ár)
FROM Eladások, Forgalmazók
WHERE Eladások.forgalmazó = Forgalmazók.név AND
    dátum >= '2006-01-04'
GROUP BY állam;
```

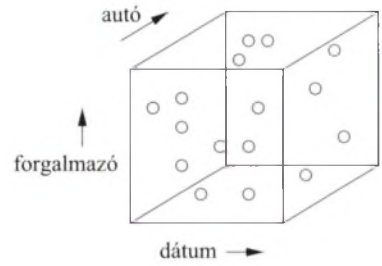
10.24. ábra. Államonkénti átlagos fogyasztói ár

Figyeljük meg, hogy a lekérdezés végigveszi az adatbázis nagy részét, miközben az `Eladások` relációban tárolt vásárlási adatokat osztályozza a forgalmazó címe szerint. Ezzel szemben a „Milyen áron adták el a 123-as sorszámú autót?" típusú gyakori OLTP-lekérdezések az adatbázis egyetlen sorát érintik csupán. □

Hogy lássunk egy másik OLAP-példát is, vegyünk egy hitelkártya-kibocsátó céget, amely a kártyaigénylőiről akarja eldönteni, hogy hitelképesek lesznek-e. Készítenek egy adattárházat, amely a cég összes ügyfeléről és az eddigi befizetéseiről tartalmaz információkat. Az OLAP-lekérdezések olyan tényezőket (például életkor, jövedelem, ingatlantulajdon, irányítószám) fognak keresni, amelyek segíthetnek annak az előrejelzésében, hogy egy adott ügyfél be fogja-e fizetni időben a számláit vagy nem. Ehhez hasonlóan kórházak is rendelkezhetnek betegek adatait (felvétel időpontja, elvégzett laborvizsgálatok, ezek eredménye, diagnózis, gyógykezelés leírása és így tovább) tartalmazó adattárházzal, hogy elemezni tudják a kezeléssel járó kockázatot, és ennek tükrében választják ki a legjobbnak ítélt módszert.

10.6.3. OLAP-adatok többdimenziós nézete

A tipikus OLAP-alkalmazásokban mindig létezik egy központi reláció vagy adatgyűjtemény: a *ténytáblázat* (fact table). A ténytáblázat olyan érdekkel bíró eseményeket vagy objektumokat tartalmaz, mint a 10.26. példában megismert vásárlási adatok. Gyakran segít, ha a ténytáblázatban tárolt objektumokra úgy gondolunk, mintha egy többdimenziós térben vagy „kockában" lennének elrendezve. A 10.25. ábrán háromdimenziós adatobjektumok láthatók, a kocka belső pontjaiként ábrázolva. A dimenziók elnevezései: autó, forgalmazó, dátum, a korábbi személygépkocsi-értékesítés példának megfelelően. A kocka minden egyes belső pontjára gondolhatunk úgy, mint egy-egy gépkocsieladásra, a dimenziókra pedig úgy, mintha ennek az eladásnak a körülményeit, jellemzőit írnák le.



10.25. ábra. Az adatok többdimenziós kockába szervezése

Egy, a 10.25. ábrához hasonló adatteret informálisan „adatkockának" vagy még pontosabban *nyers adatkocká*nak nevezzük, amely nevet akkor használjuk, amikor meg akarjuk különböztetni a 10.7. alfejezetben szereplő, ennél jóval összetettebb „adatkockától". Ez utóbbi adatkockát formális adatkockának nevezzük a megkülönböztethetőség kedvéért. Ez a nyers adatkockától z két dologban különbözik:

1. Tartalmazza a dimenziók részhalmazaira vett összesítéseket is csakúgy, mint magukat az adatokat.

2. A formális adatkocka pontjai ábrázolhatják a nyers adatkocka pontjainak egy kezdeti összesítését. Például az „autó" dimenzió ahelyett, hogy ábrázolja az összes autót egyenként (ahogy azt a nyers adatkockánál javasoltuk), csak modellkénti összesítéseket tartalmazhat. Egy formális adatkockának lesznek olyan pontjai, amelyek egy adott modellhez tartozó összes autónak az eladásait összesítik egy eladóra nézve, egy adott napon.

A formális adatkocka és a nyers adatkocka közti különbség tükröződik a kocka szerkezetű OLAP-adatot támogató speciális rendszerek által vett két fő irányvonalban is:

1. *ROLAP* vagy *relációs OLAP.* Ebben a megközelítésben az adatot relációkban tároljuk, de egy speciális szerkezetben, a „csillagsémában" (lásd bővebben 10.6.4. alfejezetben). A relációk egyike a „ténytáblázat", amely a *nyers*, vagyis a „még nem összesített" adatokat tartalmazza, amely így már a nyers adatkocka elnevezését is megmagyarázza. A többi reláció pedig megadja a szükséges információkat bármely választott dimenzió értékeire nézve. A rendszer lekérdező nyelve, indexelési szerkezete és más képességei ezt az adatszervezési módot kihasználva alakíthatók.

2. *MOLAP* vagy *többdimenziós OLAP.* Ebben az esetben a fent említett speciális szerkezetet, a formális „adatkockát" használjuk az adat tárolására. Ahogy már említettük, ez az adat gyakran már részben összesített. Az efféle adat OLAP-lekérdezéseit támogatandó, a rendszer nem relációs műveleteket is megvalósíthat.

10.6.4. A csillagséma

A *csillagséma* a ténytáblázat sémájából áll, amely több más relációhoz, a „dimenziótáblákhoz" kapcsolódik. A „csillag" központjában található a ténytábla, a csúcsaiban pedig a dimenziótáblák. A ténytáblának általában van néhány dimenzióattribútuma – ezek az egyes *dimenzió*kat jelölik –, és egy vagy több *függő* attribútuma, amelyek az adat, mint a többdimenziós tér egy pontjának, mint egésznek számunkra fontos tulajdonságait jelölik. Például a vásárlásra vonatkozó adatok dimenziói között szerepelhet a vásárlás időpontja, helyszíne (melyik üzletben történt), a vásárolt cikk típusa, a fizetési mód (készpénz vagy hitelkártya) és így tovább. Függő attribútum(ok) lehet(nek) például a fogyasztói ár, a kereskedelmi ár vagy az adó mennyisége.

10.27. példa. Az előző példában bevezetett `Eladások` reláció

```
Eladások(sorszám, dátum, forgalmazó, ár)
```

a ténytáblázat. A dimenziók a következők:

1. `sorszám`, az eladott gépkocsit jelöli, azaz az összes gépkocsi terében az ezzel a sorszámmal rendelkező autónak megfelelő pont.

2. `dátum`, a vásárlás időpontját jelöli, azaz a vásárlás, mint esemény pozícióját az idő dimenzióban.

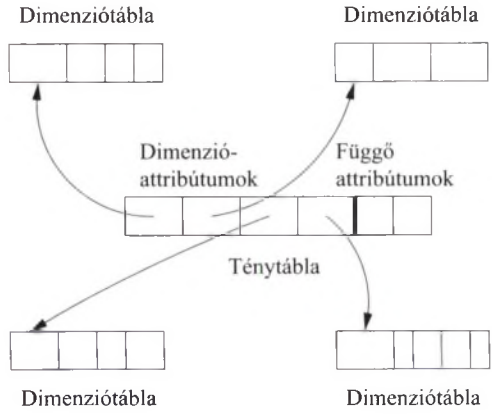
3. `forgalmazó`, az esemény pozícióját jelöli az összes forgalmazó terében.

Az egyetlen függő attribútum az `ár`, amelyre az adatbázis OLAP-lekérdezéseinek jellemzően valamilyen összesített formában lesz majd szüksége. Persze az összár vagy az átlagár megállapítása mellett a leszámlálós lekérdezéseknek is lehet értelmük. Például „adjuk meg azt a listát, amely minden forgalmazóhoz tartalmazza az általuk 2006 májusában megtörtént eladásoknak a számát". □

A ténytáblázat kiegészítéseképpen használhatók a *dimenziótáblák*, amelyek az egyes dimenziók lehetséges értékeit írják le. A ténytáblázat mindegyik dimenzióattribútuma rendszerint idegen kulcsként működik a megfelelő dimenziótáblában, ahogy ezt a 10.26. ábra szemlélteti. A dimenziótábla attribútumai azt is leírják, hogy melyek azok a lehetséges csoportosítások, amelyeknek értelmük lenne egy SQL-lekérdezés `GROUP BY` záradékában. A következő példán keresztül bemutatjuk az új fogalmakat.

10.28. példa. A 10.26. példa gépkocsi-adatbázisában a dimenziótáblák közül kettő nyilvánvaló:

```
Autók(sorszám, modell, szín)
Forgalmazók(név, város, állam, tel)
```



10.26. ábra. A ténytábla dimenzióattribútumai a dimenziótáblák kulcsmezőire hivatkoznak

Az `Eladások` ténytábla `sorszám` attribútuma idegen kulcs, és az `Autók` dimenziótáblában[2] a `sorszám` attribútumra hivatkozik. Az `Autók.modell` és az `Autók.szín` attribútumok pedig az adott autó tulajdonságait írják le. Szerepelhetne még sokkal több attribútum is ebben a relációban, igaz/hamis értékekkel jelölve, hogy a kocsiban van-e automata sebességváltó vagy akármilyen más extra felszerelés. Ha az `Eladások` ténytáblát összekapcsoljuk az `Autók` dimenziótáblával, akkor a `modell` és `szín` attribútumok szerint különböző érdekes módon lehet csoportosítani a vásárlásokat. Például lebonthatjuk az adatokat szín szerint, vagy a Tüskés modell értékesítéseit hónap és forgalmazó szerint.

Hasonlóan, az `Eladások` tábla forgalmazó attribútuma idegen kulcs a `Forgalmazó` dimenziótáblában a `név` attribútumra hivatkozva. Ha az `Eladások` és `Forgalmazó` táblákat összekapcsoljuk, további lehetőségek adódnak az adatok csoportosítására. Például lebonthatjuk a vásárlásokat állam, város vagy forgalmazó szerint.

Eltűnődhetünk azon, hol találjuk az időnek (az `Eladások` tábla `dátum` attribútumának) megfelelő dimenziótáblát. Mivel az idő egy fizikai adottság, nincs értelme az adatbázisban időre vonatkozó tényeket tárolni, hiszen a „Melyik évre esik 2007. július 5.?" típusú lekérdezések eredményét úgysem tudjuk megváltoztatni. Azonban az elemzőknek gyakran van szükségük különböző időegységek, például hét, hónap, negyedév, év szerinti csoportosításra, segít, ha az idő fogalmát beépítjük az adatbázisba, éppen úgy, mintha a következő dimenziótáblával rendelkeznénk:

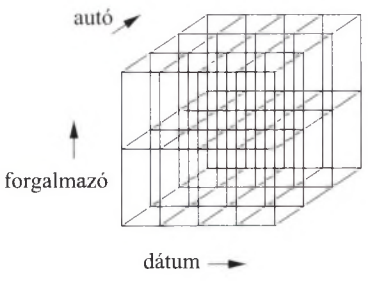
```
Napok(nap, hét, hónap, év)
```

[2] Most véletlenül a `sorszám` az `Eladások` relációban is kulcsként működik, de nem kell hogy legyen olyan attribútum, amely a ténytáblának is kulcsa és idegen kulcs is egyben valamelyik dimenziótáblában.

A „reláció" egy jellegzetes, 2007. július 5-ének megfelelő sora lenne a következő: (5, 27, 7, 2007). Ezt úgy értelmezzük, hogy ez a nap a 2007. év hetedik hónapjának az ötödik napjára esik, és történetesen a 2007. év 27-edik teljes hetéhez tartozik. Ez bizonyos fokig redundáns, mert a hetet ki lehet számolni a másik három attribútum alapján, viszont a hetek nem illeszthetők pontosan a hónapokhoz, azaz hetek szerinti csoportosításból nem nyerhetünk hónapok szerinti csoportosítást, és megfordítva sem. Tehát van értelme annak, ha úgy képzeljük el, hogy a hetek és a hónapok is jelölve vannak ebben a „dimenziótáblázatban". □

10.6.5. Szeletelés és kockázás

A nyers adatkockára úgy is gondolhatunk, mintha minden dimenzió mentén valamilyen finomsággal fel lenne osztva. Például az idődimenziót feloszthatjuk (SQL-es szóhasználattal „csoportosíthatjuk"[3]) napok, hetek, hónapok, évek szerint, de azt is megtehetjük, hogy nem osztjuk fel egyáltalán. Az autó dimenziót feloszthatjuk modell szerint, szín szerint, modell és szín szerint, a forgalmazó dimenziót pedig a forgalmazó neve, a város vagy az állam szerint. Természetesen e két dimenzió esetén is megtehetjük, hogy egyáltalán nem csoportosítunk.



10.27. ábra. A dimenziók felosztása darabolja (kockázza) a kockát

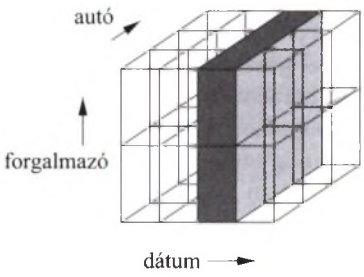
Ha minden dimenzióhoz választunk egy felosztást, ez „felkockázza" az adatkockát, ahogy ez a 10.27. ábrán látható. A kockázás eredményeképpen az adatkockában kisebb kockák jönnek létre. Az ezeket alkotó pontcsoportok jellemzőit az a lekérdezés összesíti, amely a `GROUP BY` záradékban a felosztásnak megfelelő csoportosítást hajtja végre. A `WHERE` záradékon keresztül a lekérdezésnek arra is lehetősége van, hogy csak bizonyos felosztásokra koncentráljon egy (vagy több) dimenzió mentén, azaz a kockának csak bizonyos „szeleteivel" foglalkozzon.

10.29. példa. A 10.28. ábra egy olyan lekérdezésre utal, amely a kockából az egyik dimenzió (dátum) mentén kivág egy szeletet, majd ezt a másik két dimenzió mentén (autó és forgalmazó) tovább kockázza. A dátum négy részre van felosztva, talán arra a négy évre, amelyek alatt ezek az adatok összegyűltek.

[3] Az angol *group by* kifejezésből. *(A fordító megjegyzése.)*

Az ábra satírozása azt fejezi ki, hogy ebből a négy évből csak egyetlenegyre vagyunk kíváncsiak.

Az autók három csoportra vannak osztva, például szedánokra (négyüléses, zárt autókra), kupékra (sportkocsikra) és a kabrioletekre (leereszthető tetejű kocsikra). A forgalmazók kettőre, például a keleti, illetve a nyugati országrészen működőkre. A lekérdezés eredménye egy olyan táblázat, amely a számunkra érdekes évben megadja a gépkocsi-értékesítésekből származó összbevételt mind a hat kategóriában. □



10.28. ábra. Egy szelet kiválasztása a darabolt (kockázott) kockából

Az úgynevezett „szeletelős-kockázós" lekérdezés általános formája tehát a következő:

```
SELECT <csoportosító attribútumok és összesítések>
FROM <ténytábla nulla vagy több dimenziótáblával összekapcsolva>
WHERE <bizonyos attribútumok konstans értékekkel vannak összehasonlítva>
GROUP BY <csoportosító attribútumok>;
```

10.30. példa. Folytassuk a személygépkocsis példánkat, de a 10.28. példában megtárgyalt `Napok` fogalmi szintű dimenziótáblázattal együtt. Ha a Tüskés modellt nem vásárolják annyian, mint gondoltuk, megpróbálhatjuk kiszűrni azokat a színeket, amelyek nem mennek olyan jól. Ez a lekérdezés csak az `Autók` dimenziótáblát használja, és SQL-ben így nézhet ki:

```
SELECT szín, SUM(ár)
FROM Eladások NATURAL JOIN Autók
WHERE modell = 'Tüskés'
GROUP BY szín;
```

Először szín szerint kockázza, majd modellenként szeleteli az adatkockát. Eközben a Tüskés modellre összpontosít, és a többi adatot figyelmen kívül hagyja.

Tegyük fel, hogy a lekérdezés eredménye nem mondott sokat, minden színből körülbelül ugyanakkora jövedelmünk származott. Mivel a lekérdezés az időt nem bontotta részekre, a színek szerinti végösszeget az egész időtartamra számította

Mélyre ásás és felgörgetés

A 10.30. példában az adatkockát szeletelő és kockázó lekérdezések sorozatában két gyakori mintával találkozunk.

1. A *mélyre ásás* az a folyamat, amely során a megfelelő dimenziókat egyre finomabb részekre osztjuk, és/vagy a dimenzió bizonyos értékeire koncentrálunk. A 10.30. példában az utolsó lépés kivételével mindegyik a mélyre ásás folyamatába illik.
2. A *felgörgetés* az egyre durvább particionálás folyamata. Példaként erre az utolsó két lépést hozhatjuk fel, amelyben hónapok helyett évek szerint csoportosítottunk, hogy az adatok esetlegességét kiküszöböljük.

ki. Feltehetjük, hogy egy vagy több szín gyenge szereplése csak az utóbbi időre jellemző. Ekkor megpróbálkozhatunk a következő, immár bővített lekérdezéssel, amely hónapok szerint is feloszt:

```
SELECT szín, hónap, SUM(ár)
FROM (Eladások NATURAL JOIN Autók) JOIN Napok ON dátum = nap
WHERE modell = 'Tüskés'
GROUP BY szín, hónap;
```

Fontos észben tartanunk, hogy a `Napok` reláció nem egy szokásos módon tárolt reláció, bár kezelhetjük úgy, mintha a következő sémával rendelkezne.

```
Napok(nap, hét, hónap, év)
```

Az OLAP-lekérdezésekre specializált rendszerek többek között azért is különböznek a megszokott ABKR-ektől, mert képesek az efféle „relációk" használatára is.

Az előző lekérdezés alapján esetleg azt állapíthatjuk meg, hogy a piros Tüskések iránt az utóbbi időben nem volt túl nagy kereslet. Következő lépésként megpróbálhatnánk kideríteni, hogy ez általánosan az összes forgalmazóra igaz, vagy csak egy részüknél figyelhető meg ez a probléma. Az újabb lekérdezésben csak a piros Tüskésekre összpontosítunk, és a forgalmazó dimenzió szerint is csoportosítunk. Ily módon a lekérdezés:

```
SELECT forgalmazó, hónap, SUM(ár)
FROM (Eladások NATURAL JOIN Autók) JOIN Napok ON dátum = nap
WHERE modell = 'Tüskés' AND szín = 'piros'
GROUP BY hónap, forgalmazó;
```

Ezen a ponton azt vesszük észre, hogy a piros Tüskésekből havonta olyan keveset adtak el, hogy ez alapján nem figyelhető meg könnyen semmilyen jellemző irányvonal. Úgy döntünk tehát, hogy hiba volt a hónapok szerinti felosztás. Jobb ötlet lenne csak évekre bontani az adatokat és csak az utolsó két évet vizsgálni (ebben a képzeletbeli példában 2006-ot és 2007-et). A végleges lekérdezés a 10.29. ábrán látható. □

```
SELECT forgalmazó, év, SUM(ár)
FROM (Eladások NATURAL JOIN Autók) JOIN Napok ON dátum = nap
WHERE modell = 'Tüskés' AND
    szín = 'piros' AND
    (év = 2006 OR év = 2007)
GROUP BY év, forgalmazó;
```

10.29. ábra. A végleges szeletelő-kockázó lekérdezés a piros Tüskés vásárlásokról

10.6.6. Feladatok

10.6.1. feladat. Egy online számítógép-értékesítő cég adatbázisban akarja nyilvántartani a vásárlók megrendeléseit. A vevők a PC-jükbe különböző processzorok, merevlemezek és CD-, illetve DVD-olvasók közül választhatnak, és megadhatják, hogy mekkora központi memóriára van szükségük. Az adatbázis ténytáblázata lehet a következő:

```
Megrendelések(vásárló, dátum, proc, memória, lemez,
    cd, menny, ár)
```

A `vásárló` attribútum a vevő azonosítójaként működik, és egyúttal idegen kulcs a vásárlókat nyilvántartó dimenziótáblához. Ugyanez igaz a `proc`, `lemez` és `cd` attribútumokra is. A `lemez` azonosító szerepelhet például abban a dimenziótáblában, amely megadja a merevlemez gyártóját és más jellemzőit. A `memória` attribútum egy egész szám, a megrendelt memória méretét adja meg megabájtban. A `menny` attribútum a vásárló által megrendelt konfigurációk számát jelenti, az `ár` attribútum pedig ebből egy darab összköltségét.

a) Melyek a dimenzióattribútumok és melyek a függő attribútumok?

b) Néhány dimenzióattribútumhoz valószínűleg dimenziótábla szükséges. Adjunk meg ezekhez megfelelő sémákat!

! **10.6.2. feladat.** Tegyük fel, hogy az előző feladat adatbázisát vizsgáljuk annak érdekében, hogy a jellemző trendek alapján előre jelezhessük a cégnek, hogy milyen alkatrészekből kell majd többet rendelnie. Adjuk meg mélyre ásó és felgörgető lekérdezések egy olyan sorozatát, amely ahhoz a következtetéshez vezethet, hogy a vásárlók egyre inkább előnyben részesítik a DVD-meghajtót a CD-meghajtóval szemben!

10.7. Adatkockák

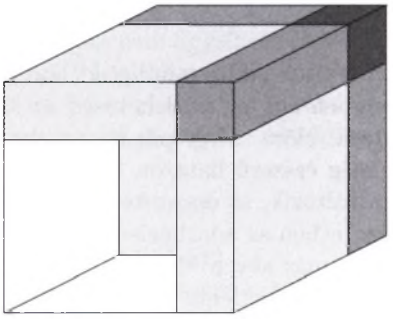
A döntéstámogató lekérdezések eddig mindig ad hoc lekérdezések formájában jelentek meg. Ezzel szemben egy másik lehetőség az összes lehetséges összesítés módszeres kiszámítása, előre. Meglepő, de az ehhez szükséges tártöbblet mennyisége gyakran a még ésszerű határon belül marad, és amíg az adattárházban tárolt adat nem változik, az összesített adat frissítése sem jelent többletköltséget. Ebben a fejezetben az adatbázis-kezelő rendszerek egy családjával, az *adatkockarendszerek*kel, más néven MOLAP- (többdimenziós OLAP-) rendszerekkel foglalkozunk. Ezek közvetlenül a 10.25. ábrán bemutatott (adat)kocka adatmodellt támogatják, és a legfontosabb OLAP-műveletek elvégzését is biztosítják.

Az adatkockarendszerekben teljesen hétköznapi dolognak számít, ha még az adatkocka-tároló rendszerbe való felvétel előtt a ténytáblázat nyers adata néhány szempont szerint összesítve van. Az autós adatbázisunkban például a csillagsémában szereplő sorszámhoz rendelt dimenziót kicserélhetjük a modell dimenziójára, így minden bejegyzés az adatkockában egy modell, egy forgalmazó, egy dátum, és ezzel együtt az adott modellből az adott napon az adott forgalmazónál történt vásárlások összegének leírásává válik. Az adatkocka pontjainak összességét továbbra is „ténytáblának" nevezzük, még ha a pontok értelmezése egy kicsit el is tér a csillagséma ténytáblájától.

10.7.1. A kockaművelet

Adott F ténytábla esetén definiálhatunk egy kibővített KOCKA(F) táblát, amely egy további, *-gal jelölt értéket ad minden dimenzióhoz. A * jelentése „bármi", és összesítést jelent annak a dimenziónak a mentén, ahol feltűnik. A 10.30. ábra szemlélteti azt az eljárást, amely során a kockához minden dimenzió mentén a * értéket, és az ennek megfelelő összesített értéket képviselő új határokat adjuk hozzá. Az ábrán három dimenziót látunk, a leghalványabb részek jelölik az egy dimenzió mentén vett összesítéseket, a kicsit sötétebbek a két dimenzió mentén, a legsötétebb kis kocka a sarokban pedig a mind a három dimenzió mentén vett összesítéseket. Gondoljuk meg, hogy ha az egyes dimenziók mentén az előforduló értékek száma elég nagy, akkor a „határ" csak kis többletet jelent a kocka terjedelméhez (azaz a ténytábla sorainak számához) képest. Ebben az esetben a KOCKA(F) tábla mérete nem sokkal nagyobb az F ténytábla méreténél.

A KOCKA(F) tábla egy olyan sora, amelyben egy vagy több dimenzióattribútum mentén * található, a függő attribútumoknak megfelelő mezőkben egy összeget (vagy egy más összesítő függvénnyel számított mennyiséget) tartalmaz. Ez úgy számítható ki, hogy vesszük az összes olyan sort, amelyben a dimenzióattribútum értéke nem *, hanem egy valódi érték, és ezen sorok függő attribútumai mentén végezzük el az összesítést. Gyakorlatilag beépítjük az adatokba az összes lehetséges dimenzióhalmazon végzett összesítés eredményét. Vegyük azonban észre, hogy a KOCKA művelet nem támogatja a „köztes szin-



10.30. ábra. A kockaművelet a dimenziók minden lehetséges kombinációján kiszámított összesítéseknek megfelelő határokkal bővíti az adatkockát

ten" számított összesítéseket. Az adatokat például vagy hagyjuk napokra (illetve az idődimenzió legfinomabb felbontásának megfelelő időközre) lebontva, vagy a teljes időtartamra nézve összesítünk. A KOCKA művelettel önmagában nem összesíthetünk hetek, hónapok vagy évek szerint.

10.31. példa. A 10.26. példa Földi Malac adatbázisát vizsgáljuk meg a KOCKA művelet lehetőségeinek fényében. Emlékeztetőül, az ott használt ténytáblázat a következő:

```
Eladások(sorszám, dátum, forgalmazó, ár)
```

A `sorszám` attribútumnak megfelelő dimenzió azonban nem alkalmas arra, hogy a kockában használjuk, mert a sorszám egyértelműen azonosítja a gépkocsikat, és így a sorszám kulcsként működik az `Eladások` relációban. Vagyis ha a kocsik árát összegezzük a teljes időtartamra vagy az összes forgalmazóra nézve úgy, hogy közben a sorszám rögzítve marad, akkor ennek nem lesz hatása – az adott sorszámú (egyetlen) autó árának az összegét kapjuk. Használhatóbb az adatkocka, ha a sorszámot két másik attribútummal, a `modell`-lel és a `szín`-nel cseréljük fel. Ezek azok az attribútumok, amelyekhez a sorszám az `Autók` dimenziótáblán keresztül az `Eladások` relációt kapcsolja. Vegyük észre, hogy ha a `sorszám` attribútumot kicseréljük a `modell` és `szín` attribútumokra, akkor a kockának már egyetlen dimenziója sem működik kulcsként. Így a kocka egy-egy bejegyzése az olyan kocsik összértékét adja, amelyek adott színűek, adott modellek és az adott napon az adott forgalmazónál vásárolták meg azokat.

Ha az `Eladások` ténytáblát az adatkockarendszerben akarjuk megvalósítani, egy további változtatás is hasznunkra válhat. Mivel a KOCKA operátor a függő attribútumokat általában összegezni szokta, ezért ha kíváncsiak vagyunk az átlagárra is, szükségünk lehet az eladott kocsik összértékére minden kategóriában (adott szín és modell az adott napon, az adott forgalmazónál) és az abba a kategóriába eső vásárlások darabszámára is. Vagyis az `Eladások` reláció, amire a KOCKA műveletet alkalmazzuk, a következő:

```
Eladások(modell, szín, dátum, forgalmazó, összért, db)
```

Az `összért` attribútum az adott modellhez, színhez, dátumhoz és forgalmazóhoz tartozó eladott kocsik összértéke, a `db` pedig az ebbe a kategóriába eső autók darabszáma. Vegyük észre, hogy ebben az adatkockában az egyes autók külön-külön nem azonosíthatók, vagyis ilyen értelemben nincsenek jelen a kockában. Természetes viszont, hogy a saját kategóriájukon belül az `összért` és `db` attribútumok értékére kifejtik a hatásukat.

Vegyük szemügyre a KOCKA(`Eladások`) relációt. Ennek egy lehetséges sora, amely az `Eladások` relációban is megtalálható, a következő:

```
('Tüskés', 'piros', '2001-05-21', 'Undok Ubul', 45000, 2)
```

Ezt úgy értelmezzük, hogy 2001. május 21-én Undok Ubul eladott két piros Tüskést, összesen 45 000 dollár értékben.

A következő sor:

```
('Tüskés', *, '2001-05-21', 'Undok Ubul', 152000, 7)
```

azt jelenti, hogy 2001. május 21-én Undok Ubul összesen hét Tüskést adott el, amelyek összértéke 152 000 dollár. Vegyük észre, hogy ez a sor előfordulhat a KOCKA(`Eladások`) relációban, de az `Eladások` biztosan nem tartalmazza.

A KOCKA(`Eladások`) reláció olyan sorokat is tartalmaz, amelyek egynél több attribútum mentén is tartalmaznak összesítést. Például a következő sor:

```
('Tüskés', *,2001-05-21', *, 2348000, 100)
```

azt jelenti, hogy 2001. május 21-én a forgalmazók összesen 100 Tüskést adtak el, és ezek összértéke 2 348 000 dollár volt.

```
('Tüskés', *, *, *, 1339800000, 58000)
```

Ez a sor pedig azt jelenti, hogy a nyilvántartott egész időtartam alatt a forgalmazók összesen 58 000 Tüskést adtak el (mindenféle színből), összesen 1 339 800 000 dollár értékben. Végül, ebből a sorból:

```
(*, *, *, *, 3521727000, 198000)
```

azt tudhatjuk meg, hogy a nyilvántartott időtartam alatt a Süni típusú személygépkocsikból összesen 198 000 példányt adtak el (színre és forgalmazóra való tekintet nélkül), és ezek értéke összesen 3 521 727 000 dollár. □

10.7.2. A kockaművelet SQL-ben

Az SQL lehetővé teszi a kockaművelet használatát a lekérdezéseinkben. Ha a `WITH CUBE` kifejezést is hozzáadjuk a csoportosító utasításhoz, akkor már nem csak az egyes csoportokhoz tartozó sorokat kapjuk vissza, hanem azokat a sorokat is, amelyek összesítéseket fejeznek ki egy vagy több, a csoportosításban szereplő dimenzióra nézve. Az eredményben azon sorok, amelyekre korábban * jelet használtunk, a `NULL` értéket tartalmazzák a * helyén.

10.32. példa. Létrehozhatunk a 10.31. példában szereplő KOCKA(Eladások)-nak nevezett adatkockának egy megvalósított formáját a következő utasítással:

```
CREATE MATERIALIZED VIEW EladásKocka AS
    SELECT modell, szín, dátum, forgalmazó,
        SUM(összért), SUM(db)
    FROM Eladások
    GROUP BY modell, szín, dátum, forgalmazó WITH CUBE;
```

Az EladásKocka nemcsak a csoportosító művelet által eredményezett sorokat adja, mint például:

```
('Tüskés', NULL, '2001-05-21', 'Undok Ubul', 45000, 2)
```

hanem azokat a KOCKA(Eladások) sorokat is, amelyek a GROUP BY záradékban felsorolt dimenziók felgörgetésével kaptunk. Néhány ilyen sor az alábbi pár sor:

```
('Tüskés', NULL, '2001-05-21', 'Undok Ubul', 152000, 7)
('Tüskés', NULL, '2001-05-21', NULL, 2348000, 100)
('Tüskés', NULL, NULL, NULL, 1339800000, 58000)
(NULL, NULL, NULL, NULL, 3521727000, 198000)
```

Emlékeztetőül a NULL egy felgörgetett dimenziót jelentett ugyanúgy, mint a * az elvont KOCKA művelet eredményében. □

A KOCKA művelet egy variánsa a ROLLUP, amely csak abban az esetben állítja elő az összesített sorokat, ha azok a csoportosító attribútumok sorozatának valamelyik záró részsorozata felett összesítenek. Ezt az opciót a csoportosító művelet után írt WITH ROLLUP kifejezéssel érhetjük el.

10.33. példa. Az Eladás adatkockájának egy részét kaphatjuk meg a ROLLUP művelet alkalmazásával:

```
CREATE MATERIALIZED VIEW EladásFelGörg AS
    SELECT modell, szín, dátum, forgalmazó,
        SUM(összért), SUM(db)
    FROM Eladások
    GROUP BY modell, szín, dátum, forgalmazó WITH ROLLUP;
```

Az EladásFelGörg a következő sorokat tartalmazza:

```
('Tüskés', 'piros', '2001-05-21', 'Undok Ubul', 45000, 2)
('Tüskés', 'piros', '2001-05-21', NULL, 3678000, 135)
('Tüskés', 'piros', NULL, NULL, 657100000, 34566)
('Tüskés', NULL, NULL, NULL, 1339800000, 58000)
(NULL, NULL, NULL, NULL, 3521727000, 198000)
```

Ugyanis ezek a csoportosító lista attribútumainak megfelelő sorrendben hátulról egymást követő dimenziókra, illetve az összes dimenzióra vonatkozó összesítések lesznek.

Az `EladásFelGörg` nem tartalmazza viszont a következő sorokat:

```
('Tüskés', NULL, '2001-05-21', 'Undok Ubul', 152000, 7)
('Tüskés', NULL, '2001-05-21', NULL, 2348000, 100)
```

Ezeknél mindkét esetben ugyanannál a dimenziónál (nevezetesen `szín`) `NULL` szerepel, de az őt követő attribútumdimenzió már nem `NULL`. □

10.7.3. Feladatok

10.7.1. feladat. Adjuk meg a KOCKA(F) és az F ténytábla méretarányát, ha F a következő tulajdonságokkal rendelkezik:

a) F-nek tíz dimenzióattribútuma van, mindegyik tíz különböző értékkel.

b) F-nek tíz dimenzióattribútuma van, mindegyik két különböző értékkel.

10.7.2. feladat. Az 10.32. ábrában tárgyalt `EladásKocka` nevű megvalósított nézet felhasználásával válaszoljuk meg a következő lekérdezéseket:

a) Adjuk meg, hogy az egyes forgalmazóknak mennyi bevételük származott a kék autók eladásából!

b) Adjuk meg, hogy „Kedves Kunigunda" a zöld Tüskésekből összesen hányat adott el!

c) Adjuk meg, hogy 2007 márciusában naponta az egyes forgalmazók a különböző színű Tüskésekből átlagosan hány darabot adtak el!

! **10.7.3. feladat.** Mennyiben segít minket az, ha a 10.33. példa `EladásFelGörg` visszagörgetését használjuk a 10.7.2. feladat összes lekérdezésében?

10.7.4. feladat. A 10.6.1. feladatban PC-k szerint rendezett adatokról beszéltünk, mint egy olyan ténytábláról, amelyhez tartozó dimenziótáblákat az attribútumai a `vásárló`, `proc`, `memória`, `lemez` és `cd` voltak. A rendezettségben szereplő PC-kről szóló adatokról a `Megrendelések` ténytábla minden egyes sorának volt egy azonosítója ezekre az attribútumokra nézve. Írjunk egy SQL-lekérdezést, amely az ehhez a ténytáblához tartozó adatkockákat állítja elő.

10.7.5. feladat. Határozzuk meg a 10.7.4. feladatban szereplő kocka azon sorait, amelyeket a következő lekérdezések megválaszolásához használnánk:

a) Adjuk meg, hogy 2007-ben havonként összesen mennyit rendeltek a különböző sebességű gépekből!

b) Listázzuk ki, hogy összesen hány számítógépet rendeltek az egyes processzor- és merevlemeztípusok (például SCSI vagy IDE) kombinációira lebontva!

c) Adjuk meg 2005 januárjától kezdve havonként a 3.0 GHz-es számítógépek átlagárát!

! **10.7.6. feladat.** A 10.32. példában említett kockasorok nem szerepeltek a 10.33. példa visszagörgetésében. Van más olyan visszagörgetés, amely tartalmazza ezeket a sorokat?

!! **10.7.7. feladat.** Amennyiben az F ténytáblázat ritka (azaz sokkal kevesebb sora van, mint a dimenziók lehetséges értékei számának a szorzata), akkor a KOCKA(F) és F táblázatok méretaránya nagyon nagy lehet. Mennyire?

10.8. Összefoglalás

- *Jogosultságok:* Biztonsági követelmények teljesítése érdekében egy SQL-rendszer sokféle jogosultságot biztosít az egyes adatbáziselemeken, mint például: a relációk lekérdezési (olvasási), beszúrási, törlési vagy módosítási jogosultságai, valamint megszorításokban a relációnak más relációkból való hivatkozásának jogosultsága, és a trigger létrehozásának a joga.

- *Engedélyezési diagramok:* Az egyes jogosultságokat azok tulajdonosai adhatják tovább egy másik felhasználónak vagy az általános felhasználónak (`PUBLIC`). Továbbadható az engedélyezési képesség is, és az így megszerzett jogosultságokat azok megszerzői tetszésük szerint átruházhatják másokra is. A jogosultságok akár vissza is vonhatók. Az engedélyezési diagram hasznos szemléltető eszköz. Segítségével könnyen számon tartható az, hogy egy adott adatelemre vonatkozóan ki milyen jogosultságokkal rendelkezik, és kitől szerezte ezeket a jogosultságokat.

- *Rekurzív lekérdezések SQL-ben:* Az SQL-ben a relációk rekurzívan is megadhatóak, azaz saját környezetükkel. Sőt több reláció is definiálható kölcsönösen rekurzívan.

- *Monotonitás:* Az SQL-rekurzióban szereplő negációknak és összesítéseknek monotonnak kell lenni, azaz egy sor hozzáadása egy relációhoz nem eredményezheti sorok törlését egy másik relációból (önmagát is beleértve). Vagy másként, egy relációt nem szabad se direkt módon, se indirekt módon (vagyis saját negáltjával vagy saját összesítésével) önmagával definiálni.

- *Objektumrelációs modell:* Egy választási lehetőség az ODL-hez hasonló tiszta objektumorientált adatbázismodellel szemben a relációs adatmodell kiterjesztése lehet a legfőbb objektumorientált lehetőségekkel. Ezek a kiterjesztések tartalmazzák a beágyazott relációkat, azaz egy reláció

attribútumainak összetett típusát (a relációt is mint típust). Egyéb kiterjesztések eljárások megadását is megengedik egy típusra, illetve azt is lehetővé teszik, hogy egy sorból egy másikra tudjunk hivatkozni hivatkozási típus használatával.

- *Felhasználói típusok SQL-ben:* Az SQL objektumrelációs adottságai az UDT (vagy *user-defined type*) körül forognak. Ezeket a típusokat megadhatjuk attribútumaik és más, a tábladeklarációknál is használt információk listájával. Sőt metódusokat is deklarálhatunk az UDT-khez.

- *UDT-típussal rendelkező relációk:* Egy reláció attribútumainak megadása helyett egy relációt deklarálhatunk úgy, hogy egy UDT-típussal rendelkezik. Ebben az esetben a sorai csak egy komponensből állnak, és ez a komponens egy UDT-objektum lesz.

- *Hivatkozási típus:* Egy attribútum típusa lehet egy hivatkozás egy UDT-re. Az ilyen attribútumok tulajdonképpen a szóban forgó UDT-objektumra mutató pointerek.

- *Objektumazonosítás UDT-ben:* Egy UDT-típussal rendelkező reláció létrehozásánál egy olyan attribútumot is deklarálunk, amely a sorok OID-ját (objektumazonosítóját) szolgáltatja. Ez a komponens egy hivatkozás a hozzá tartozó sorra. Ehhez az „OID" oszlophoz, az objektumorientált rendszerekkel ellentétben, a felhasználó is hozzáférhet, habár ez az információ nem túl sokatmondó.

- *Egy UDT komponenseinek elérése:* Az SQL megfigyelő- és változtató függvényeket nyújt egy UDT minden egyes komponenséhez. Ezen függvények visszaadnak, illetve megváltoztatnak egy megfelelő attribútumértéket, amikor meghívjuk őket a hozzájuk tartozó UDT-objektumra.

- *Az UDT rendezési függvényei:* Az objektumok összehasonlíthatósága miatt (az olyan SQL-műveletek esetén, mint a `DISTINCT`, `GROUP BY` vagy az `ORDER BY`) kellenek olyan UDT-eszközök, amelyekkel olyan függvényt tudunk megvalósítani, amellyel le tudjuk írni, hogy a szóban forgó objektumok megegyeznek-e vagy az egyik nagyobb-e a másiknál.

- *OLAP:* Az online analitikus feldolgozás olyan összetett lekérdezéseket jelent, amelyek az egész adatbázist vagy annak egy nagy részét foglalják le ugyanabban az időben. Gyakran egy elkülönített adatbázist, vagyis adattárházat, hoznak létre az ilyen lekérdezések futtatására, miközben az aktuális adatbázist csak rövid távú tranzakciók futtatására használják. (OLTP – On-Line Transaction Processing – Online Tranzakciós Feldolgozás)

- *ROLAP és MOLAP:* Ha OLAP-alkalmazás céljából építünk adattárházat, gyakran hasznos, ha az adatra egy kocka képében gondolunk, ahol a kocka dimenziói mentén az adatot más és más megvilágításban látjuk. Az

olyan rendszer, amely ezt az adatmodellt támogatja, vagy relációs szempontból tekint a kockára (ROLAP- vagy relációs OLAP-rendszerek), vagy az adatkockára specializálódik (MOLAP- vagy többdimenziós OLAP-rendszerek).

- *Csillagsémák:* Egy csillagséma esetén minden adatelemet (például egy árucikk eladását) egy relációban, a ténytáblában tárolunk, a dimenziók különböző értékeinek az értelmezéséhez (például az 1234 azonosítójú cikk milyen típusú termék?) szükséges információkat pedig az egyes dimenziókhoz tartozó dimenziótáblákban.

- *Kockaművelet:* Egy speciális művelet, a kocka a ténytábla dimenzióinak lehetséges részhalmazai mentén előösszesítést végez. Az így kibővített táblázat kicsit több helyet foglal, mint az eredeti ténytábla, de segítségével nagymértékben csökkenthető az OLAP-lekérdezések futási ideje.

- *Adatkockák SQL-ben:* A lekérdezésünk eredményét egy adatkockába is rakhatjuk a csoportosítási utasítást követő `WITH CUBE` segítségével. Az adatkocka egy részét is előállíthatjuk, ha az előző helyett `WITH ROLLUP` kifejezést írunk.

10.9. Irodalomjegyzék

Az SQL hozzáférési mechanizmusa [4]-ből, illetve [1]-ből ered.

Az SQL objektumrelációs lehetőségeinek leírásai a 6. fejezet irodalomjegyzéke alapján összegyűjthetők.

[2] az SQL-99 rekurzióra tett javaslatának a forrása. Ez az ajánlás és a monotonitási kritérium is azon a sok éven keresztül fejlesztett alapokon nyugszik (mint a rekurzió és a negáció a Datalogban), amelyet [5] tartalmaz.

A kocka műveletet [3] javasolta.

[1] R. Fagin, „On an authorization mechanism," *ACM Transactions on Database Systems* **3**:3, pp. 310–319, 1978.

[2] S. J. Finkelstein, N. Mattos, I. S. Mumick, H. Pirahesh, „Expressing recursive queries in SQL," ISO WG3 report X3H2–96–075, March, 1996.

[3] J. N. Gray, A. Bosworth, A. Layman, H. Pirahesh, „Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals," *Proc. Intl. Conf. on Data Engineering* (1996), pp. 152–159.

[4] P. P. Griffiths, B. W. Wade, „An authorization mechanism for a relational database system," *ACM Transactions on Database Systems* **1**:3, pp. 242–255, 1976.

[5] J. D. Ullman, *Principles of Database and Knowledge-Base Systems, Volume I*, Computer Science Press, New York, 1988.

III. rész

Félig-strukturált adatok modellezése és programozása

11. fejezet

A félig-strukturált adatmodell

A következőkben egy új típusú adatmodell vizsgálatába kezdünk. A félig-strukturáltnak nevezett modellt az különbözteti meg a többitől, hogy a séma az adattartalomból kikövetkeztethető, és nem elkülönített helyen kerül meghatározásra, mint a relációs és más eddig tárgyalt modelleknél. A félig-strukturált adatmodell általános tárgyalása után az elképzelés legfontosabb megvalósítását, az XML-t vizsgáljuk. Feltárjuk azokat a módszereket, amelyekkel az XML-adatok leírhatók. Valójában sémát erőszakolunk rá ezekre a „séma nélküli" adatokra. A módszerek között szerepel a DTD (Dokumentum Típus Definíció) használata és az XML sémanyelv is.

11.1. Félig-strukturált adat

A félig-strukturált adatmodell az adatbázisok világában speciális szerepet játszik:

1. Adatbázisok integrálásához alkalmas modellt kínál azzal, hogy leírható vele két vagy több adatbázisban található hasonló, de különböző sémával tárolt adat.
2. Alapjául szolgálhat olyan jelölésekhez, mint például az XML, amelyről részletesen a 11.2. alfejezet szól, és információk interneten történő megosztására használatos.

Ebben a fejezetben a félig-strukturált adatokkal kapcsolatos főbb elméleteket mutatjuk be, továbbá azt, hogy mi módon ábrázolhatjuk az információt sokkal rugalmasabban, mint a korábban látott modellekkel.

11.1.1. A félig-strukturált adatmodell-motivációk

Az eddig látott modellek – E/K, UML, relációs, ODL – mindegyike egy séma megadásából indul ki. A séma egy merev váz, amelyen belül az adatok elhelyezhetők. Ez a merevség bizonyos előnyöket is nyújt. Különösen a relációs modell

köszönheti nagymértékben sikerét annak, hogy számos hatékony implementációja létezik. Ez a hatékonyság egyrészt abból adódik, hogy a relációs adatbázisban található adatok megfelelnek a sémának, amelyet a lekérdezésfeldolgozó ismer. Például a séma rögzítésével elérhető, hogy az adatokat olyan struktúra felhasználásával tároljuk, amely hatékony válaszadásra alkalmas, ahogyan a 8.3. alfejezetben is láttuk.

Másrészt a félig-strukturált adatmodell kialakítását legfőképpen a rugalmassága motiválta. Különösen az, hogy a félig-strukturált adatnak nincs sémája, pontosabban az adat *önleíró*. Magában hordozza az információt arról, hogy milyen a sémája, de ez a séma tetszőlegesen változhat az idő múlásával és az adatbázison belül is.

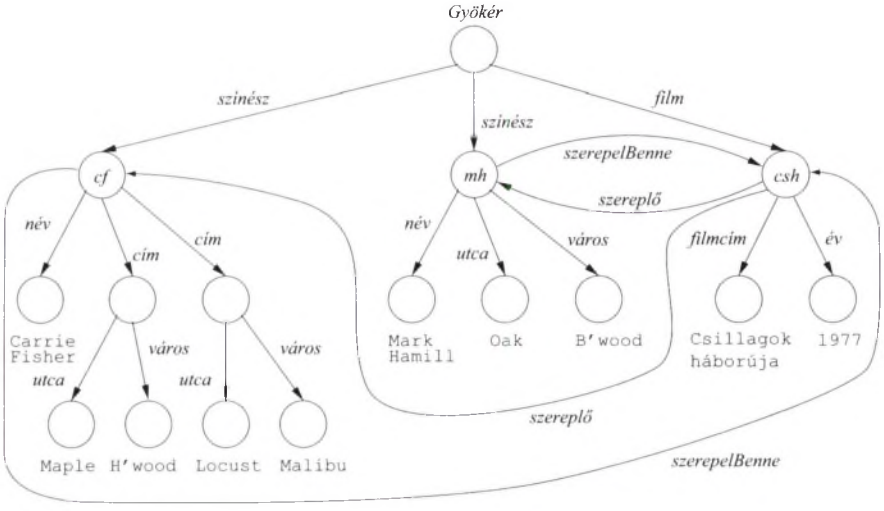
Természetesen meglepő, hogy előnyös lehet sémanélküli adatbázist készíteni, amelybe bárki bevihet adatokat, és csatolhat olyan sémainformációkat, amelyeket jellemzőnek talál az adatra. Valójában csupán néhány kisméretű információs rendszer van, mint például a Lotus Notes, amely az önleíró adatmodellt használja. Ez a megközelítés nagyon megnehezítheti a lekérdezésfeldolgozást, de jelentős előnyöket nyújt a felhasználók számára. Például karbantartunk egy adatbázist mozifilmekről félig-strukturált adatmodellt használva, majd felvehetünk egy attribútumot, amely olyasmit jelent, hogy „megnézném-e ezt a filmet?", amennyiben szeretnénk. Az attribútumoknak nem kell értéket felvenniük minden egyes filmre, sőt az sem kell, hogy egynél több filmhez legyen ilyen attribútum rendelve. Hasonlóan felvehetünk egy kapcsolatot anélkül, hogy a sémát meg kellene változtatni, vagy egynél több mozifilmpárosnál szerepelnie kellene a kapcsolatnak.

11.1.2. Félig-strukturált adatok ábrázolása

Egy *félig-strukturált adatok*at tartalmazó adatbázis csúcsok halmaza. Mindegyik csúcs *levél* vagy *belső csomópont*. Minden levélcsomóponthoz tartoznak hozzárendelt adatok, amelyek atomi típusúak, például számok vagy szövegek. A belső csúcsok rendelkeznek egy vagy több kimenő éllel. Minden élnek van *címké*je, amely azt jelzi, hogy milyen módon kapcsolódik a kiinduló- és a célcsomópont egymáshoz. A *gyökércsomópont* olyan belső csúcs, amelynek nincs bemenő éle, és ez reprezentálja a teljes adatbázist. Az összes csúcs elérhető a gyökércsúcsból, de a gráf szerkezete nem feltétlenül fa.

11.1. példa. A 11.1. ábra egy mozifilmeket és színészeket tartalmazó félig-strukturált adatbázist ábrázol. Az ábra tetején látható a *Gyökér* nevű csúcs, ez az a pont, amelyen keresztül az összes adatbázisban szereplő információ elérhető. A fő objektumok és egyedek – ebben az adatbázisban színészek és filmek – a gyökérelem gyermekeiként vannak ábrázolva. Ezenkívül számos levélelemet láthatunk. Például a bal szélső levél címkéje `Carrie Fisher`, a jobb szélsőé 1977. Szintén sok belső csomópont szerepel az ábrán. Három csomópont, a *cf*, *mh* és *csh* nevűek rendre „Carrie Fisher"-t, „Mark Hamill"-t és a „Csillagok háborúja"-t jelképezik. Ezen csomópontok elnevezése nem a modell része, csupán azért helyeztük el őket az ábrán, hogy a leírásban hivatkozhassunk azokra a csomó-

pontokra, amelyek egyébként nem lennének névvel ellátva. Felfoghatjuk az *csh* nevű csomópontot úgy is, mint a „Csillagok háborúja" fogalom ábrázolását: a címe, a megjelenés éve és egyéb információk, amelyek nem láthatók, például a film hossza vagy a szereplői. □



11.1. ábra. Színészeket és filmeket leíró félig-strukturált adat

Egy C címke, amely az N csomópontból kiindulva az M csomópontba megy, kétféle szerepet játszhat az ábrázolásban:

1. Lehetséges, hogy N egy csomópont, amely egy objektum vagy egyed, M pedig valamely attribútumának a jelölése. Ekkor C jelentése az attribútum neve.

2. Előfordulhat, hogy N és M is egy-egy objektum vagy egyed. Ilyenkor C a köztük lévő reláció nevét fejezi ki.

11.2. példa. Vizsgáljuk meg ismét a 11.1. ábrát. A *cf*-fel jelölt csomópontot tekinthetjük úgy, mint a Carrie Fisher-t reprezentáló `Színész` objektumot. A csomópontból kiindul egy él, amelynek a címkéje *név*, amely a `név` attribútumot jelöli, és egy olyan levélcsúcshoz vezet, amely tartalmazza a pontos nevet. Látható továbbá két él, amelyek a `cím` szöveggel vannak címkézve. Ezek az élek egy névtelen csomóponthoz vezetnek, amelyet tekinthetünk Carrie Fisher két lakcímének ábrázolásaként. Nincs semmilyen sémánk, amely megmondhatná, hogy vajon lehet-e egy színésznek egynél több lakcíme, egyszerűen csak felveszünk két címeket tartalmazó csomópontot a gráfban, ha azt helyesnek érezzük.

Figyeljük meg a 11.1. ábrán, hogy mindkét csomópont rendelkezik kimenő élekkel, amelyeken az *utca* és a *város* címkék vannak. Továbbá ezek az élek

levélcsúcsokba vezetnek, melyek atomi értékeket tartalmaznak. A *cím* csúcsra tekinthetünk olyan struktúraként vagy objektumként, melynek két mezője van, az *utca* és a *város*. A félig-strukturált modellben azonban lehetséges új komponensek hozzáadása vagy egyéb hiányzó mezők, például irányítószám megadása.

A 11.1. ábrán az élek egy másik típusa is megtalálható. Például a *cf* csomópontnak van egy kimenő éle, amely az *csh* csúcsba mutat és a *szerepelBenne* nevet viseli. Az *mh* nevű (Mark Hamillt leíró) csomópont is rendelkezik ilyen éllel, az *csh* csomópontból pedig kiindul két él a *cf* és az *mh* felé, amelyek neve *szereplő*. Ezek az élek reprezentálják a *szerepelBenne* relációt a színészek és a filmek között. □

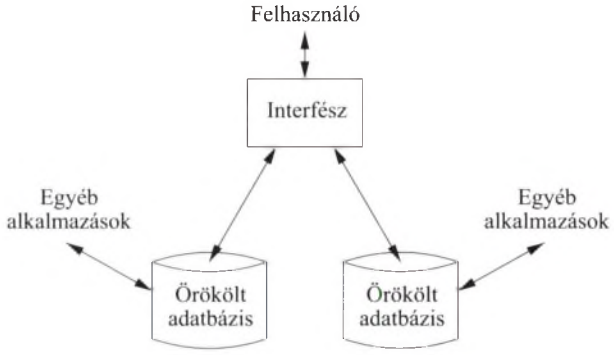
11.1.3. Információintegráció félig-strukturált adatokon keresztül

A félig-strukturált adattípust a rugalmassága és az önleíró tulajdonsága két alkalmazási területen teszi fontossá. A 11.2. alfejezetben az adatcserében betöltött szerepét fejtjük ki, de előtte a jelen fejezetben az információintegráció egy eszközeként vizsgáljuk meg. Az adatbázisok elszaporodásával együtt mindennapos követelmény lett az, hogy két vagy több adatbázis tartalmát úgy lehessen elérni, mintha egyetlen adatbázisban lennének. Például vállalatok egyesülnek, de mindkettőnek megvan a saját személyi nyilvántartása, saját adatbázisa van az eladási adatokhoz, a készletnyilvántartáshoz, a terméktervezéshez és esetleg más egyéb adatokhoz. Ha a megfelelő adatbázisok sémája azonos, akkor az összekapcsolásuk egyszerű, például vehetjük a két reláció sorainak unióját, melyek azonos sémával rendelkeznek és azonos szerepet töltöttek be az eredeti adatbázisokban.

A valóság azonban ritkán ennyire egyszerű. Egymástól függetlenül fejlesztett adatbázisok esetén valószínűtlen, hogy megosztják egymással a sémát, még akkor is, ha ugyanarról a dologról készülnek, mint például az alkalmazottak. Például, egy alkalmazottakról készült adatbázisban tárolják a házastárs nevét, egy másikban viszont nem. Egyikben lehetséges, hogy több, különböző lakcím, telefonszám és e-mail előfordulhat, míg a másikban pontosan egy lehet belőlük. Vannak, amelyek a tanácsadókat is alkalmazottként kezelik, mások pedig nem. Az egyik adatbázis lehet relációs, a másik pedig objektumorientált.

A probléma még bonyolultabbá válhat attól, hogy az adatbázisok felhasználása során számos programot készítettek az adatokra, amelyek teljes kikapcsolása és az általuk használt összes adat átmásolása, áttranszformálása az új adatbázisba lehetetlen még akkor is, ha találunk hatékony módot az adatok áttöltésére a két séma között. Ezt a helyzetet gyakran *adatbázis-öröklési problémának* nevezik. Ha egy adatbázis létezik egy ideig, akkor lehetetlenné válik az elválasztása a ráépülő programoktól, tehát soha többé nem lehet üzemen kívül helyezni.

Az adatbázis-öröklési probléma egy lehetséges megoldása látható a 11.2. ábrán. Az ábrán két örökölt adatbázist láthatunk egy közös interfésszel, amelyekhez számos rendszer csatlakozhat. Az örökölt rendszerek egyike sem változik, így kiszolgálhatják a megszokott alkalmazásaikat.



11.2. ábra. Két megörökölt adatbázis integrációja félig-strukturált adatokat támogató interfészen keresztül

Az integráció rugalmassága érdekében az interfész támogatja a félig-strukturált adatokat, és a felhasználónak lehetősége van az interfészen keresztül lekérdezéseket végrehajtani az ilyen adatszerkezetnek megfelelő lekérdező nyelven. A félig-strukturált adat előállítható a forrásadatok áttranszformálásával, ún. illesztőmodulokon („adaptereken") keresztül, amelyek mindegyike arra lett tervezve, hogy a forrásadatokat félig-strukturált szerkezetbe transzformálja.

Egy másik megközelítés szerint a félig-strukturált adat nem létezik az interfészen. A felhasználó az interfészre írja a kérdését úgy, mintha az egy félig-strukturált adat lenne. A válaszadás során az interfész a kérdéseket átalakítja a forrásoknak megfelelő formájúvá úgy, hogy mindegyik a forrásadat sémájára vonatkozzon.

11.3. példa. A 11.1. ábrán szereplő adatbázis lehet annak az eredménye, hogy a színészekről szóló információkat különböző adatbázisokból gyűjtötték össze. Figyeljük meg, hogy a Carrie Fisherhez tartozó lakcím adat használ egy címfogalmat, és a cím két részre, utcanévre és városra bomlik. Ez a felépítés nagyjából ahhoz az adathoz hasonló, mint amilyet a `Színészek(név, cím(utca, város))` beágyazott relációséma leír.

Másfelől a Mark Hammilhez tartozó cím nem rendelkezik címfogalommal, csak egy utca és egy város tulajdonsággal. Ez az információ származhat egy olyan sémából, amelynek a szerkezete `Színészek(név, utca, város)`, és amely egyetlen címet tud nyilvántartani egy színészhez. Néhány más, itt nem tárgyalt variáció is bemutatható a sémáról, amelyet a 11.1. ábrán látható egyszerű példa nem tükröz, de bekövetkezhet, ha a mozifilmek adatait több forrásból kapjuk, beleértve a nem kötelező filmtípus információt, a rendezőt, a kiadót, a bevételt és információkat arról, hogy a filmet jelenleg hol játsszák. □

11.1.4. Feladatok

11.1.1. feladat. Mivel nem létezik olyan séma, amivel félig-strukturált adatmodell tervezhető, így nem adhatunk olyan feladatot, amely sémák tervezését írja elő különböző feladatok leírására. Mégis, a következő feladatokban azt kell bemutatni, hogy milyen adatszervezéssel lehet egyes állításokat, tényeket ábrázolni.

a) Adjuk hozzá a 11.1. ábrához a következő tényt: A *Csillagok háborúja* rendezője George Lucas, producere Gary Kurtz.

b) Adjunk hozzá a 11.1. ábrához információkat arról, hogy Carrie Fisher és Mark Hamill szerepelt *A birodalom visszavág* és az *Jedi visszatér* című filmekben.

c) Az előző pontban megfogalmazottakhoz adjunk hozzá információt a filmstúdióról (Fox), ahol a filmek készültek és a stúdió címéről (Hollywood).

11.1.2. feladat. Mutassuk be, hogyan lehet félig-strukturált adatmodellel ábrázolni bankokról és ügyfelekről szóló információkat hasonlóan ahhoz, amit a 4.1.1. feladatban láttunk.

11.1.3. feladat. Mutassuk be, hogyan lehet félig-strukturált adatmodellel ábrázolni tipikus adatokat játékosokról, csapatokról és szurkolókról úgy, ahogyan a 4.1.3. feladatban volt látható.

11.1.4. feladat. Mutassuk be, hogyan lehet félig-strukturált adatmodellel ábrázolni családi kapcsolatokat ahhoz hasonlóan, mint a 4.1.6. feladatban volt látható.

! **11.1.5. feladat.** Az UML és a félig-strukturált adatmodell egyaránt „grafikus" természetű abban az értelemben, hogy mindketten csúcsokat és címkéket használnak, ill. összekötőket a csúcsok között a kifejezések leírására. Mégis alapvető különbség van a két modell között. Mi ez?

11.2. XML

XML *(Extensible Markup Language – Kiterjesztett Leíró Nyelv)* egy címke alapú jelölésrendszer, amelyet eredetileg dokumentumok leírására terveztek, hasonlóan a HTML-hez. Napjainkban az XML-lel leírt adat sokféle módon ábrázolható. Ebben a fejezetben azonban az XML-adatot úgy tekintjük, mint egy vagy több dokumentumban ábrázolt adatot. Amíg a HTML-jelölők a dokumentumok megjelenését írják le – például, hogy mi jelenjen meg dőlt betűtípussal, vagy hogy mik egy lista elemei – addig az XML-jelölők a dokumentum egyes részeinek a jelentéséről adnak információt.

Ebben a fejezetben az XML-alapismereteket tekintjük át. Látni fogjuk, hogy lineáris szerkezetben magában foglalja azt a struktúrát, amelyet a félig-strukturált adatok gráfjai írtak le a 11.1. alfejezetben. Lényegében a jelölők

játsszák ugyanazt a szerepet, mint az élek elnevezései a félig-strukturált adatot ábrázoló gráfban.

11.2.1. A jelölők jelentése

A jelölők az XML-ben egyszerű szövegek csúcsos zárójellel körülvéve, azaz <...> ugyanúgy, mint a HTML-ben. Úgy, mint a HTML-ben, a jelölők összeillő párokból állnak, amelynek része egy *nyitó jelölő*, például <valami> és egy *záró jelölő*, ugyanazzal a szóval és egy / jellel, például </valami>. Egy összeillő pár, <valami> és </valami> két része között elhelyezkedhet szöveg, beágyazott HTML-részlet, és tetszőleges számú beágyazott XML-jelölőelempár. Egy jelölőelempár és minden benne szereplő összetevő együttesen alkot egy XML-*elemet*.

Egy egyszerűsített nyitó jelölő, amelyhez nem tartozik záró jelölő, szintén megengedett az XML-ben. Ilyen forma használatakor az elem végén, a zárójel előtt szerepel egy / jel, például <valami/>. Ez az elem nem tartalmaz semmilyen szöveget vagy más beágyazott elemet. Ettől függetlenül attribútumai lehetnek (lásd 11.2.4. alfejezet).

11.2.2. XML sémával és séma nélkül

Az XML-t úgy tervezték, hogy két, némiképp különböző módon lehessen használni:

1. A *jólformált* XML megengedi saját jelölők megalkotását hasonlóan az élek feliratihoz a félig-strukturált adat esetében. Ez a mód nagyjából megfelel a félig-strukturált adatnak, amely nem rendelkezik előredefiniált sémával és minden dokumentumban szabadon használható bármely jelölő, amelyet a dokumentum készítője szükségesnek tart. Természetesen a jelölők egymásba ágyazásával kapcsolatos szabályt be kell tartani, különben a dokumentum már nem jólformált.

2. Az *érvényes* XML-hez tartozik egy „DTD", vagy „Dokumentum Típus Definíció" (lásd 11.3. alfejezet), amely meghatározza a megengedhető jelölőket, és nyelvtant szolgáltat az egymásba ágyazáshoz. Ez az XML-forma félúton van a szigorú sémával leírt modellek, mint a relációs modell, és a félig-strukturált adatmodell teljesen séma nélküli világa között. Ahogyan a 11.3. alfejezetben látni fogjuk, a DTD-k nagyobb rugalmasságot engednek az adatokban, mint a hagyományos sémák; a DTD-k például általában megengednek opcionális mezőket vagy hiányzó mezőket.

11.2.3. Jólformált XML

A minimumkövetelmény egy jólformált XML-dokumentummal szemben az, hogy a dokumentum egy deklarációval kezdődjön, amelyből kiderül, hogy XML-dokumentumról van szó és van egy *gyökéreleme*, amely a dokumentum törzsét

alkotja. Tehát egy jólformált XML-dokumentum felépítése az alábbihoz hasonló:

```
<? xml version="1.0" encoding="utf-8" standalone="yes" ?>
<ValamiJelölő>
  ...
</ValamiJelölő>
```

Az első sor jelenti azt, hogy a fájl egy XML-dokumentum. Az UTF-8 kódolás (Unicode Transformation Format) megszokott választás a dokumentumok karakterkódolásához, mert kompatibilis az ASCII-karakterekkel és az ASCII-karakterek ábrázolásához csak egy bájtot használ. A `standalone="yes"` attribútumérték-pár jelzi, hogy ehhez a dokumentumhoz nem tartozik DTD, azaz ez egy jólformált XML. Vegyük észre, hogy ez a deklarációs rész speciális jelölőkkel van körbevéve, a `<?...?>` jelekkel. Ennek a dokumentumnak a gyökéreleme a `<ValamiJelölő>`.

```
<? xml version = "1.0" encoding = "utf-8"
                              standalone = "yes" ?>

<FilmSzínészAdat>
    <Színész>
        <Név>Carrie Fisher</Név>
        <Cím>
            <Utca>123 Maple St.</Utca>
            <Város>Hollywood</Város>
        </Cím>
        <Cím>
            <Utca>5 Locust Ln.</Utca>
            <Város>Malibu</Város>
        </Cím>
    </Színész>
    <Színész>
        <Név>Mark Hamill</Név>
        <Utca>456 Oak Rd.</Utca>
        <Város>Brentwood</Város>
    </Színész>
    <Film>
        <FilmCím>Csillagok háborúja</FilmCím>
        <Év>1977</Év>
    </Film>
</FilmSzínészAdat>
```

11.3. ábra. Színészeket és filmeket tartalmazó XML-dokumentum

11.4. példa. A 11.3. ábrán egy XML-dokumentum látható, amely nagyjából a 11.1. ábra adatainak – pontosabban a félig-strukturált adat faszerű részének – felel meg: a gyökérnek és az összes elemnek és élnek, kivéve azokat az „oldalirányú" éleket, amelyek a *cf*, *mh* és *csh* csomópontok között húzódnak. A 11.2.4. alfejezetben azt is látni fogjuk, hogy azok az élek milyen módon reprezentálhatók.

A gyökérelem a `FilmSzínészAdat`. Ezen az elemen belül látható két másik elem. Mindkettő a `<Színész>` jelölővel kezdődik és a hozzá tartozó `</Színész>` jelölővel végződik. Mindegyik elemen belül van egy gyermekelem, amely a színész nevét adja meg. A Carrie Fisherhez tartozó elem szintén két alárendelt elemmel rendelkezik, amelyek egy-egy lakásának címét adják meg. Ezek az elemek körbe vannak véve egy `<Cím>` nyitó és záró jelölővel. A Mark Hamillhez tartozó elemben csak az utca és a város nevét tartalmazó gyermekelemek vannak, és nem jelenik meg az ezeket csoportosító `<Cím>` jelölő. Ez a különbség már megjelent a 11.1. ábrán is. Szintén látható az ábrán egy `<Film>` jelölő a hozzá tartozó záró jelölővel. Ennek az elemnek is vannak beágyazott elemei, egy a címének, egy az évnek, amikor a film készült.

```
<Színész>
    <Név>Mark Hamill</Név>
    <Utca>456 Oak Rd.</Utca>
    <Város>Brentwood</Város>
    <Film>
        <FilmCím>Csillagok háborúja</FilmCím>
        <Év>1977</Év>
    </Film>
    <Film>
        <FilmCím>A birodalom visszavág</FilmCím>
        <Év>1980</Év>
    </Film>
</Színész>
```

11.4. ábra. A filmek beágyazása a színész elembe

Itt jegyezzük meg, hogy a 11.3. ábrán nem jelenik meg a színészek és filmek közötti szerepelBenne reláció. Jelölhetjük egy színész filmjeit a hozzá tartozó elemben, felvéve a filmek címét és a készítésük évét. A 11.4. ábrán erre az ábrázolásra láthatunk példát. □

11.2.4. Attribútumok

Ahogyan a HTML-ben, az XML-ben is lehetnek egy elemnek *attribútum*ai (név-érték párok) a nyitó jelölőjében. Az attribútumok használata alternatív módszer a félig-strukturált adat levélelemeinek ábrázolására. Az attribútumok ugyanúgy, mint a jelölők, címkézett éleket jelölhetnek a félig-strukturált adat gráfjából.

Az attribútumok továbbá használhatók a 11.1. ábrán látott „mellék" élek leírására is.

11.5. példa. Az *csh* csomópont *filmcím* és az *év* gyermekelemei közvetlenül a `<Film>` elemben is ábrázolhatók a beágyazott elemek helyett. Azaz a 11.3. ábrán a `<Film>` elemet kicserélhetjük az alábbira:

```
<Film év=1977><FilmCím>Csillagok háborúja</FimCím></Film>
```

Mindkét gyermekelemet helyettesíthetjük attribútumokkal:

```
<Film FilmCím = "Csillagok háborúja" év = 1977></Film>
```

vagy akár:

```
<Film FilmCím= "Csillagok háborúja" év = 1977 />
```

Ez utóbbi esetben egy egyszerűsített jelölőformát használtunk záró jelölő nélkül, amely a végén látható / jelből derül ki. □

11.2.5. Attribútumok, amelyek összekapcsolnak elemeket

Az attribútumok egyik fontos felhasználása, amikor olyan, a félig-strukturált adatban jelen lévő kapcsolatokat ábrázolunk velük, amelyek nem faszerkezetet alkotnak. A 11.3.4. alfejezetben látni fogjuk, hogyan hozhatunk létre attribútumokat, amelyek az elemeik azonosítói. Szintén látni fogjuk, hogyan kell olyan attribútumokat készíteni, amelyek hivatkoznak ezekre az azonosítókra. Egyelőre nézzünk egy példát ezek használatának bemutatására.

11.6. példa. A 11.5. ábra tekinthető a 11.1. gráfon ábrázolt félig-strukturált adat pontos XML-reprezentációjának. Azonban a pontos értelmezéshez elegendő információval kell rendelkeznünk a sémáról, például tudnunk kell, hogy a `SzínészId` egy azonosítója annak az elemnek, amelyben előfordul. Azaz `cf` az első színész elem azonosítja (Carrie Fisheré), az `mh` a második színész elem azonosítója (Mark Hamillé). Hasonlóan létre kell hoznunk a `FilmId` attribútumot a `Film` elemen belül, hogy azonosítható legyen vele az elem. Így tehát az `csh` lesz az azonosítója a 11.5. ábrán látható egyetlen `<Film>` elemnek.

Továbbá a sémának ki kell fejeznie, hogy a `<Színész>` elemek `szerepelBenne` attribútuma, illetve a `<Film>` elem szereplője attribútuma hivatkozás egy vagy több azonosítóra. Tehát a `szerepelBenne` attribútumok `csh` értéke a két `<Színész>` elem esetében azt jelenti, hogy mind Carrie Fisher, mind Mark Hamill játszott a *Csillagok háborújában*. Hasonlóan a `cf` és `mh` értéklista a `<Film>` elem szereplője attribútumában azt fejezi ki, hogy ezek a színészek szerepeltek a *Csillagok háborújában*. □

```
<? xml version = "1.0" encoding = "utf-8"
                            standalone = "yes" ?>

<FilmSzínészAdat>
    <Színész SzínészId = "cf" szerepelBenne = "csh">
        <Név>Carrie Fisher</Név>
        <Cím>
            <Utca>123 Maple St.</Utca>
            <Város>Hollywood</Város>
        </Cím>
        <Cím>
            <Utca>5 Locust Ln.</Utca>
            <Város>Malibu</Város>
        </Cím>
    </Színész>
    <Színész SzínészId = "mh" szerepelBenne = "csh">
        <Név>Mark Hamill</Név>
        <Utca>456 Oak Rd.</Utca>
        <Város>Brentwood</Város>
    </Színész>
    <Film FilmId = "csh" szereplő = "cf", "mh">
        <FilmCím>Csillagok háborúja</FilmCím>
        <Év>1977</Év>
    </Film>
</FilmSzínészAdat>
```

11.5. ábra. A szerepelBenne információ felvétele az XML-dokumentumba

11.2.6. Névterek

Vannak olyan helyzetek, amikor az XML-adatok jelölői két vagy több forrásból származnak, ennek következtében az elnevezések ellentmondásosak lehetnek. Például nem szeretnénk összekeverni egy szövegben használt HTML-jelölőt egy XML-jelölővel, amely egy szöveg jelentését határozza meg. A 11.4. alfejezetben látni fogjuk, hogyan fordulhat elő, hogy szükség legyen egy XML-séma leírásához két különböző szótár használatára. Annak érdekében, hogy elkülöníthessük a szótárak jelölőit egy dokumentumon belül, a jelölők egy csoportjához *névtereket* rendelünk.

A nyitó jelölőbe felvett `xmlns` attribútummal megadhatjuk, hogy egy jelölőt vagy egyes attribútumait egy megadott névtér részeként kell értelmezni. Az `xmlns` attribútumot használhatjuk a nyitó jelölőjében. Ezekre az attribútumokra egy speciális formát használunk:

`xmlns:`*név*`="`*URI*`"`

Azon az elemen belül, amelyben felvettük ezt az attribútumot, a *név* szócska bármely jelölőt módosíthat olyanra, amely a név névtérhez tartozik. Azaz készíthetünk *minősített* neveket a *név:jelölő* formával, ahol a *név* annak a névtérnek a neve, amelyhez a *jelölő* tag tartozik.

Az URI (Uniform Resource Identifier – Egységes Erőforrás Azonosító) tipikusan egy URL, amely egy olyan dokumentumra mutat, amelyben le van írva a használt névtér jelölőinek jelentése. Ez a leírás nem feltétlenül formális, lehet csupán egy informális szöveg a típussal kapcsolatos elvárásokról. Előfordulhat, hogy valójában nem is találunk a jelölőn semmit, mert csak annyi volt a célunk, hogy az azonos nevű jelölők között különbséget tegyünk.

11.7. példa. Tegyük fel, hogy a 11.5. ábra `FilmSzínészAdat` elemében bizonyos jelölők a dokumentumban definiált `infolab.stanford.edu/movies` névtérhez tartoznak. Adhatunk nevet ennek a névtérnek, például az `md` nevet, az alábbi nyitó jelölő használatával:

```
<md:FilmSzínészAdat xmlns:md=
    "http://infolab.stanford.edu/movies">
```

A célunk az, hogy a `FilmSzínészAdat` is ennek a névtérnek a része legyen, ezért felvesszük hozzá az `md:` prefixumot, és ugyanígy a záró jelölőben is az `</md:FilmSzínészAdat>` szöveget használjuk. Ezen az elemen belül megtehetjük, hogy más jelölőket is ehhez a névtérhez rendelünk oly módon, hogy a nyitó és záró jelölőikhez illesztjük az `md:` előtagot. □

11.2.7. Az XML és az adatbázisok

Az XML-ben kódolt információt nem minden esetben tervezik úgy, hogy az egy adatbázisban tárolható legyen. A számítógépek számára megszokottá vált, hogy az interneten XML-elemek formájában osszanak meg egymással adatokat. Ezeknek az üzeneteknek az élettartama meglehetősen rövid annak ellenére, hogy készülhettek egy adatbázis valamely sorából, illetve kerülhetnek a fogadó félnél is adatbázisba. Például a 11.5. ábra XML-adatai átalakíthatók sorokká és beilleszthetők a korábbi példáink `FilmSzínész` és `SzerepelBenne` relációiba.

Habár az XML egyre több szerepet kap, az adatok hagyományosan relációs adatbázisokban kerülnek tárolásra. Például ahogyan a 11.1.3. alfejezetben tárgyaltuk, hogyan integrálhatók több vállalat információi egyetlen közös adatbázisba. Az XML fontos tényező lehet ezeknek a nézeteknek a megjelenítésében, alternatívájaként azoknak a nézeteknek, amelyek relációkból és objektumosztályokból állnak. Az integrált nézetek azután lekérdezhetők valamely speciális XML lekérdező nyelvvel, amelyekkel a 12. fejezetben fogunk találkozni.

Amikor XML-t tárolunk adatbázisban, feltétlenül foglalkoznunk kell a hatékony adathozzáférés követelményével, különösen akkor, ha nagyméretű XML-dokumentumokkal vagy sok kisebb méretűvel kell dolgozni[1]. Egy relációs

[1] Emlékezzünk rá, hogy az XML-adatnak nem kell feltétlenül dokumentum formában lennie (azaz fejléc egy gyökérelemmel). Például az XML-adat lehet elemek sora fejlécek nélkül. Ennek ellenére a továbbiakban is XML-adatokat értünk a dokumentum kifejezés alatt.

adatbázis-kezelő rendszerben használhatunk indexeket és egyéb eszközöket, amelyekkel hatékonnyá tehetjük az adathozzáférést, ahogyan azt a 8.3. alfejezetben láttuk. Kétféle úton biztosíthatjuk az XML-adatokhoz való hatékony hozzáférést:

1. Az XML-adatokat elemzett formában tároljuk, és biztosítunk egy eszköztárat az adatban való navigáláshoz. Erre a két leggyakoribb megoldás a SAX (Simple API for XML – Egyszerű XML API) vagy a DOM (Document Object Model – Dokumentum Objektum Modell).

2. A dokumentumokat és elemeiket relációkként ábrázoljuk, és egy hagyományos adatbázis-kezelő rendszert használunk a tároláshoz.

Annak érdekében, hogy az XML-t relációkban tárolhassuk, minden elemet el kell látnunk egy egyedi azonosítóval. Egy dokumentum esetében ez lehet az URL-je vagy az elérési útvonala a fájlrendszerben. Egy lehetséges relációséma az alábbi:

```
DokGyökér(dokId, gyökérElemId)
GyermekElem(szülőId, gyerekId, pozíció)
ElemAttribútum(elemId, név, érték)
ElemÉrték(elemId, érték)
```

Ez a séma alkalmas azon dokumentumokhoz, amelyek teljesítik azt a kritériumot, hogy egy elem vagy csak szöveget, vagy csak gyermekelemeket tartalmaz. A *vegyes összetételű* elemek beillesztése a modellbe a feladatokra marad.

A `DokGyökér` nevű első reláció összekapcsolja a dokumentumok azonosítóit a gyökérelemeik azonosítóival. A második reláció, a `GyermekElem` minden elemet („szülő") összekapcsol minden egyes közvetlen leszármazottjával („gyerek"). A reláció harmadik attribútuma megadja a gyermekelem helyét a szülő többi gyermekeleme között.

Az `ElemAttribútum` reláció összekapcsolja az elemeket az attribútumaikkal. Minden elem minden attribútumához megadjuk a nevét és az értékét. Végül az `ElemÉrték` relációban tároljuk el azokat az elemeket, amelyek nem tartalmaznak további elemeket, legfeljebb egyszerű szöveget.

Kisebb problémát jelent, hogy az attribútumok értékei különböző típusúak lehetnek, például egészek vagy karakterláncok, amíg a relációs adatbázis mezőinek egyedi típusa van. Az attribútumok `érték`-eit mindig kezelhetjük szövegként, emellett értelmezhetjük úgy, mintha egész számok vagy más meghatározott típusok lennének akkor, amikor feldolgozzuk az adatokat. Vagy az utóbbi két relációból készíthetünk több példányt, amelyek mindegyikében különböző adattípusokat tárolunk.

11.2.8. Feladatok

11.2.1. feladat. Ismételjük meg a 11.1.1. feladatot XML használatával.

11.2.2. feladat. Mutassuk meg, hogy bármely reláció ábrázolható XML-alakban. *Segítség:* készítsünk egy elemet minden sorhoz és minden komponenséhez egy gyermekelemet.

! **11.2.3. feladat.** Hogyan ábrázolható egy üres elem az adatbázisban (amelynek sem szöveges tartalma, sem gyermekeleme nincs) a 11.2.7. alfejezetben bemutatott sémával?

! **11.2.4. feladat.** A 11.2.7. alfejezetben megadtunk egy sémát azon dokumentumok ábrázolására, amelyek nem tartalmaznak kevert adatot, azaz olyan elemet, amelynek egyaránt van gyermekeleme, és szöveges tartalma (`#PCDATA`) is. Mutassuk meg, hogyan kell módosítni a sémát, ha az elemeknek kevert tartalma is van.

11.3. Dokumentumtípus-definíció

Egy XML-dokumentumokat feldolgozó számítógép számára nagy segítség, ha rendelkezik valamiféle sémaleírással a dokumentumokhoz. Hasznos tudni, hogy milyen elemek fordulhatnak elő a dokumentumhalmazban, és ezek az elemek hogyan lehetnek egymásba ágyazva. A séma leírása egy nyelvtanszerű szabályhalmazból áll, amelyeket *dokumentumtípus-definíció*nak (Document Type Definition) vagy DTD-nek nevezünk. Az eredeti elképzelés az volt, hogy minden vállalat vagy közösség, amelyik meg szeretne osztani valamilyen adatot, készít hozzá egy DTD-t, amely leírja a megosztott adat szerkezetét, létrehozva egy megosztott nézetet az elemeik jelentéséről. Például létrehozható DTD a proteinek felépítéséről, autók vételéről/eladásáról stb.

11.3.1. A DTD formális megadása

A DTD-struktúra nagyjából az alábbi:

```
<!DOCTYPE gyökérjelölő [
    <!ELEMENT elemnév (komponensek)>
        további elemek
]>
```

A nyitó *gyökérjelölő* és a hozzá tartozó záró jelölő közrezárja azt a dokumentumot, amelyik a DTD szabályait kielégíti. Az `!ELEMENT` kulcsszóval bevezetett elemleírások adják meg a dokumentumban szereplő elemet és bezárójelezve a „komponenseit". Ezek olyan összetevők, amelyeknek szerepelniük kell a jellemzett elemleírásban, vagy előfordulhatnak benne. Az összetevőkkel kapcsolatos pontos követelmények megjelenítését a következőkben tekintjük át.

Két fontos speciális komponens van:

1. (#PCDATA) („parsed character data" – értelmezett karakteres adat) az elem neve után azt jelenti, hogy az elem értéke szöveges, és nincsenek további beágyazott elemei. Ez az adat tekinthető úgy, mint egy HTML-szöveg, tartalmazhat formázási információkat, és a speciális karaktereket maszkolni kell – mint például az <-nek, – csakúgy, mint a HTML-kódokat. Például az

```
<!ELEMENT Cím(#PCDATA)>
```

sor jelentése az, hogy a <Cím> és a </Cím> jelölők között csak karaktersorozat fordulhat elő. Habár beágyazott elemek már nem szerepelhetnek az XML-ben, HTML-szöveg például megjelenhet ebben a részben.

2. Az EMPTY kulcsszó zárójelek nélkül azt jelenti, hogy a leírt elem nem rendelkezik záró jelölővel. Ezeknek az elemeknek nincsenek beágyazott elemeik, sem szöveges tartalmuk. Például az

```
<!ELEMENT Valami EMPTY>
```

sor azt jelenti, hogy a Valami jelölő egyetlen lehetséges alakja: <Valami/>.

11.8. példa. A 11.6. ábrán láthatunk egy DTD-t színészekhez[2]. A DTD neve és a gyökéreleme a Színészek. Az első definíció azt mondja ki, hogy <Színészek>...</Színészek> elempáron belül nulla, egy vagy több Színész elemet fogunk találni, melyek mindegyike egy-egy színészt jelent. A darabszámot a * jel határozza meg a (Színész*) leírásban, és a jelentése az, hogy „nulla, egy vagy több", azaz „tetszőleges darabszámú".

A második, Színész nevű elemről azt mondjuk ki, hogy három beágyazott elemet tartalmaz: Név, Cím és Filmek. A beágyazott elemek sorrendje kötött, és mindegyiknek meg kell jelennie. A Cím mellett szereplő + jel azt mondja, hogy a Cím elemből egy vagy több tartozhat a színészhez, azaz tetszőleges számú lehet, de legalább egynek mindenképpen kell lennie. A Név elemet szövegként definiáljuk. A negyedik sorból az derül ki, hogy a Cím két leszármazott elemből áll, az Utca és a Város nevűekből, ebben a sorrendben.

A Filmek elemet úgy definiáljuk, hogy nulla, egy vagy több Film elemet tartalmaz, ismét a * jelenti azt, hogy tetszőleges számú elem jelenhet meg. Minden Film-et úgy definiálunk, hogy tartalmazzon egy FilmCím és egy Év adatot, amelyek egyszerű szöveges típusúak. A 11.7. ábrán egy olyan dokumentum látható, amely megfelel a 11.6. ábrán látható DTD-nek. □

Egy E elem komponensei általában más elemek. Ezeknek mindenképpen az $<E>$ és $</E>$ jelölők között kell elhelyezkedniük. Ezen túl azonban több olyan műveletünk van, amelyek megszabhatják ezeknek a beágyazott elemeknek a számát.

[2] A 11.3. ábrán látható színészeket és mozifilmeket tartalmazó XML-dokumentum nem úgy készült, hogy megfeleljen ennek a DTD-nek.

```
<!DOCTYPE Színészek [
    <!ELEMENT Színészek(Színész*)>
    <!ELEMENT Színész (Név, Cím+, Filmek)>
    <!ELEMENT Név (#PCDATA)>
    <!ELEMENT Cím (utca, város)>
    <!ELEMENT Utca (#PCDATA)>
    <!ELEMENT Város (#PCDATA)>
    <!ELEMENT Filmek (Film*)>
    <!ELEMENT Film (FilmCím, Év)>
    <!ELEMENT FilmCím (#PCDATA)>
    <!ELEMENT Év (#PCDATA)>
]>
```

11.6. ábra. Filmszínészeket leíró DTD

1. Az elem után írt * jel azt jelenti, hogy az elem akárhányszor előfordulhat, beleértve ebbe azt is, hogy meg sem jelenik a dokumentumban.

2. Ha + jelet használunk, akkor az elem egy vagy több alkalommal fordulhat elő.

3. A ? azt jelenti, hogy az elem nulla vagy egy példányban fordulhat elő, de egynél több nem lehet belőle.

4. A | jellel elválasztva felsorolhatjuk azokat az értékeket, amelyek közül pontosan egy fordulhat elő egy elemben. Például a <Film> elemnek lehet egy <Műfaj> leszármazottja, és ezt az alábbi módon definiálhatjuk:

   ```
   <!ELEMENT Műfaj (vígjáték|dráma|sci-fi|gyerekfilm)>
   ```

   ezzel megadva, hogy a <Műfaj> a négy felsorolt érték egyike lehet.

5. Az elemek csoportosításához használhatók zárójelek. Például ha a címet az alábbiak szerint adjuk meg:

   ```
   <!ELEMENT Cím Utca, (Város|Irányítószám)>
   ```

   akkor a <Cím> elemek mindegyikének lesz egy <Utca> leszármazott eleme, amit egy <Város> vagy egy <Irányítószám> elem követ, de sohasem mindkettő.

11.3.2. A DTD-k használata

Ha egy dokumentumot egy DTD-vel szeretnénk leírni, akkor két lehetőségünk van ennek megadására:

```
<Színészek>
    <Színész>
        <Név>Carrie Fisher</Név>
        <Cím>
            <Utca>123 Maple St.</Utca>
            <Város>Hollywood</Város>
        </Cím>
        <Cím>
            <Utca>5 Locust Ln.</Utca>
            <Város>Malibu</Város>
        </Cím>
        <Filmek>
            <Film>
                <FilmCím>Csillagok háborúja</FilmCím>
                <Év>1977</Év>
            </Film>
            <Film>
                <FilmCím>A birodalom visszavág</FilmCím>
                <Év>1980</Év>
            </Film>
            <Film>
                <FilmCím>Jedi visszatér</FilmCím>
                <Év>1983</Év>
            </Film>
        </Filmek>
    </Színész>
    <Színész>
        <Név>Mark Hamill</Név>
        <Cím>
            <Utca>456 Oak Rd.<Utca>
            <Város>Brentwood</Város>
        </Cím>
        <Filmek>
            <Film>
                <FilmCím>Csillagok háborúja</FilmCím>
                <Év>1977</Év>
            </Film>
            <Film>
                <FilmCím>A birodalom visszavág</FilmCím>
                <Év>1980</Év>
            </Film>
            <Film>
                <FilmCím>Jedi visszatér</FilmCím>
                <Év>1983</Év>
            </Film>
        </Filmek>
    </Színész>
</Színészek>
```

11.7. ábra. A 11.6. ábrán látható DTD-nek megfelelő dokumentum

a) a DTD-t beillesztjük a dokumentum elejébe, mint egy bevezetőt, vagy

b) a nyitó sorban hivatkozunk a DTD-re, amelyet ilyenkor külön állományként tárolunk a fájlrendszerben úgy, hogy elérhető legyen a dokumentumot feldolgozó program számára.

11.9. példa. A 11.7. példában szereplő dokumentumról megadhatjuk, hogy megfelel a 11.6. ábra szerinti DTD-nek:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE Színészek SYSTEM "színészek.dtd">
```

A `standalone = "no"` attribútum azt adja meg, hogy használunk valamilyen DTD-t. Emlékezzünk rá, hogy ennek az attribútumnak `"yes"` értéket adtunk, amikor nem akartunk DTD-t készíteni a dokumentumhoz. Az a hely, ahol a DTD elérhető, a `!DOCTYPE` állításban szerepel, a `SYSTEM` kulcsszót követően. □

11.3.3. Az attribútumlista

A DTD arra is lehetőséget ad, hogy specifikáljuk az elemek lehetséges attribútumait és azok típusát. A deklarációkat az alábbi formában adjuk meg:

`<!ATTLIST` *elemnév attribútumnév típus* `>`

Ennek a sornak a jelentése pedig az, hogy a megnevezett attribútum a megadott elemnek lehet egy attribútuma, és a típusa a megjelölt típus. Egy `ATTLIST` állításban különböző attribútumok definiálhatók, de ez nem szükségszerű. Az `ATTLIST` állítások a DTD bármely részén elhelyezkedhetnek.

Az attribútumok leggyakoribb típusa a `CDATA` (karakteres adat) típus. Ez a típus alapvetően egy karaktereket, szöveget tartalmazó típus, amelyben a speciális karaktereket, mint például a < jelet, helyettesítjük oly módon, ahogyan a `#PCDATA` esetében tettük. Egy másik lehetőség a felsorolástípus, amely a lehetséges szövegek felsorolása, körbevéve zárójelekkel és egymástól | jelekkel elválasztva. Az adattípust követően lehet még egy kulcsszó, a `#REQUIRED` vagy `#IMPLIED`, ami azt jelenti, hogy egy attribútum felvétele kötelező vagy opcionális.

```
<!ELEMENT Film EMPTY>
    <!ATTLIST Film
        filmcím CDATA #REQUIRED
        év CDATA #REQUIRED
        műfaj (vígjáték|dráma|sci-fi|gyerekfilm) #IMPLIED
    >
```

11.8. ábra. A filmek adatai attribútumként jelennek meg

11.10. példa. Ahelyett, hogy a <Film> elemhez felvennénk a filmcím és az év elemeket, megadhatjuk azokat attribútumként is. A 11.8. ábrán egy lehetséges attribútumlista leírása látható. Fontos, hogy a Film most egy üres elem. Megadjuk a filmcím, év és műfaj attribútumokat. Az első kettő CDATA típusú, míg a műfaj-nak meghatározott értékei lehetnek egy felsorolástípusból. A dokumentumban ennek értékei, például a *vígjáték*, minden esetben idézőjelek között szerepel, azaz

```
<Film filmcím="Csillagok háborúja" év="1977"
     műfaj="sci-fi"/>
```

egy lehetséges elem a DTD-nek megfelelő dokumentumban. □

```
<!DOCTYPE FilmSzínészAdat [
    <!ELEMENT FilmSzínészAdat(Színész*, Film*)>
    <!ELEMENT Színész (Név, Cím+)>
        <!ATTLIST Színész
            színészId #REQUIRED
            szerepelBenne IDREFS #IMPLIED
        >
    <!ELEMENT Név (#PCDATA)>
    <!ELEMENT Cím (Utca, Város)>
    <!ELEMENT Utca (#PCDATA)>
    <!ELEMENT Város (#PCDATA)>
    <!ELEMENT Film (FilmCím, év)>
        <!ATTLIST Film
            filmId #REQUIRED
            szereplő IDREFS #IMPLIED
        >
    <!ELEMENT FilmCím (#PCDATA)>
    <!ELEMENT Év (#PCDATA)>
]>
```

11.9. ábra. A színészek adatait leíró DTD ID és IDREF használatával

11.3.4. Azonosítók és hivatkozások

Idézzük fel a 11.2.5. alfejezetből, hogy egyes attribútumok elemek azonosítójaként használhatók. Egy DTD-ben ezeket az attribútumokat ID típusúnak írjuk le. Más attribútumok felvesznek olyan értékeket, amelyek ezekre az azonosítókra hivatkoznak. Ezeket IDREF típusúként kell definiálni. Egy IDREF típusú attribútum csak olyan értéket vehet fel, amely egy vagy több elem ID típusú attribútumában szerepel, tehát az IDREF valójában egy mutató az ID-re. Megadható egy elemről az is, hogy IDREFS típusú. Ebben az esetben az attribútum

értéke egy szöveg, amely azonosítók listája whitespace-ekkel elválasztva. Valójában az IDREFS attribútum ezt az elemet összeköti azon elemek halmazával, amelyek az ID attribútumban vannak felsorolva.

11.11. példa. A 11.9. ábrán egy DTD látható, amelyben a színészek és a filmek ugyanolyan szerepet játszanak, és az ID-IDREFS kapcsolatok írják le azt a sok-sok kapcsolatot a színészek és a filmek között, amelyeket a 11.1. ábrán láttunk. A dokumentum felépítése abban különbözik a 11.6. ábrán látott DTD-vel leírttól, hogy a színészek és a filmek ugyanazon a szinten jelennek meg, a gyökérelem közvetlen leszármazottjaként. Azaz a gyökérelemhez tartozó DTD-elem neve FilmSzínészAdat, és színészelemek sorozatából áll, amelyet filmek követnek.

Egy színész elemnek a továbbiakban nincsenek filmeket leíró gyermekelemei, ahogyan a 11.6. ábrán látható DTD-ben voltak. A színész elem csak név és cím gyermekelemeket tartalmaz, és a <Színész> nyitó jelölőben megjelenik egy SzerepelBenne nevű attribútum, amely IDREFS típusú, és az értéke a filmjei azonosítóinak listája.

```
<? xml version = "1.0" encoding = "utf-8"
                              standalone = "yes" ?>

<FilmSzínészAdat>
    <Színész SzínészId = "cf" szerepelBenne = "csh">
        <Név>Carrie Fisher</Név>
        <Cím>
            <Utca>123 Maple St.</Utca>
            <Város>Hollywood</Város>
        </Cím>
        <Cím>
            <Utca>5 Locust Ln.</Utca>
            <Város>Malibu</Város>
        </Cím>
    </Színész>
    <Színész SzínészId = "mh" szerepelBenne = "csh">
        <Név>Mark Hamill</Név>
            <Utca>456 Oak Rd.</Utca>
            <Város>Brentwood</Város>
    </Színész>
    <Film FilmId = "csh" szereplő = "cf mh">
        <FilmCím>Csillagok háborúja</FilmCím>
        <Év>1977</Év>
    </Film>
</FilmSzínészAdat>
```

11.10. ábra. A szereplő információ felvétele az XML-dokumentumba

A <Színész> jelölő rendelkezik egy színészId attribútummal is. Mivel ID típusúnak definiáltuk, a színészId értékére hivatkozhatunk a <Film> elemben, megjelölve a film színészeit. A 11.9. ábrán a Film elem attribútumai között szerepel egy FilmId nevű attribútum, amey ID típusú. Ezek azok az azonosítók, amelyek előfordulhatnak a SzerepelBenne attribútumban szereplő lista elemeiként. Szimmetrikusan a Film elem szereplő attribútuma IDREFS típusú, ami színész ID-k felsorolása. □

11.3.5. Feladatok

11.3.1. feladat. Egészítsük ki a 11.10. ábrán látható dokumentumot az alábbi tényekkel:

a) *A birodalom visszavág* (1990) és a *Jedi visszatér* (1983) című filmekben Carrie Fisher és Mark Hamill is szerepeltek.

b) Harrison Ford szerepelt a *Csillagok háborújában*, az *a)* pontban említett két filmben és a *Tűzfal* (2006) című filmekben.

c) Carrie Fisher szerepelt még a *Hannah és nővérei* (1985) című filmben is.

d) Matt Damon szerepelt a *A Bourne-rejtély* (2002) című filmben.

11.3.2. feladat. Mutassunk egy lehetőséget arra, hogy bankokról és ügyfeleikről szóló tipikus adatok, amelyeket a 4.1.1. feladatban láttunk, hogyan írhatók le DTD-vel.

11.3.3. feladat. Mutassuk be, hogy játékosokról, csapatokról és szurkolókról szóló tipikus adatok, amilyeneket a 4.1.3. feladatban láttunk, hogyan írhatók le DTD-vel.

11.3.4. feladat. Mutassuk be, hogy családi kapcsolatokat ábrázoló tipikus adatok, amilyeneket a 4.1.6. feladatban láttunk, hogyan írhatók le DTD-vel.

! **11.3.5. feladat.** A 11.2.2. feladatban elkészített leírás felhasználásával készítsünk olyan algoritmust, amely bármely relációsémához (a relációk neve és az attribútumok neveinek listája) elkészít egy DTD-t, amely leírja a relációt reprezentáló dokumentumot.

11.4. XML-séma

Az *XML-sémák* használata egy alternatív módszer XML-dokumentumok sémájának leírására. Az eszköz erősebb, mint a DTD, mert a séma készítőjének további lehetőségei is vannak. Például az XML-séma tetszőleges számú megszorítást enged meg az elemek előfordulásain. Megengedi egyszerű elemek típusának definiálását, mint például egész vagy lebegőpontos szám; és biztosítja a lehetőséget kulcsok és idegen kulcsok deklarálására.

11.4.1. Az XML-séma formai követelményei

Egy séma XML-séma leírása maga is egy XML-dokumentum. Az alábbi URL-en elérhető névteret használja:

```
http://www.w3.org/2001/XMLSchema
```

amelyet a World-Wide-Web Consortium gondoz. Így minden XML-séma dokumentum az alábbi formájú:

```
<? xml version = "1.0" encoding = "utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    ...
</xs:schema>
```

Az első sor jelentése az, hogy egy XML-dokumentumról van szó és a speciális `<?` és `?>` zárójeleket használja. A második sor a dokumentum, és így a séma gyökérjelölője. Az `xmlns` (XML-névtér) attribútuma megadja, hogy az `xs` változó fogja jelölni az előbb említett névteret. Ez azt eredményezi, hogy az `<xs:schema>` jelölőt ebben a névtérben úgy kell értelmezni, mint a `schema` jelölőt az XML-séma névterében. Ahogyan a 11.2.6. alfejezetben láthattuk, az XML-séma minden egyes kifejezését az `xs:` kezdőtaggal látjuk el, amelynek eredményeként az XML-séma szabályai szerint lesznek értelmezve. A séma az `<xs:schema>` nyitó jelölő és a `</xs:schema>` záró jelölő között helyezkedik el. A következő részekben megtanuljuk az XML-séma névtér legfontosabb jelölőit és azok jelentését.

11.4.2. Elemek

A sémák nagyon fontos összetevője az *elem*, amely hasonlít egy elem DTD-beli definíciójára. Fontos megjegyezni a következő részben tárgyalt ismeretekről, hogy az XML-sémadefiníciók maguk is XML-dokumentumok, így ezek a sémák is „elemekből" épülnek fel. Habár a séma is elemekből épül fel, mindegyiknek `xs:` az előjelölője, ezek nem a séma által definiált elemek[3]. Egy elem definíciója XML-sémával az alábbi:

```
<xs:element name = elem neve type = elem típusa >
    megszorítások és/vagy szerkezeti felépítés információk
</xs:element>
```

Az elem neve az a jelölő, amely elemekhez a sémát készítjük. A típus lehet egyszerű vagy összetett típus. Az egyszerű adattípusok a leggyakrabban használt elemi típusokat tartalmazzák: `xs:integer`, `xs:string` és `xs:boolean`. Egyszerű adattípust tartalmazó elemnek nem lehet gyermekeleme.

[3] Annak érdekében, hogy különbséget tudjunk tenni a sémadefiníció és a séma által leírt elemek között, ez utóbbit minden esetben nagy kezdőbetűvel írjuk.

11.12. példa. Az alábbiakban a filmcím és az év elemek XML-sémadefiníciója látható:

```
<xs:element name = "FilmCím" type = "xs:string" />
<xs:element name = "Év" type = "xs:integer" />
```

Ezen `<xs:element>` elemek mindegyike üres elem, így lezárható rövidített módon a `/>` karaktersorral. Az elsőként definiált, `FilmCím` nevű elem típusa szöveg. A másodikat Év-nek hívják és egész típusú. A dokumentumokban (például a mozifilmekről szólókban), amelyekben a `<FilmCím>` és az `<Év>` előfordul, az elemek nem leszek üresek, hanem egy szöveg (a `FilmCím` után) vagy egy egész szám (az Év-et) követi őket, illetve egy záró jelölőpár, a `</FilmCím>` vagy a `</Év>`. □

11.4.3. Összetett típusok

Az XML-séma *összetett típusa* sokféle formában megjelenhet, de a leggyakrabban elemek sorozataként. Ezeknek az elemeknek szerepelniük kell a megadott sorozatban, de az előfordulásaik száma meghatározható a `minOccurs` és a `maxOccurs` attribútumokkal, amelyeket az elem definíciójában lehet megtalálni. Az attribútumok jelentése a nevüknek megfelelően: a sorozatban a `minOccurs` értékénél nem jelenhet meg kevesebbszer az elem és a `maxOccurs` értékénél többször sem. Ha egy elem egynél többször fordul elő, akkor az azonos típusú elemeknek egymást követően kell megjelenniük. Alapértelmezetten, ha valamelyik vagy mindkét attribútum hiányzik, az egy előfordulást jelent. Ha a lehetséges előfordulások felső korlátját nem akarjuk megszabni, akkor a `maxOccurs` attribútumnak az `"unbounded"` értéket kell adni.

```
<xs:complexType name = típus neve >
   <xs:sequence>
      elemdefiníciók listája
   </xs:sequence>
</xs:complexType>
```

11.11. ábra. Összetett típus definiálása elemek sorozataként

A sorozat típusú összetett elem definícióját a 11.11. ábrán látható módon kell megadni. Az összetett típus nevét nem kötelező megadni, de szükséges, ha a típust fel szeretnénk használni a sémában definiált más elemek típusaként is. Másik lehetséges megoldás, hogy az összetett típus definícióját egy elem nyitó `<xs:element>` jelölője és a hozzá tartozó záró jelölő közé helyezzük, így megadva, hogy az elem típusa a definiált összetett típus.

11.13. példa. Írjunk egy teljes XML-sémát, amely egy nagyon egyszerű sémát határoz meg filmek leírására. A dokumentum gyökéreleme a `<Filmek>` elem lesz, amelynek nulla, egy vagy több `<Film>` gyermekeleme van. Minden `<Film>`

```
1)  <? xml version = "1.0" encoding = "utf-8" ?>
2)  <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">

3)    <xs:complexType name = "filmTípus">
4)      <xs:sequence>
5)        <xs:element name = "FilmCím" type = "xs:string" />
6)        <xs:element name = "Év" type = "xs:integer" />
7)      </xs:sequence>
8)    </xs:complexType>

9)    <xs:element name = "Filmek">
10)     <xs:complexType>
11)       <xs:sequence>
12)         <xs:element name = "Film" type = "filmTípus"
              minOccurs = "0" maxOccurs = "unbounded"/>
13)       </xs:sequence>
14)     </xs:complexType>
15)   </xs:element>

16) </xs:schema>
```

11.12. ábra. Egy filmekhez készült XML-séma leírása

elemnek lesz két leszármazottja, a filmcím és az év, ebben a sorrendben. Az XML-séma dokumentuma a 11.12. ábrán látható.

Az 1. és 2. sorok tipikus bevezetőjét képezik az XML-séma definícióknak. A 3. és 8. sorok között definiáljuk a `filmTípus` nevű összetett típust. Ez a típus két elem sorozatából áll, a `FilmCím` és az `Év` típusokéból; ezek azok az elemek, amelyeket a 11.12. példában láttunk. A típusdefiníció önmagában nem készít egyetlen elemet sem, de a 12. sorban a `filmTípus` kulcsszó használata megadja, hogy a `<Film>` elem ilyen típusú.

A 9. és 15. sorok között a `Filmek` elemet definiáljuk. Habár választhatnánk itt is azt a módszert, hogy egy összetett típust definiálunk, ahogyan a `Film` elemnél tettük, mi azt választjuk, hogy az elemdefinícióban helyezzük el a típusleírást is. Tehát a 9. sorba nem helyezünk el semmilyen attribútumot, ehelyett a 9. sor nyitó `<xs:element>` jelölője és a 15. sor záró jelölője között helyezkedik el a `Filmek` összetett típus definíciója. Ennek az összetett típusnak nincs neve. A 11. sorban definiáljuk, hogy elemek sorozata. Jelen esetben ez a sorozat csak egyféle típusú, `Film` típusú elemeket tartalmaz, a 12. sornak megfelelően. Ez az elem a definíció szerint `filmTípus`, azaz az összetett típus, amelyet a 3. és 8. sorok között definiáltunk. Továbbá megadjuk azt is, hogy az előfordulásainak száma 0 és végtelen között lehet. Tehát a 11.12. ábrán látható séma ugyanazokat az állításokat tartalmazza, mint a 11.13. ábrán szereplő DTD. □

```
<!DOCTYPE Filmek [
    <!ELEMENT Filmek (Film*)>
    <!ELEMENT Film (FilmCím, Év)>
    <!ELEMENT FilmCím (#PCDATA)>
    <!ELEMENT Év (#PCDATA)>
]>
```

11.13. ábra. Filmekhez tartozó DTD

Ezenkívül számos más módszerünk van összetett típusok létrehozására.

- Az `xs:sequence` helyén használhatjuk az `xs:all` kulcsszót, amely azt jelenti, hogy a nyitó `<xs:all>` és a hozzá tartozó záró jelölő között minden elemnek pontosan egyszer kell előfordulnia, tetszőleges sorrendben.
- Továbbá lecserélhetjük az `xs:sequence`-t `xs:choice`-ra. Ekkor pontosan egy elem jelenhet meg az `<xs:choice>` nyitó jelölő és a lezáró párja között leírtakból.

A `sequence` vagy `choice` között meghatározott elemekhez használhatók a `minOccurs` és a `maxOccurs` attribútumok, melyekkel megadható, hogy hányszor fordulhatnak elő a dokumentumban. A `choice` használata esetén egyetlen elem jelenhet csak meg, de abból több példány is, amennyiben a `maxOccurs` paraméterének értéke 1 feletti. Az `xs:all` szabályai ettől eltérőek, a `maxOccurs` attribútum nem vehet fel 1-től eltérő értéket, míg a `minOccurs` lehet 0 vagy 1. Az előbbi esetben az elem lehet hogy egyáltalán nem szerepel.

11.4.4. Attribútumok

Az összetett típusoknak lehetnek attribútumaik is. Azaz, amikor definiálunk egy T elemet, beszúrhatunk `<xs:attribute>` elemeket. Amikor T-t az E elem típusaként használjuk, az E-nek lehet hogy van (kell hogy legyen) ilyen attribútuma. Az attribútumdefiníciók formája az alábbi:

`<xs:attribute name =` *attribútum neve* `type =` *attribútum típusa*
egyéb információk az attribútumról `/>`

Az „egyéb információk” rész tartalmazhat alapértelmezett értéket, információt arról, hogy az attribútum kötelező vagy opcionális (ezek közül az utóbbi az alapértelmezett).

11.14. példa. Az alábbi két sor egy év nevű, egész típusú attribútumot definiál:

```
<xs:attribute name = "év" type = "xs:integer"
    default = "0" />
```

Nem tudjuk megmondani, hogy az év attribútum melyik elemhez tartozik, ez attól függ, hol helyezzük el a definíciót. Az év alapértelmezett értéke 0, ami azt jelenti, hogy ha a dokumentumban egy elem év attribútumának nincs értéke, akkor azt automatikusan 0-ként fogjuk értelmezni.

A következő sorok egy másik definíciót tartalmaznak az év attribútumra:

```
<xs:attribute name = "év" type = "xs:integer"
    use = "required" />
```

Ebben a definícióban a use attribútum required értéke azt jelenti, hogy minden ilyen típusú elemnek meg kell adni az év attribútumát. □

Az attribútumdefiníciók az összetett típus definícióján belül kerültek elhelyezésre. A következő példában átdolgozzuk a 11.13. példát úgy, hogy a filmTípus típusnak legyen egy attribútuma a filmcím és az év jellemzőkhöz ahelyett, hogy ezek beágyazott elemben lennének megadva.

```
1) <? xml version = "1.0" encoding = "utf-8" ?>
2) <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">

3)   <xs:complexType name = "filmTípus">
4)     <xs:attribute name = "filmcím" type = "xs:string"
           use = "required" />
5)     <xs:attribute name = "év" type = "xs:integer"
           use = "required" />
6)   </xs:complexType>

7)   <xs:element name = "Filmek">
8)     <xs:complexType>
9)       <xs:sequence>
10)        <xs:element name = "Film" type = "filmTípus"
                minOccurs = "0" maxOccurs = "unbounded"/>
11)      </xs:sequence>
12)    </xs:complexType>
13)  </xs:element>

14) </xs:schema>
```

11.14. ábra. Attribútumok használata egyszerű elemek helyett

11.15. példa. A 11.14. ábrán láthatjuk az átdolgozott XML-séma definíciót. A 4. és 5. sorokban a `filmcím` és az `év` attribútumokat a `filmTípus` kötelező elemeiként definiáltuk. Onnantól, hogy a `Film` elemnek típust adunk a 10. sorban, tudjuk, hogy minden `Film` elemnek rendelkeznie kell a `filmcím` és az `év` jellemzőkkel. A 11.15. ábrán a 11.14. ábrához hasonló tartalmú DTD-t láthatunk. □

```
<!DOCTYPE Filmek [
    <!ELEMENT Filmek (Film*)>
    <!ELEMENT Film EMPTY>
        <!ATTLIST Film
            filmcím CDATA #REQUIRED
            év CDATA #REQUIRED
        >
]>
```

11.15. ábra. A 11.14. ábrán látható sémával ekvivalens DTD

11.4.5. Egyszerű típusok megszorításokkal

Az egyszerű típusokból, mint amilyen az egész szám vagy a szöveg, lehet korlátozott típusokat is készíteni úgy, hogy korlátozzuk az általuk felvehető értékeket. Ezek a típusok aztán felhasználhatók elemek vagy attribútumok értékeként. Az alábbiakban a megszorítások két lehetséges megadási módját vizsgáljuk meg.

1. Numerikus értékek esetében a `minInclusive` attribútumban adható meg a legalacsonyabb felvehető érték, míg a `maxInclusive` attribútumban a felső korlát[4].

2. A lehetséges értékeket egy felsorolástípussal korlátozzuk.

A tartománykorlátozás alakja a 11.16. ábrán látható. A megszorításnak van egy bázistípusa, amely egy egyszerű típus lehet (például `xs:szöveg`).

```
<xs:simpleType name = típus neve >
   <xs:restriction base = alaptípus >
      alsó és/vagy felső korlátok
   </xs:restriction>
</xs:simpleType>
```

11.16. ábra. A tartománykorlátozás megadási módja

[4] Az „inclusive" szó azt jelenti, hogy a benne megadott érték is a használható tartományban van. Ha az `Inclusive` részt az `Exclusive` kulcsszóra cseréljük, azzal jelezhetjük, hogy a megadott érték már kívül van a megengedett értelmezési tartományon.

11.16. példa. Tegyük fel, hogy a mozifilmek év mezőjét szeretnénk korlátozni úgy, hogy ne lehessen 1915-nél korábbi. Ahelyett, hogy az `xs:integer` típust használjuk az év elem típusaként a 11.12. ábra 6. sorában vagy a 11.14. ábra 5. sorában, definiálhatunk egy új elemet is, ahogyan a 11.17. ábrán látható. A `filmÉvTípus` az `xs:integer` helyére kerül. □

```
<xs:simpleType name = "filmÉvTípus">
  <xs:restriction base = "xs:integer" >
    <xs:minInclusive value = "1915" />
  </xs:restriction>
</xs:simpleType>
```

11.17. ábra. Ebben a típusban az egész számokat korlátozzuk, csak 1915-nél nagyobb értékeket vehetnek fel

A második módszerünk arra, hogy egy egyszerű típusra megszorításokat alkalmazzunk, a lehetséges értékek megadása. Az egyszerű felsorolástípus megadásának formája a következő:

`<xs:enumeration value =` *valamilyen értékek* `/>`

A megszorítás tetszőleges számú értéket tartalmazhat.

11.17. példa. Tervezzünk egy egyszerű típust, amely alkalmas a filmek műfajának megadására. A szokásos példánkban feltettük, hogy négy műfaj lehetséges: vígjáték, dráma, sci-fi és gyerekfilm. A 11.18. ábra a `műfajTípus` definíciójának megadását mutatja, amely később egy műfajt ábrázoló elem vagy attribútum típusaként szolgálhat. □

```
<xs:simpleType name = "műfajTípus">
  <xs:restriction base = "xs:string" >
    <xs:enumeration value = "vígjáték" />
    <xs:enumeration value = "dráma" />
    <xs:enumeration value = "sci-fi" />
    <xs:enumeration value = "gyerekfilm" />
  </xs:restriction>
</xs:simpleType>
```

11.18. ábra. Felsorolástípus megadása XML-sémával

11.4.6. Kulcsok az XML-sémában

Egy elemhez tartozhat kulcsdeklaráció. Ez azt jelenti, hogy amikor egy bizonyos C típusú elemek osztályát nézzük, megadott *mező* vagy *mezők* értékei ezeken az elemeken belül egyediek. A „mező" fogalom ebben az esetben meglehetősen

általános, de leggyakrabban mező alatt érthetünk beágyazott elemet és attribútumot is. A C osztály elemeit „szelektorokkal" definiáljuk. Ahogyan a mezők, úgy a szelektor is lehet összetett, de leggyakrabban egy vagy több elemnév sorozata, és mindegyik az őt megelőző beágyazott eleme. A félig-strukturált adatfák nyelvén az osztály azon elemeket jelenti, amelyek egy magadott csúcsból elérhetők az élcímkék neveit tartalmazó listát követve.

11.18. példa. Tegyük fel, hogy a 11.1. ábrán látható félig-strukturált adatban minden olyan csúcsról, amely a gyökérelemből *színész* címkét, majd a *név* címkét követve elérhető, azt állítanánk, hogy egyedi. A „szelektor" a *színész* lesz, míg a „mező" a *Név*. Ennek az állításnak az a következménye, hogy a bemutatott gyökérelemen belül nem helyezkedhet el két azonos nevű színész. Ha a mozifilmeknek a filmcím helyett név jellemzőjük lenne, a kulcsmegadás akkor sem akadályozhatná meg, hogy azonos nevű színész és film kerüljön be az adatbázisba. Továbbá, ha a 11.1. ábrán látott fához hasonló elemből több is lenne a dokumentumban (például minden objektum, amelyet „gyökérelemnek" neveztünk, egy mozifilm lenne a szereplőivel együtt), akkor a különböző fákban megjelenhetnének azonos nevű színészek megszorítás megsértése nélkül. □

A kulcs deklarációjának alakja az alábbi:

```
<xs:key name = kulcs neve >
   <xs:selector xpath = útvonalleírás >
   <xs:field xpath = útvonalleírás >
</xs:key>
```

Az `xs:field` elemet tartalmazó sorból több is lehet, ekkor a különböző mezők együtt alkotják a kulcsot. Másik lehetőség lehet az `xs:unique` használata az `xs:key` helyett. A különbség annyi, hogy ha a „key" megadást használjuk, akkor a mezőnek meg kell jelennie a szelektorokkal kiválasztott elemek mindegyikében. Ezzel szemben, ha a „unique"-ot használjuk, nem kötelező a megadásuk, és a megszorítás csupán annyi, hogy ha léteznek, akkor legyenek egyediek.

A szelektor elérési útja lehet elemek egy sorozata, amelyek mindegyike az őt megelőző leszármazottja. Az elemneveket / jellel választjuk el. A mező lehet az utolsó elem gyermekeleme vagy egy attribútuma. Attribútum esetén a nevét megelőzi egy `@` jel. További lehetőségek is vannak, mert a szelektor és a mező tetszőleges XPath-kifejezés lehet. Az XPath lekérdező nyelvvel részletesen a 12.1. alfejezetben foglalkozunk.

11.19. példa. A 11.19. ábrán a 11.12. ábra egy bővítése látható. A `filmTípus` elemhez hozzáadtuk a `Műfaj` jellemzőt annak érdekében, hogy a `Film` elemnek legyen nem kulcs gyermekeleme. A 3. és 10. sorok között a `műfajTípus`-t definiáljuk ugyanúgy, mint a 11.17. példában. A `Műfaj` gyermekelemet a 15. sorban adjuk hozzá a `filmTípus` elemhez.

A 24. és 28. sorok között a kulcs megadásával megváltozott a `Filmek` elemek definíciója. A kulcs neve `FilmKulcs`. Ezt a nevet fogjuk használni akkor, amikor idegen kulcsos hivatkozásként adjuk meg, amivel a 11.4.7. fejezetben

```
1) <? xml version = "1.0" encoding = "utf-8" ?>
2) <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">

3)   <xs:simpleType name ="műfajTípus">
4)      <xs:restriction base = "xs:string">
5)         <xs:enumeration value = "vígjáték" />
6)         <xs:enumeration value = "dráma" />
7)         <xs:enumeration value = "sci-fi" />
8)         <xs:enumeration value = "gyerekfilm" />
9)      </xs:restriction>
10)  </xs:simpleType>

11)  <xs:complexType name = "filmTípus">
12)     <xs:sequence>
13)        <xs:element name = "FilmCím" type = "xs:string" />
14)        <xs:element name = "Év" type = "xs:integer" />
15)        <xs:element name = "Műfaj" type = "műfajTípus"
              minOccurs = "0" maxOccurs = "1" />
16)     </xs:sequence>
17)  </xs:complexType>

18)  <xs:element name = "Filmek">
19)     <xs:complexType>
20)        <xs:sequence>
21)           <xs:element name = "Film" type = "filmTípus"
                 minOccurs = "0" maxOccurs = "unbounded" />
22)        </xs:sequence>
23)     </xs:complexType>
24)     <xs:key name = "FilmKulcs">
25)        <xs:selector xpath = "Film"/>
26)        <xs:field xpath = "FilmCím" />
27)        <xs:field xpath = "Év" />
28)     </xs:key>
29)  </xs:element>

30) </xs:schema>
```

11.19. ábra. XML-sémával megadott séma mozifilmekhez

foglalkozunk. Ettől eltekintve a névnek nincs jelentősége. A szelektor útvonala a `Film`, amelynek két mezője van: `FilmCím` és `Év`. Ez a kulcsdeklaráció azt jelenti, hogy a `Filmek` elem összes `Film` gyermekelemében semelyik két filmnek nem lehet ugyanaz a címe és a keletkezésének éve, illetve egyik érték sem hiányozhat. Megjegyezzük, hogy a `filmTípus` 13. és 14. sorban megadott definíciójában nincs `minOccurs` és `maxOccurs` érték megadva a `FilmCím` és az `Év` mezőkhöz, tehát az alapértelmezett 1 lesz az érvényes, így mindkét mezőből pontosan egy fordulhat elő. □

11.4.7. Idegen kulcsok az XML-sémákban

Azt is megadhatjuk, hogy egy elemnek van olyan, esetleg mélyen beágyazott mezője, amely más elemek elsődleges kulcsára hivatkozik. Ez a tulajdonság hasonlít ahhoz, amit az ID-k és az IDREF-ek között láttunk a DTD-k esetében (lásd 11.3.4. alfejezet). Ez utóbbiak nem típusos hivatkozások, míg az XML-sémában főként típusos hivatkozások vannak. XML-sémában idegen kulcsot az alábbi formában definiálhatunk:

```
<xs:keyref name = idegenkulcs neve refer = kulcs neve >
   <xs:selector xpath = xpath-leírás >
   <xs:field xpath = xpath-leírás >
</xs:keyref>
```

Az új sémaelem az `xs:keyref`. Az idegen kulcsnak van neve és valamely kulcsra vagy egyedi értékű mezőre hivatkozik. A szelektor és a mezők megadása ugyanazt a szerepet tölti be, mint a kulcsok esetén.

11.20. példa. A 11.20. ábra egy `<Színészek>` elem definícióját tartalmazza. A séma megadásánál azt a formát választottuk, hogy az összetett típust abban az elemben definiáljuk, amelyik használja. Így a 4. és 6. sorok között azt adjuk meg, hogy a `<Színészek>` elem egy vagy több `<Színész>` gyermekelemet tartalmaz.

A 7. és 11. sorok között láthatjuk, hogy minden egyes `<Színész>` elemnek háromféle leszármazottja lehet. Pontosabban van neki egy `<Név>` és egy `<Cím>` gyermekeleme és tetszőleges számú `<SzerepelBenne>` leszármazottja. A 12. sor és a 15. sor között azt látjuk, hogy a `<SzerepelBenne>` elemnek nincs leszármazottja, de van két attribútuma, a `filmcím` és az `év`.

A 22. és 26. sorok között egy idegen kulcsot hozunk létre. A 22. sorban látható, hogy az idegen kulcs neve `filmRef` és arra a `FilmKulcs`-ra hivatkozik, amelyet a 11.19. ábrán definiáltunk. Figyeljük meg, hogy a 11.19. ábrán ezt a kulcsot a `<Színészek>` elemen belül hoztuk létre. A `Színész/SzerepelBenne` szelektort használjuk, ami azt jelenti, hogy a `<Színészek>` elem minden egyes `<Színész>` elemében kell lennie egy `<SzerepelBenne>` gyermekelemnek. Ezekből a `<SzerepelBenne>` elemekből kiolvashatók a `filmcím` és az `év` mezők. A `@` jelöli, hogy nem beágyazott elemekről, hanem attribútumokról van szó. Az idegen kulcs azt állítja, hogy minden olyan filmcím-év párhoz, amelyet így megtalálunk, lesz olyan `<Film>` elem, amely tartalmazza ezek párját a `<FilmCím>` és az `<Év>` elemekben. □

```
1) <? xml version = "1.0" encoding = "utf-8" ?>
2) <xs:schema xmlns:xs = "http://www.w.org/2001/XMLSchema">

3) <xs:element name = "Színészek">

4)   <xs:complexType>
5)     <xs:sequence>
6)       <xs:element name = "Színész" minOccurs = "1"
               maxOccurs = "unbounded">
7)         <xs:complexType>
8)           <xs:sequence>
9)             <xs:element name = "Név"
                 type = "xs:string" />
10)            <xs:element name = "Cím"
                 type = "xs:string" />
11)            <xs:element name = "SzerepelBenne"
                 minOccurs = "0"
                 maxOccurs = "unbounded">
12)              <xs:complexType>
13)                <xs:attribute name ="filmcím"
                     type = "xs:string"/>
14)                <xs:attribute name ="év"
                     type = "xs:integer"/>
15)              </xs:complexType>
16)            </xs:element>
17)          </xs:sequence>
18)        </xs:complexType>
19)      </xs:element>
20)    </xs:sequence>
21)  </xs:complexType>

22)  <xs:keyref name = "filmRef" refers = "FilmKulcs">
23)    <xs:selector xpath = "Színész/SzerepelBenne" />
24)    <xs:field xpath = "@filmcím" />
25)    <xs:field xpath = "@év" />
26)  </xs:keyref>

27) </xs:element>
```

11.20. ábra. Színészek adatai idegen kulcsokkal

11.4.8. Feladatok

11.4.1. feladat. Adjunk példát olyan dokumentumra, amely megfelel a 11.12. ábrán látható XML-sémának, majd olyanra is, amely tartalmaz minden említett elemet, mégsem felel meg a definíciónak.

11.4.2. feladat. Írjuk át a 11.12. ábrát úgy, hogy a `Filmek` típusa nevesítve legyen, a `Film` típusnak viszont ne legyen neve.

11.4.3. feladat. Írjuk át a 11.19. és 11.20. ábrán látható XML-séma definíciókat DTD-vé.

11.5. Összefoglalás

- *Félig-strukturált adat:* Ebben a modellben az adatot egy gráf reprezentálja. A csomópontok objektumszerűek vagy attribútumok értékei, és címkézett élek kapcsolják össze őket mind az attribútumértékekkel, mind a más, velük relációban álló objektumokkal.

- *XML:* A kiterjesztett leíró nyelv (Extensible Markup Language) a World-Wide-Web Consortium által létrehozott szabvány, amely képes a félig-strukturált adatot lineárisan ábrázolni.

- *XML-elem:* Az elemek egy nyitó jelölővel kezdődnek (például `<valami>`) és egy záró jelölővel fejeződnek be (például `</valami>`) és minden ami ezek között van. Előfordulhat benne bármilyen szöveg, illetve beágyazott elemek, tetszőleges mélységben.

- *XML-attribútum:* A jelölőkben megjelenhetnek tulajdonságérték-párok. Ezek az attribútumok további információkat szolgáltatnak azokról az elemekről, amelyekben elhelyezzük őket.

- *Dokumentumtípus-definíció (Document Type Definition):* A DTD egy egyszerű nyelvtan XML-elemek és attribútumok definiálására, így egy körülbelüli sémát ad azon XML-dokumentumoknak, amelyek használják az adott DTD-t. Egy elemről megadhatjuk, hogy gyermekelemek sorozatát tartalmazza, továbbá előírhatjuk, hogy ezek az elemek pontosan egyszer, többször, maximum egyszer vagy tetszőlegesen sokszor fordulhatnak elő a szülőben. Egy elemhez továbbá definiálhatunk kötelező vagy opcionális attribútumokat is.

- *Azonosítók és hivatkozások DTD-ben:* Annak érdekében, hogy a nem faszerkezetű gráfokat is tudjuk ábrázolni, a DTD lehetőséget biztosít `ID` és `IDREF(S)` típusú attribútumok definiálására is. Egy elemnek adhatunk egyedi azonosítót, amelyre hivatkozhatunk egy másik elemből, amellyel össze szeretnénk kapcsolni.

- *XML-séma (XML Schema):* Ez egy újabb módszer arra, hogy egyes XML-dokumentumokhoz sémát definiáljunk. Az XML-sémadokumentumok maguk is XML-ben íródnak, de egy olyan névtér jelölőit használják, amely a World-Wide-Web Consortium szabványa.
- *Egyszerű típusok az XML-sémában:* Az XML-sémában definiálva vannak egyszerű típusok, mint például az egész vagy a szövegtípus. További egyszerű típusokat hozhatunk létre úgy, hogy e típusok értékkészletét korlátozzuk tartomány megadásával vagy a lehetséges értékek felsorolásával.
- *Összetett típusok az XML-sémában:* Összetett típusokat definiálhatunk elemek sorozataként, melynek minden tagjáról megadhatjuk az előfordulásainak minimális, illetve maximális számát.
- *Kulcsok és idegen kulcsok az XML-sémában:* Elemek vagy attribútumok egy halmazáról megadhatjuk, hogy bizonyos elemeken belül egyediek legyenek. Más elem- vagy attribútumhalmazról pedig megadhatjuk, hogy értékei csak olyanok lehetnek, amelyek kulcsként előfordulnak bizonyos más típusú elemekben.

11.6. Irodalomjegyzék

A félig-strukturált adat mint adatmodell első tanulmányai a [5] és az [4]. [3] a LOREL-ről, az ehhez a modellhez első prototípusként megvalósított lekérdező nyelvről szól. Tanulmányok találhatók a félig-strukturált adatmodellről az [1], [7] és [2] könyvekben.

Az XML-szabványt a World-Wide-Web Consortium dolgozta ki. [9] az XML-szabvány webes elérése. Szintén itt található a DTD-k és XML-séma leírása. XML-elemzők számára a DOM a [8] linken érhető el, a SAX a [6] alatt. A [10] linken a témához kapcsolódó tutorialok érhetők el.

[1] S. Abiteboul, „Querying semi-structured data," *Proc. Intl. Conf. on Database Theory* (1997), Jegyzetek: Computer Science 1187 (F. Afrati, P. Kolaitis, eds.), Springer-Verlag, Berlin, pp. 1–18.

[2] S. Abiteboul, D. Suciu, P. Buneman, *Data on the Web: From Relations to Semistructured Data and XML*, Morgan-Kaufmann, San Francisco, 1999.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. L. Weiner, „The LOREL query language for semistructured data," In *J. Digital Libraries* **1**:1, 1997.

[4] P. Buneman, S. B. Davidson, D. Suciu, „Programming constructs for unstructured data," *Proceedings of the Fifth International Workshop on Database Programming Languages*, Gubbio, Italy, Sept., 1995.

[5] Y. Papakonstantinou, H. Garcia-Molina, J. Widom, „Object exchange across heterogeneous information sources," *IEEE Intl. Conf. on Data Engineering*, pp. 251–260, March 1995.

[6] Sax Project, `http://www.saxproject.org/`

[7] D. Suciu (ed.) Special issue on management of semistructured data, *SIGMOD Record* **26**:4 (1997).

[8] World-Wide-Web Consortium, `http://www.w3.org/DOM/`

[9] World-Wide-Web Consortium, `http://www.w3.org/XML/`

[10] W3 Schools, `http://www.w3schools.com`

12. fejezet

Programozási nyelvek az XML-hez

Figyelmünket most a félig-strukturált adatok kezeléséhez használatos programozási nyelvekre fordítjuk. Az ilyen típusú, széles körben használt nyelvek mindegyike alkalmazható XML-adatokra, és lehetőséget biztosít a félig-strukturált adatok más módon történő ábrázolására is. Ebben a fejezetben meg fogunk ismerni ezen nyelvek közül hármat. Először az XPath-t, amely egy egyszerű nyelv a félig-strukturált adatok gráfszerkezetében hasonló útvonalak (ösvények) halmazának leírására. Az XQuery az XPath egy kiterjesztése, amely tartalmaz valamennyit az SQL-szerű leírásból. Ez a nyelv megenged halmazok feletti iterációkat, alkérdéseket, és sok egyéb olyan jellemzője van, amely már ismerős lehet SQL-tanulmányainkból.

A fejezet harmadik témája az XSLT. Ez a nyelv eredetileg egy transzformációs nyelvként lett kifejlesztve az XML-dokumentumok átstrukturálására, illetve az XML-dokumentumok nyomtatható (HTML-) dokumentumokká alakítására. Mindamellett kifejező ereje nagyon hasonlóvá teszi az XQueryhez, és képessé teszi XML-dokumentumok előállítására. Emiatt az XML egy lekérdező nyelveként is tud szolgálni.

12.1. XPath

Ebben a részben az XPath-t mutatjuk be. Először tárgyaljuk az XPath legújabb verziójának, az XPath 2.0-nak az adatmodellezését. Ez a modell nagyon jól használható az XQueryben. A modell analóg szerepet játszik az „egyszerű komponensű sorok multihalmazaihoz", amelyet a relációs modell alkalmaz a relációk értékeként.

A későbbi részekben megtanuljuk az XPath útkifejezéseit és azok jelentését. Általánosan ezek a kifejezések lehetővé teszik a mozgást a dokumentum egyik eleméből egy másikba vagy bármelyik gyerekelemébe. A „tengelyek" használata

lehetőséget biztosít arra, hogy a dokumentumon belül változatos útvonalakon mozoghassunk, és elérjük az elemek attribútumait.

12.1.1. Az Xpath-adatmodell

A relációs modellhez hasonlóan az XPath feltételezi, hogy az összes érték – amelyeket a köztes lépésekben előállít vagy megszerkesztésre kerülnek – azonos általános „alakkal" rendelkezik. A relációs modellben ez az „alak" a sorok multihalmaza. Egy adott multihalmazban lévő soroknak azonos számú komponensük van, és ezek a komponensek valamilyen egyszerű típussal rendelkeznek, például egészek vagy szövegek. Az XPath-ban az „alak" megfelelője az *adattételek* – továbbiakban *tételek* – *szekvenciája*. Egy *tétel* a következők egyike:

1. Valamilyen egyszerű típusú érték: például egész, valós, logikai vagy szöveg.
2. Egy *csomópont*. Számos csomópont létezik, de tárgyalásunkban csak a következő három fajtával fogunk foglalkozni:
   a) *Dokumentumok*. Ezek XML-dokumentumot tartalmazó fájlok, esetleg lokális elérési útvonalukkal vagy egy URL-lel jelölve.
   b) *Elemek*. Ezek XML-elemek, beleértve a nyitó jelölőjüket, a megfelelő záró jelölőjüket és mindent, ami e kettő között szerepel. (Például egy félig-strukturált adat fáját, melyet egy XML-dokumentum reprezentál.)
   c) *Attribútumok*. Ezek a nyitó jelölőkben találhatók, amelyekkel a 11. fejezetben foglalkoztunk.

Egy szekvencia elemei nem kötelezően azonos típusúak, bár a legtöbbször azok.

```
10

"tíz"

10.0

<Számrendszer bázis = "8">
    <Digit>1</Digit>
    <Digit>2</Digit>
</Számrendszer>

@val="10"
```

12.1. ábra. Öt tétel szekvenciája

12.1. példa. A 12.1. ábra öt tétel szekvenciáját mutatja. Az első a 10 egész szám, a második egy szöveg, a harmadik pedig egy valós szám. Ezek a tételek mind egyszerű típusúak.

A negyedik tétel egy csomópont, amelynek típusa „elem". Figyeljük meg, hogy ez az elem egy `Számrendszer` nevű jelölővel rendelkezik, amelynek egy attribútuma és két gyerekeleme van `Digit` jelölőkkel. Az utolsó elem egy attribútum-csomópont. □

12.1.2. Dokumentum-csomópontok

Azok a dokumentumok, amelyeket az XPath feldolgoz, különböző forrásokból származhatnak, de az a közös bennük, hogy az XPath által feldolgozható dokumentumok fájlok. A következő függvény alkalmazásával tudunk egy dokumentum-csomópontot készíteni egy fájlból:

`doc(`*fájlnév*`)`

A megjelölt fájlnak XML-dokumentumnak kell lennie. A fájlt megadhatjuk lokális névvel, illetve egy URL-lel is, amennyiben egy távoli fájlról van szó. Így az alábbiak példák dokumentum-csomópontokra:

```
doc("filmek.xml")
doc("/usr/sally/data/filmek.xml")
doc("infolab.stanford.edu/~hector/filmek.xml")
```

Minden Xpath-lekérdezés egy dokumentumra hivatkozik. Sok esetben ez a dokumentum világosan kiderül a szövegkörnyezetből. Példaként hivatkozunk az XML-séma kulcsainak tárgyalására a 11.4.6. alfejezetben. Xpath-kifejezéseket használtunk a szelektor és a mezők megadására a kulcs számára. Abban a környezetben a dokumentum a következő volt: „a mindenkori dokumentum, amelyre a sémadefiníciót alkalmazzuk".

12.1.3. Útkifejezések

Egy Xpath-kifejezés tipikusan egy dokumentum gyökerével kezdődik, és egy szekvenciát tartalmaz, amely jelölőkből és törtvonalakból (/) áll, például $/T_1/T_2/\cdots/T_n$. Kiértékeljük ezt a kifejezést onnan kezdve, hogy a tételek szekvenciája egyetlen csomópontból áll: a dokumentumból. Ezt követően feldolgozzuk minden egyes $T_1, T_2, \ldots$-t. Egy T_i feldolgozása figyelembe veszi a szekvenciában az előző tételek feldolgozása során kapott eredményeket, amennyiben vannak ilyenek. Sorban meg kell vizsgálnunk ezeket a tételeket, és meg kell keresni az összes olyan gyerekelemet, melynek a jelölője T_i. Ezek a tételek a kimeneti szekvencia részei abban a sorrendben, ahogy azok a dokumentumban megjelennek.

Speciálisan a T_1 gyökérjelölőt a dokumentum-csomópont egy „gyerekelemének" tekintjük. Így a $/T_1$ kifejezés egyetlen tételből álló szekvenciát fog előállítani, amely a dokumentum teljes tartalmát magában foglaló elemcsomópont lesz. A következő finomság tűnhet fel: a $/T_1$ kifejezés alkalmazása előtt a dokumentum-csomópontunk reprezentált egy fájlt, míg a $/T_1$ kifejezés erre a csomópontra történő alkalmazása után egy olyan csomópontot kapunk, amely a fájlban lévő szöveget reprezentálja.

12.2. példa. Tételezzük fel, hogy a dokumentumunk a 11.5. ábra XML-t tartalmazó fájlja, amelyet reprodukáltunk a 12.2. ábrán. A `/FilmSzínészAdat` útkifejezés egyetlen elem szekvenciáját állítja elő. Ez az elem természetesen rendelkezik a `<FilmSzínészAdat>` jelölővel és az 1. soron kívül mindent tartalmaz a 12.2. ábráról.

```
 1)  <? xml version="1.0" encoding="utf-8" standalone="yes" ?>
 2)  <FilmSzínészAdat>
 3)      <Színész színészID = "cf" szerepelBenne = "csh">
 4)          <Név>Carrie Fisher</Név>
 5)          <Cím>
 6)              <Utca>123 Maple St.</Utca>
 7)              <Város>Hollywood</Város>
 8)          </Cím>
 9)          <Cím>
10)              <Utca>5 Locust Ln.</Utca>
11)              <Város>Malibu</Város>
12)          </Cím>
13)      </Színész>
14)      <Színész színészID = "mh" starredIn = "csh">
15)          <Név>Mark Hamill</Név>
16)          <Utca>456 Oak Rd.</Utca>
17)          <Város>Brentwood</Város>
18)      </Színész>
19)      <Film filmID = "csh" Szereplő= "cf", "mh">
20)          <FilmCím>Csillagok háborúja</FilmCím>
21)          <Év>1977</Év>
22)      </Film>
23)  </FilmSzínészAdat>
```

12.2. ábra. Egy XML-dokumentum az útkifejezések feldolgozására

Most tekintsük a következő útkifejezést:

```
/FilmSzínészAdat/Színész/Név
```

Amikor alkalmazzuk a `FilmSzínészAdat` jelölőt a dokumentum szekvenciájára, akkor a gyökérelemet tartalmazó szekvenciát kapjuk vissza, ahogy fen-

tebb megbeszéltük. A következő lépésben alkalmazzuk erre a szekvenciára a `Színész` jelölőt. A `FilmSzínészAdat` elemnek két olyan gyerekeleme van, amely rendelkezik a `Színész` jelölővel. Ezek a 3. és a 12. sorok között találhatók Carrie Fisherről, illetve a 14. és a 18. sorok között Mark Hamillről. Így a `/FilmSzínészAdat/Színész` útkifejezés ennek a két elemnek a szekvenciája lesz, ebben a sorrendben.

Végezetül alkalmazzuk erre a szekvenciára a `Név` jelölőt. Az első elem egy `Név` gyerekelemmel rendelkezik a 4. sorban. A második elem szintén egy `Név` gyerekelemmel rendelkezik a 15. sorban. Így a

```
<Név>Carrie Fisher</Név>
<Név>Mark Hamill</Név>
```

szekvencia lesz az eredménye a `/FilmSzínészAdat/Színész/Név` útkifejezés alkalmazásának a 12.2. ábra dokumentumára. □

12.1.4. Relatív útkifejezések

Számos összefüggésben úgy fogjuk használni az XPath-kifejezéseket, hogy azok *relatív*ak az aktuális csomóponthoz vagy csomópontok szekvenciájához.

- A 11.4.6. alfejezetben beszéltünk a szelektorokról és a mezőértékekről, amelyek valójában relatív XPath-kifejezések voltak azokhoz a csomópontokhoz vagy a csomópontok szekvenciájához, amelyekhez egy kulcsot definiáltunk.
- A 12.2. példában beszéltünk a `Színész` XPath-kifejezés alkalmazásáról az egész dokumentumot tartalmazó elemre, illetve a `Név` kifejezés alkalmazásáról a `Színész` elemek szekvenciájára.

A relatív kifejezések nem kezdődhetnek / jellel. Minden ilyen kifejezést valamilyen környezetben kell alkalmazni, amely a használatából válik világossá. A hasonlóság a UNIX fájlrendszerében használatos fájlok és könyvtárak útvonalmegadásához nem véletlen.

12.1.5. Attribútumok az útkifejezésekben

Az útkifejezések lehetővé teszik számunkra, hogy egy dokumentum összes elemét megtaláljuk, amelyek elérhetők a gyökértől indulva pontos útvonalmegadással (a jelölők szekvenciájával). Időnként mi nem ezeket az elemeket akarjuk megtalálni, hanem ezen elemek egy attribútumának az értékét. Ilyenkor az útvonal kifejezést egy attribútum nevével kell zárni, amelyet megelőz egy `@` jel. Ebben az esetben az útvonal-kifejezés $/T_1/T_2/\cdots/T_n/@A$ alakú.

Ennek a kifejezésnek az eredményéhez először a $/T_1/T_2/\cdots/T_n$ útkifejezés illeszkedését határozzuk meg, amely az elemek egy szekvenciáját adja. Ezt követően minden elem nyitó jelölőjét megvizsgáljuk azért, hogy megtaláljuk az A attribútumot. Amennyiben ez megvan, akkor ennek az attribútumnak az értékét kell hozzáfűznünk a szekvencia által megadott eredményhez.

12.3. példa. A

```
/FilmSzínészAdat/Színész/@színészID
```

útkifejezését alkalmazva a 12.2. ábra dokumentumára két `Színész` elemet találunk, és ezek nyitó jelölőit vizsgáljuk a 3. és 14. sorokban, hogy megtaláljuk a `színészID` attribútumuk értékét. Mindkét elem rendelkezik ezzel az attribútummal, így az eredményül kapott szekvencia `"cf" "mh"` lesz. □

12.1.6. Tengelyek

Eddig csak két navigációs lehetőségünk volt egy félig-strukturált adatgráfban: egy csomóponttól a gyerekeihez vagy egy attribútumhoz eljutni. Az XPath valójában a *tengelyek* nagy számát nyújtja, amelyek a navigáció módjai. Ezek közül kettő a *gyerek* (az alapértelmezett tengely) és az *attribútum*, amelyhez a `@` a rövid leírás. Egy útkifejezés minden lépésében beszúrhatunk egy jelölő vagy egy attribútum neve elé egy tengelynevet és egy dupla kettőspontot. Például a

```
/FilmSzínészAdat/Színész/@színészID
```

egy rövid leírása a következőnek:

```
/child::FilmSzínészAdat/child::Színész/attribute::színészID
```

Néhány az egyéb tengelyek közül: a szülő, elől (igazi elől), az utód (valójában igazi utód), a következő testvér (valamelyik jobboldali testvér), megelőző testvér (valamelyik bal oldali testvér), az önmaga és a leszármazott-vagy-önmaga. Ez utóbbinak a `//` egy rövid leírása, és ezeknek az elemeknek és az összes ilyen gyerekelemeiknek a szekvenciáját adja vissza az adott kiindulási helytől tetszőleges mélységben.

12.4. példa. Nehéznek tűnik megtalálni a 12.2. ábra dokumentumában az összes várost, ahol a színészek laknak. A probléma az, hogy Mark Hamill városa nem egy `Cím` elembe van ágyazva, így nem tudjuk ugyanazon az úton elérni, mint Carrie Fisher városát. Azonban a

```
//Város
```

útkifejezés megtalálja az összes `Város` gyerekelemet a beágyazottság bármely szintjén, és visszaadja abban a sorrendben, ahogy az a dokumentumban megjelenik. Ennek az útkifejezésnek az eredménye a következő szekvencia:

```
<Város>Hollywood</Város>
<Város>Malibu</Város>
<Város>Brentwood</Város>
```

amelyeket egyenként a 7., 11. és 17. sorokban találhatunk.

Használhatjuk a // tengelyt egy útkifejezés belsejében is. Például, ha van egy dokumentum, amely nem csak a színészekkel kapcsolatban tartalmaz városinformációkat (például stúdiók és ezek címei), akkor korlátozni tudjuk az utakat annak figyelembevételével, hogy biztosítjuk, hogy a városok egy `Színész` elem gyerekelemei legyenek. Az adott dokumentumra a

```
/FilmSzínészAdat/Színész//Város
```

útkifejezés ugyanezt a három `Város` elemet adja eredményül. □

Több egyéb tengely is rendelkezik rövid leírással. Például a .. szülő helyén áll, a . az önmaga helyén. Már láttuk a @ jelet az attribútumra és a / jelet a gyerekre.

12.1.7. A kifejezések szövegösszefüggése

Hogy megértsük, mit is jelent egy olyan tengely, mint a szülő, további vizsgálatokra van szükség XPath adatainak kinézetével kapcsolatban. Egy kifejezés eredménye elemek vagy egyszerű értékek szekvenciája. Az XPath-kifejezések és eredményeik azonban nincsenek különválasztva. Ha így lenne, akkor nem lenne értelme egy elem szülőjét kérni. Ez normális abban a szövegkörnyezetben, ahol a kifejezés kiértékelhető. Példáinkban egy szimpla dokumentumunk volt, amelyből az elemeket kibontottuk. Amennyiben úgy gondoljuk, hogy egy bizonyos XPath-kifejezés eredménye egy hivatkozás a dokumentumbeli elemre, akkor alkalmazhatjuk az illeszkedés megtalálására azokat a tengelyeket, mint az elem szülője, őse vagy a következő testvére a szekvenciában.

Említettük például a 11.4.6. alfejezetben, hogy a kulcsok egy XML-sémában definiálhatók XPath-kifejezések párosával. A kulcsmegszorításoknak az XML-dokumentumokra való alkalmazása azt jelenti, hogy eleget tesznek annak a sémának, amely a megszorításokat tartalmazza. Az összes ilyen dokumentum egy szövegkörnyezetet ad magában a sémában szereplő XPath-kifejezés számára. Az ilyen kifejezésekben ezért szabad használni az összes Xpath-tengelyt.

12.1.8. Helyettesítő jelek

Ahelyett, hogy specifikálnánk egy jelölőt az útvonal minden lépésében, használhatjuk a * jelet a „tetszőleges jelölő" értelemben. Hasonlóan, egy attribútum specifikálása helyett a „tetszőleges attribútumra"-ra használható a @* jel.

12.5. példa. Alkalmazzuk a

```
/FilmSzínészAdat/*/@*
```

útkifejezést a 12.2. ábra dokumentumára. Elsőként a `/FilmSzínészAdat/*` a gyökérelem összes gyerekelemét szolgáltatja. A következő hármat: két színészt és egy filmet. Így ennek az útkifejezésnek az eredménye a 3. és 13. közötti, a 14. és 18. közötti, illetve a 19. és 22. sorok közötti soroknak a szekvenciája.

Azonban a kifejezés kéri ezen elemek összes attribútumának értékét. Ennek következtében vizsgálódunk ezeknek az elemeknek a legkülső jelölői között, és visszaadjuk ezek értékeit abban a sorrendben, ahogy azok a dokumentumban elhelyezkednek. Így a

```
"cf" "csh" "mh" "csh" "csh" "cf" "mh"
```

szekvencia az eredménye az Xpath-lekérdezésnek.

A kényes pont az, hogy a `Szereplő` attribútum 19. sorban található értéke önmagában tartalmazza az elemek szekvenciáját – a `"cf"` és `"mh"` szövegeket. Az XPath kiterjeszti azokat a szekvenciákat, amelyek más szekvenciák részei, így minden tétel a „felső szintre" kerül, ahogy fent bemutattuk. Azaz a tételek szekvenciája önmagában nem tétel. □

12.1.9. Feltételek az útkifejezésekben

Amikor egy útkifejezést kiértékelünk, ezt korlátozhatjuk abban, hogy csak egy részhalmazát kövessük azoknak az utaknak, amelyek jelölői illeszthetők a kifejezés jelölőire. Ennek megvalósítására egy feltétellel ellátott jelölőt alkalmazunk. Az ilyen feltétel szögletes zárójelek között szerepel. Ez a feltétel bármi lehet, aminek logikai értéke van. Értékek hasonlíthatók össze olyan operátorokkal, mint az = vagy a >=. A „nem egyelőt" a C nyelvhez hasonlóan a `!=` reprezentálja. Összetett feltételt a feltételekkel és a köztük szereplő `or` vagy `and` operátorokkal készíthetünk.

Az értékek összehasonlítása alkalmazható útkifejezésekre. Ebben az esetben az összehasonlítás a kifejezés által előállított szekvenciákra vonatkozik. Az összehasonlításoknak van egy „létezik" jelentése; két szekvenciára teljesül az összehasonlítási reláció, amennyiben létezik egy tételpár, egy-egy tétel mindkét szekvenciából, amelyekre teljesül a megadott összehasonlító operátor. Tekintsünk egy példát, amely világossá teheti ezt a felvetést.

12.6. példa. A következő útkifejezés:

```
/FilmSzínészAdat/Színész[//Város = "Malibu"]/Név
```

azoknak a színészeknek a nevét adja vissza, akiknek legalább az egyik otthonuk Malibuban van. Kezdetben a `/FilmSzínészAdat/Színész` útkifejezés az összes `Színész` elem szekvenciáját adja vissza. Ezen elemek mindegyikére ki kell értékelnünk a `//Város = "Malibu"` feltétel igazságát. Itt a `//Város` egy útkifejezés, de – hasonlóan más, a feltételekben szereplő útkifejezésekhez – relatív ahhoz az elemhez, amely az útkifejezésben őt tartalmazza. Így azt feltételezve alkalmazzuk a kifejezést, hogy az elem, amelyre alkalmaztuk, a teljes dokumentum volt.

Kezdjük a Carrie Fisherhez tartozó elemmel, amely a 3. és 13. sorok között található a 12.2. ábrán. A `//Város` kifejezés arra sarkall minket, hogy megvizsgáljuk az összes gyerekelemet, amely nulla vagy több szint mélységben van beágyazva, és rendelkezik `Város` jelölővel. Ebből kettő van, a 7. és 11. sorokban. A Carrie Fisher elemre alkalmazott `//Város` útkifejezésnek az eredménye a következő szekvencia:

```
<Város>Hollywood</Város>
<Város>Malibu</Város>
```

Ennek a szekvenciának minden tétele összehasonlításra kerül a `"Malibu"` értékkel. Egy olyan elem, amelynek értéke egyszerű (például szöveg) típus, egyenlő lehet ezzel a szöveggel, így a második adat átmegy a teszten. Mint eredmény, a 3. és 13. sorok közötti teljes `Színész` elem kielégíti a feltételt.

Amikor alkalmazzuk a 14. és 18. sorok közötti Mark Hamillre vonatkozó második tételre a feltételt, akkor találunk egy `Város` gyerekelemet, de ennek értéke nem egyezik a `"Malibu"`-val, így ez az elem nem tesz eleget a feltételnek. Így csak a Carrie Fisher elem az eredménye a

```
/FilmSzínészAdat/Színész[//Város = "Malibu"]
```

útkifejezésnek.

Az Xpath-lekérdezés befejezéséhez alkalmaznunk kell erre az egy elemmel rendelkező szekvenciára az útkifejezés folytatását, a `/Név`-et. Ezen a ponton egy `Név` gyerekelemet keresünk Carrie Fisher elemében, és meg is találjuk a 4. sorban. Következésképpen, a lekérdezés eredménye az egy elemből álló `<Név>Carrie Fisher</Név>` szekvencia. □

Néhány egyéb hasznos feltétel a következő:

- Egy $[i]$ egész kizárólag a szülő i-edik gyerekére való alkalmazással igaz.
- Egy $[T]$ jelölő kizárólag azokra az elemekre igaz, amelyek egy vagy több T jelölőjű gyerekelemmel rendelkeznek.
- Hasonlóan, egy $[A]$ attribútum kizárólag azokra az elemekre igaz, amelyek rendelkeznek értékkel az A attribútumra.

12.7. példa. A 12.3. ábra egy variációja az általunk használt filmes példának, amelyben az összes filmet csoportosítottuk a közös címük szerint egy `Film` elemben, amelynek gyerekelemei `Verzió` jelölővel rendelkeznek. A cím egy attribútuma a filmnek, és az évszám egy attribútuma a verziónak. A verziók rendelkeznek a `Színész` gyerekelemmel. Tekintsük a következő Xpath-lekérdezést erre a dokumentumra alkalmazva:

```
/Filmek/Film/Verzió[1]/@év
```

Ez az összes film első verziójának az elkészítési évét kérdezi és eredménye az `"1933" "1984"` szekvencia.

Részletesebben négy `Verzió` elem illeszkedik a

```
/Filmek/Film/Verzió
```

útvonalra. Ezek egyenként a 4. és 6. közötti sorokban, a 7. és 10. közötti sorokban, a 11. sorban és a 14. és 18. közötti sorokban találhatók. Ezek közül az első és az utolsó az első gyermekei a saját szüleiknek. Ezen verzióknak az `év` attribútuma egyenként az 1933 és az 1984. □

```
1)  <? xml version="1.0" encoding="utf-8" standalone="yes" ?>
2)  <Filmek>
3)      <Film filmcím = "King Kong">
4)          <Verzió év = "1933">
5)              <Színész>Fay Wray</Színész>
6)          </Verzió>
7)          <Verzió év = "1976">
8)              <Színész>Jeff Bridges</Színész>
9)              <Színész>Jessica Lange</Színész>
10)         </Verzió>
11)         <Verzió év = "2005" />
12)     </Film>
13)     <Film filmcím = "Gumiláb">
14)         <Verzió év = "1984">
15)             <Színész>Kevin Bacon</Színész>
16)             <Színész>John Lithgow</Színész>
17)             <Színész>Sarah Jessica Parker</Színész>
18)         </Verzió>
19)     </Film>
20) </Filmek>
```

12.3. ábra. Egy XML-dokumentum az útkifejezések feldolgozására

12.8. példa. A

```
/Filmek/Film/Verzió[Színész]
```

Xpath-lekérdezés alkalmazása a 12.3. ábra dokumentumára három `Verzió` elemet szolgáltat. A `[Színész]` a „van legalább egy `Színész` gyerekeleme" feltételt interpretálja. Ez a feltétel a 4. és 6. közötti, a 7. és 10. közötti és a 14. és 18. közötti sorokban található `Verzió` elemekre igaz, ugyanakkor hamis a 11. sor elemére. □

12.1.10. Feladatok

12.1.1. feladat. A 12.4. és 12.5. ábrák az eleje és a vége egy olyan XML-dokumentumnak, amely néhány adatot tartalmaz termékekről. Adjuk meg a következő Xpath-lekérdezéseket. Mi lesz ezek eredménye?

a) Keressünk meg a memória méretét az összes PC-ben.

b) Keressük meg mindegyik termék árát.

c) Keressük meg az összes nyomtató elemet.

! *d)* Keressük meg a lézernyomtatók gyártóit.

! *e)* Keressük meg a PC-k és/vagy laptopok gyártóit.

f) Keressük meg a modellszámát azoknak a PC-knek, amelyeknek a merevlemeze legalább 200 gigabájtos.

!! *g)* Keressük meg azokat a gyártókat, amelyek legalább kétféle PC-t gyártanak.

12.1.2. feladat. A 12.6. ábra dokumentuma hasonló adatokat tartalmaz a már használt csatahajós példánkhoz. Ebben a dokumentumban a hajók adatai a saját osztályuk elemeibe vannak ágyazva, és a hajókról szóló információk a hajóelemekben vannak. Adjuk meg a következő lekérdezéseket XPath-ban. Mi lesz ezek eredménye?

a) Keressük meg minden hajó nevét.

b) Keressük meg az összes olyan `Hajóosztály` elemet, amely osztályoknál a vízkiszorítás nagyobb, mint 35 000.

c) Keressük meg az összes olyan `Hajó` elemet, amely hajóknál a vízre bocsátás 1917 előtti.

d) Keressük meg az összes elsüllyedt hajót.

! *e)* Keressük meg az összes olyan évet, amikor vízre kerültek olyan hajók, amelyeknél a hajó és az osztály neve megegyező.

! *f)* Keressük meg azoknak a hajóknak a nevét, amelyek részt vettek csatákban.

!! *g)* Keressük meg az összes olyan `Hajó` elemet a hajók közül, amelyek kettő vagy több csatában vettek részt.

```
<Termékek>
    <Gyártó név = "A">
        <PC modell = "1001" ár = "2114">
            <Sebesség>2.66</Sebesség>
            <Memória>1024</Memória>
            <Merevlemez>250</Merevlemez>
        </PC>
        <PC modell = "1002" ár = "995">
            <Sebesség>2.10</Sebesség>
            <Memória>512</Memória>
            <Merevlemez>250</Merevlemez>
        </PC>
        <Laptop modell = "2004" ár = "1150">
            <Sebesség>2.00</Sebesség>
            <Memória>512</Memória>
            <Merevlemez>60</Merevlemez>
            <Képernyő>13.3</Képernyő>
        </Laptop>
        <Laptop modell = "2005" ár = "2500">
            <Sebesség>2.16</Sebesség>
            <Memória>1024</Memória>
            <Merevlemez>120</Merevlemez>
            <Képernyő>17.0</Képernyő>
        </Laptop>
    </Gyártó>
```

12.4. ábra. XML-dokumentum gyártási adatokkal – kezdete

```
    <Gyártó név = "E">
        <PC modell = "1011" ár = "959">
            <Sebesség>1.86</Sebesség>
            <Memória>2048</Memória>
            <Merevlemez>160</Merevlemez>
        </PC>
        <PC modell = "1012" ár = "649">
            <Sebesség>2.80</Sebesség>
            <Memória>1024</Memória>
            <Merevlemez>160</Merevlemez>
        </PC>
        <Laptop modell = "2001" ár = "3673">
            <Sebesség>2.00</Sebesség>
            <Memória>2048</Memória>
            <Merevlemez>240</Merevlemez>
            <Képernyő>20.1</Képernyő>
        </Laptop>
        <Nyomtató modell = "3002" ár = "239">
            <Színes>hamis</Színes>
            <Típus>lézer</Típus>
        </Nyomtató>
    <Gyártó név = "H">
        <Nyomtató modell = "3006" ár = "100">
            <Színes>igaz</Színes>
            <Típus>tintasugaras</Típus>
        </Nyomtató>
        <Nyomtató modell = "3007" ár = "200">
            <Színes>igaz</Színes>
            <Típus>lézer</Típus>
        </Nyomtató>
    </Gyártó>
</Termékek>
```

12.5. ábra. XML-dokumentum gyártási adatokkal – befejezése

```
<Hajók>
    <Hajóosztály név = "Kongo" típus = "bc" ország = "Japan"
            ágyúkSzáma = "8" kaliber = "14"
            vízkiszorítás = "32000">
        <Hajó név = "Kongo" felavatva = "1913" />
        <Hajó név = "Hiei" felavatva = "1914" />
        <Hajó név = "Kirishima" felavatva = "1915">
            <Csata kimenetel = "elsüllyedt">Guadalcanal</Csata>
        </Hajó>
        <Hajó név = "Haruna" felavatva = "1915" />
    </Hajóosztály>
    <Hajóosztály név = "North Carolina" típus = "bb"
            ország = "USA" ágyúkSzáma = "9" kaliber = "16"
            vízkiszorítás = "37000">
        <Hajó név = "North Carolina" felavatva = "1941" />
        <Hajó név = "Washington" felavatva = "1941">
            <Csata kimenetel = "ép">Guadalcanal</Csata>
        </Hajó>
    </Hajóosztály>
    <Hajóosztály név = "Tennessee" típus = "bb" ország = "USA"
            ágyúkSzáma = "12" kaliber = "14"
            vízkiszorítás = "32000">
        <Hajó név = "Tennessee" felavatva = "1920">
            <Csata kimenetel = "ép">Surigao Strait</Csata>
        </Hajó>
        <Hajó név = "California" felavatva = "1921">
            <Csata kimenetel = "ép">Surigao Strait</Csata>
    </Hajóosztály>
    <Hajóosztály név = "King George V" típus = "bb"
            ország = "Great Britain" ágyúkSzáma = "10"
            kaliber = "14" vízkiszorítás = "32000">
        <Hajó név = "King George V" felavatva = "1940" />
        <Hajó név = "Prince of Wales" felavatva = "1941">
            <Csata kimenetel = "sérült">Denmark Strait</Csata>
            <Csata kimenetel = "elsüllyedt">Malaya</Csata>
        </Hajó>
        <Hajó név = "Duke of York" felavatva = "1941">
            <Csata kimenetel = "ép">North Cape</Csata>
        </Hajó>
        <Hajó név = "Howe" felavatva = "1942" />
        <Hajó név = "Anson" felavatva = "1942" />
    </Hajóosztály>
</Hajók>
```

12.6. ábra. Csatahajók adatait tartalmazó XML-dokumentum

12.2. XQuery

Az XQuery az XPath egy kiterjesztése, amely azoknak az adatbázisoknak vált a szabványos magas szintű lekérdezési nyelvévé, amelyek az adatokat XML-formában tárolják. Ebben a fejezetben bemutatunk néhányat az XQuery fontos képességei közül.

12.2.1. XQuery-alapismeretek

Az XQuery ugyanolyan modellt használ az értékek számára, mint amilyet az XPath kapcsán bemutattunk a 12.1.1. alfejezetben. Ez pedig az, hogy az értékeket az XQuery tételek szekvenciájaként állítja elő. A tételek lehetnek egyszerű értékek vagy különböző típusú csomópontok, beleértve ebbe az elemeket, attribútumokat és a dokumentumokat. A szekvencia elemei valamilyen létező dokumentumokból hozhatóak át, ahogy azt a 12.1.7. alfejezetben tárgyaltuk.

Az XQuery egy *funkcionális nyelv*, amely magában foglalja, hogy tetszőleges XQuery-kifejezést használhatunk bármilyen olyan helyen, ahol kifejezés állhat. Ez a tulajdonság nagyon erős. Például az SQL sok helyen megenged ugyan allekérdezéseket, de az SQL-ben nem megengedett, hogy egy allekérdezés egy összehasonlítás operandusa legyen egy where feltételben. A funkcionális tulajdonság kétélű fegyver. Megköveteli azt, hogy az XQuery minden operátora értelmet kapjon akkor is, amikor egynél több adatra alkalmazzuk, amelynek váratlan következményei lehetnek.

Kezdetben minden XPath-kifejezés egy XQuery-kifejezés. Az XQuery azonban ennél többet is ad az FLWR (ejtsd: mint az angol „flower" szót) kifejezések használatával, ami bizonyos értelemben analóg az SQL select-from-where kifejezésével.

12.2.2. FLWR-kifejezések

Az XPath-kifejezéseken kívül az XQuery-kifejezések legfontosabb formáiban négy típusú záradék használható fel, melyeket for, let, where és a return (FLWR) záradékoknak neveznek.[1] Ebben a részben ezeknek a záradékoknak a formáját fogjuk bemutatni. Mindig legyünk körültekintőek ezeknek a záradékoknak az esetében a sorrend és az előfordulás figyelembevételére.

1. A lekérdezés nulla vagy több for és let záradékkal kezdődik. Mindegyik típusból egynél több is lehet, és tetszőleges sorrendben egymásba fűzhetők, például for, for, let, for, let.

2. Ezután jöhet egy opcionális where záradék.

3. Végezetül pontosan egy return záradéknak kell állnia.

[1] Van egy order-by záradék is, amelyet a 12.2.10. alfejezetben fogunk bemutatni. Ezen okból az FLWR egy kevésbé pontos betűszó az XQuery-lekérdezések fő formulájára, ami az FLWOR.

Az XQuery kis- és nagybetű-érzékenysége

Az XQuery érzékény a kis- és nagybetűkre. Így a kulcsszavakat, mint például a `let` vagy a `for` kisbetűvel kell írni, hasonlóan ahhoz, ahogy a Java vagy a C nyelvben.

12.9. példa. Valószínűleg a legegyszerűbb FLWR-kifejezés a következő:

```
return <Üdvözlet>Helló Világ</Üdvözlet>
```

Nem vizsgál adatot, és egyetlen értéket ad, ami egy XML-elem. □

A let záradék

A let záradék egyszerű alakja:

`let` *változó* `:=` *kifejezés*

Ennek a záradéknak a célja, hogy kiértékelje és értékül adja a változónak a kifejezést a megmaradó FLWR-kifejezések számára. Az XQuery változóinak dollárjellel kell kezdődnie. Figyeljünk arra, hogy az értékadás szimbóluma a :=, nem egy egyenlőségjel (ami az XPath-ban összehasonlításokban használatos). Általánosabban, változó-értékadások vesszőkkel elválasztott listája állhat azon a helyen, ahol példánkban mi csak egy ilyet mutattunk.

12.10. példa. A let záradék egyik használata, hogy egy változónak értékül adunk egy hivatkozást egy olyan dokumentumra, amely adatait a lekérdezésben használjuk. Például, amennyiben lekérdezést szeretnénk végrehajtani egy a `Színészek.xml` fájlban tárolt dokumentumra, kezdhetjük a lekérdezésünket a következővel:

```
let $Színészek := doc("Színészek.xml")
```

A továbbiakban a `$Színészek` értéke egy egyszerű doc csomópont. Egy XPath-kifejezés előtt ezt használhatjuk, és a kifejezés feldolgozásra kerül egy olyan XML-dokumentumra vonatkozólag, amelyet a `Színészek.xml` fájl tartalmaz. □

A for záradék

A for záradék egyszerű alakja:

`for` *változó* `in` *kifejezés*

A cél a kifejezés kiértékelése. Tetszőleges kifejezés eredménye a tételek szekvenciája lesz. A változónak értékadásra kerül mindegyik tétel egyesével, és a lekérdezés for záradékot követő része kiértékelésre kerül a változó minden egyes értékére. Nem vezet félre minket, ha párhuzamot vonunk egy XQuery for záradéka és egy C nyelvbeli for utasítás között. Általánosabban, egy for záradékban több változó is szerepelhet, és mindegyik más-más tételsorozatot futhat végig.

12.11. példa. Ennek az alfejezetnek számos példájában használni fogjuk a 12.7. ábrán bemutatott adatokat. Az adatok két fájlban találhatók, a 12.7 (a) ábra `Színészek.xml` fájljában és a 12.7 (b) ábra `filmek.xml` fájljában. Ezen két fájl adatai ugyanazok, amelyeket a 12.1. alfejezetben használtunk, ám a cél most az, hogy megmutassuk, mi ezeknek a fájloknak a jelenlegi tartalma.

Képzeljük el, hogy a lekérdezést az alábbi módon kezdjük:

```
let $filmek := doc("filmek.xml")
for $m in $filmek/Filmek/Film
    ... valamit végzünk az összes Film elemmel
```

Figyeljük meg, hogy a `$filmek/Filmek/Film` egy XPath-kifejezés, amelynek jelentése az, hogy kezdjünk a `filmek.xml` fájlban található dokumentummal, majd menjünk a gyökér `Filmek` elembe, aztán folytassuk az összes `Film` gyerekelem szekvenciájával. A „for ciklus" magjának első végrehajtásakor a `$m` egyenlő a 12.7. ábra 17. és 26. közötti soraival, majd a `$m` egyenlő a 27. és 33. közötti soraival, ezután folytatódik a dokumentum összes hátralévő `Film` elemével. □

A where záradék

A where záradék formája:

`where` *feltétel*

Ezt a záradékot egy tételre alkalmazhatjuk, és a *feltétel* – amely egy kifejezés – igaz vagy hamis értéket szolgáltat. Amennyiben az érték igaz, akkor return záradék kerül érvényesítésre a lekérdezésben szereplő változók aktuális értékeire. Ellenkező esetben semmi nem keletkezik a változók aktuális értékeiből.

A return záradék

Ennek a záradéknak az alakja:

`return` *kifejezés*

Egy FLWR-kifejezés eredménye – hasonlóan bármely XQuery-kifejezéshez – tételek szekvenciája. A tételek szekvenciáját a return záradékhoz csatolt kifejezés állítja elő az addig előállított tételek szekvenciájából. Figyeljünk arra, hogy bár egyetlen return záradék van, ez a záradék többször is végrehajtódhat egy „for ciklus" belsejében, így a lekérdezés eredménye több fázisban is előállhat. Ne gondoljuk a return záradékot egy „return utasítás"-nak, mivel ez nem zárja le a lekérdezés végrehajtását.

```
<? xml version="1.0" encoding="utf-8" standalone="yes" ?>
<Színészek>
    <Színész>
        <Név>Carrie Fisher</Név>
        <Cím>
            <Utca>123 Maple St.</Utca>
            <Város>Hollywood</Város>
        </Cím>
        <Cím>
            <Utca>5 Locust Ln.</Utca>
            <Város>Malibu</Város>
        </Cím>
    </Színész>
        ... további színészek
</Színészek>
```

(a) A `színészek.xml` dokumentum

```
<? xml version="1.0" encoding="utf-8" standalone="yes" ?>
<Filmek>
    <Film filmcím = "King Kong">
        <Verzió év = "1933">
            <Színész>Fay Wray</Színész>
        </Verzió>
        <Verzió év = "1976">
            <Színész>Jeff Bridges</Színész>
            <Színész>Jessica Lange</Színész>
        </Verzió>
        <Verzió év = "2005" />
    </Film>
    <Film filmcím = "Gumiláb">
        <Verzió év = "1984">
            <Színész>Kevin Bacon</Színész>
            <Színész>John Lithgow</Színész>
            <Színész>Sarah Jessica Parker</Színész>
        </Verzió>
    </Film>
        ... további filmek
</Filmek>
```

(b) A `filmek.xml` dokumentum

12.7. ábra. Adatok az XQuery-példák számára

Az XQuery logikai értékei

Egy a `$x = 10` alakhoz hasonlító összehasonlítás kiértékelése true vagy false eredményre vezet (szigorúan véve `xs:true` vagy az `xs:false` nevek egyikére az XML-séma névteréből). Azonban az egyéb kifejezéstípusok közül többen is felvehetik a true vagy false értéket a kiértékelésükkor, és így egy where záradék feltételévé tehetők. A következő fontos korlátozásokat kell megjegyeznünk:

1. Amennyiben az érték adatok szekvenciája, akkor az üres szekvenciát hamis, a nem üres szekvenciát igaz értékkel értelmezzük.
2. A számok között a 0 és a NaN („nem egy szám – not a number", ami lényegében egy sokjegyű szám) hamis, az egyéb számok igazak.
3. A szövegek között az üres szöveg hamis, az egyéb szövegek igazak.

12.12. példa. Egészítsük ki a 12.11. példában elkezdett lekérdezést az összes `Színész` elemre vonatkozó kérdéssel, amely elemet a filmek összes verziójában találhatjuk meg. A lekérdezés a következő:

```
let $filmek := doc("filmek.xml")
for $m in $filmek/Filmek/Film
return $m/Verzió/Színész
```

A `$m` első értéke a „for ciklusban" a 12.7. ábra 17. és 26. sorai között található elem. A `Film` elemből a `/Verzió/Színész` XPath-kifejezés a 19., 22. és 23. sorokban található három `Színész` elem szekvenciáját állítja elő. Ez a szekvencia lesz a kezdete a lekérdezés eredményének.

```
<Színész>Fay Wray</Színész>
<Színész>Jeff Bridges</Színész>
<Színész>Jessica Lange</Színész>
<Színész>Kevin Bacon</Színész>
<Színész>John Lithgow</Színész>
<Színész>Sarah Jessica Parker</Színész>
    ...
```

12.8. ábra. A 12.12. példa lekérdezésének eredményszekvencia kezdete

A `$m` következő értéke a 27. és 33. közötti sorokban található elem. Ily módon a return záradékban található kifejezés eredménye a 29., 30. és 31. sorokban található elemek szekvenciája. Így az eredményszekvencia kezdete a 12.8. ábrához hasonlóan néz ki. □

Szekvenciák szekvenciája

Emlékeztetni szeretnénk az Olvasót arra, hogy a tételek szekvenciájának nincs belső szerkezete. Így a 12.8. ábrán nincs elválasztó a Jessica Lange és Kevin Bacon, vagy az első három színész és az utolsó három színész csoportja között, még ha úgy is gondoljuk, hogy ezek a return záradék különböző végrehajtásai során jöttek létre.

12.2.3. Változók helyettesítése értékeikkel

Tekintsük a 12.12. példa lekérdezésének egy módosítását. Itt most nem csak a `<Színész>` elemek szekvenciáját szeretnénk előállítani, hanem a `Film` elemekét úgy, hogy ezek mindegyike tartalmazza egy adott című film minden színészét, tekintet nélkül arra, hogy melyik verzióban szerepeltek. A cím a `Film` elem egy attribútuma lesz.

```
let $filmek := doc("filmek.xml")
for $m in $filmek/Filmek/Film
return <Film filmcím = $m/@filmcím>$m/Verzió/Színész</Film>
```

12.9. ábra. Hibás kísérlet a `Film` elemek előállítására

A 12.9. ábra egy jónak tűnő kísérletet mutat, de valójában ez nem jó. A kifejezés, amelyet a `$m` minden értékére visszakapunk, olyannak tűnik, mintha egy nyitó `<Film>` jelölőt követne a `<Színész>` elemek szekvenciájában, majd végül egy `</Film>` jelölő zárná. A `<Film>` jelölő rendelkezik egy `filmcím` attribútummal, amely a másolata a `Film` elem azonos attribútumának a `filmek.xml` fájlban. Azonban amikor ezt a programot végrehajtjuk, akkor a következőt kapjuk:

```
<Film filmcím = "$m/@filmcím">$m/Verzió/Színész</Film>
<Film filmcím = "$m/@filmcím">$m/Verzió/Színész</Film>
    ...
```

A probléma az, hogy vagy a jelölők között vagy egy attribútum értékeként bármilyen szöveg megengedett. A return utasításban semmilyen különbség nem látszik a XQuery-processzor számára a 12.9. példa return utasításához képest, ahol tényleg egy szöveget állítottunk elő a megfelelő jelölők belsejében. Ahhoz, hogy egy XQuery-kifejezésnek megfelelő szöveget állítsunk elő a jelölők belsejében, a szöveget kapcsos zárójelekkel kell körülvennünk.

A célunk eléréséhez vezető helyes megoldást a 12.10. ábra mutatja. Ebben a lekérdezésben a zárójelben álló `$m/filmcím` és a `$m/Verzió/Színész` kifejezések XPath-kifejezésekként kerülnek értelmezésre. Az elsőt egy szöveg helyettesíti, a másodikat pedig `Színész` elemek szekvenciája, ahogy kívántuk.

```
let $filmek := doc("filmek.xml")
for $m in $filmek/Filmek/Film
return <Film filmcím = {$m/@filmcím}>{$m/Verzió/Színész}
       </Film>
```

12.10. ábra. Kapcsos zárójelek hozzáadásával megoldjuk a problémát

12.13. példa. Ez a példa nem csak a kapcsos zárójeles kifejezések interpretálását mutatja be, hanem azt is hangsúlyozza, hogy az XQuery-kifejezések hogyan használhatók, ahol ezen kifejezések megengedettek. A célunk most az, hogy duplázzuk meg a 12.12. példa eredményét, ahol a `Színész` elemek szekvenciáját kaptuk, de úgy tegyük ezt, hogy a színészek teljes szekvenciája egy `Színészek` elemben legyen. Nem tudjuk használni a 12.10. ábra trükkjét arra, hogy a `Színész` elemek helye a `Színészek` belsejében legyen, mivel ez sok különálló `Színészek` jelölőt hozna létre, amely elválasztaná a színészek csoportjait.

```
let $színészSeq := (
    let $filmek := doc("filmek.xml")
    for $m in $filmek/Filmek/Film
    return $m/Verzió/Színész
)
return <Színészek>{$színészSeq}</Színészek>
```

12.11. ábra. Jelölők elhelyezése a szekvencia köré

A 12.11. ábra megoldja a feladatot. A helyi `$színészSeq` változónak értékül adjuk a `Színész` elemeknek azt a szekvenciáját, amelyet a 12.12. példa eredményeként kaptunk. Amikor visszaadjuk ezt a szekvenciát, a jelölők közé helyezzük, figyelve arra, hogy a változót zárójelekkel zárjuk körül, így ez kiszámításra kerül és nem egyszerű szövegként kezeljük. □

12.2.4. Összekapcsolások az XQueryben

Az XQueryben össze tudunk kapcsolni két vagy több dokumentumot, nagyon hasonló módon ahhoz, ahogy az SQL-ben összekapcsolunk két vagy több relációt. Minden esetben szükségünk van változókra, amelyek hatóköre egy adott dokumentum elemeire, illetve egy adott reláció soraira vonatkoznak. Az SQL-ben egy from záradékot használunk a szükséges sorok előállítására (amely lehet egyetlen táblanév is önállóan), míg az XQueryben egy for záradékot használunk.

Azonban nagyon óvatosnak kell lennünk abban, hogy miként használjuk az összehasonlítást egy összekapcsolásban. Először is itt van az olyan összehasonlító operátor problémája a szekvenciákra vonatkoztatva – mint az = vagy a < operátor – a „léteznek elemek ebben az összehasonlításban" jelentésével, amelyről a 12.1.9. alfejezetben beszéltünk. Visszatérünk erre a kérdésre a 12.2.5. alfejezetben. Ezen felül probléma van ez elemek egyenlőségével az „elemazonosság" miatt (hasonlóan az „objektumazonossághoz"). Ez pedig az, hogy egy elem nem egyenlő egy másik elemmel még akkor sem, ha karakterről karakterre egyezik. Szerencsére általában nem akarunk elemeket összehasonlítani, csak egyszerű értékeket, mint a szöveg és az egész szám, amely az elemek attribútumértékeként vagy gyerekelemük értékeként megjelennek. Az összehasonlító operátorok elvárásunknak megfelelően működnek az egyszerű értékeken; < jelentése a szövegek esetén „a lexikografikus sorrendben megelőzés".

Létezik egy `data(E)` beépített függvény, amellyel az E elem értékét kaphatjuk meg. Ezt a függvényt arra tudjuk használni, hogy megkapjuk egy elemből a megfelelő jelölők között található szöveget.

12.14. példa. Tegyük fel, hogy azokat a városokat szeretnénk megtalálni, amelyekben a 12.7 (b) ábrán látható `filmek.xml` fájlban előforduló színészek élnek. Ennek az információnak a megszerzése érdekében segítségül hívjuk a 12.7 (a) ábra `színészek.xml` fájlját. Így be kell állítanunk egy olyan változót, amelynek hatóköre a `filmek.xml` fájl `Színész` elemei, és egy másikat a `színészek.xml` fájl `Színész` elemeire vonatkozó hatókörrel. Amikor a `filmek.xml` fájl `Színész` elemének adata megegyezik a `színészek.xml` fájl `Színész` elemének a `Név` nevű gyerekelemével, akkor egyezésünk van, és kiszedhetjük a `Város` elemet az utóbbiból.

A 12.12. ábra mutatja a megoldást. A let záradék a változókat a két dokumentumra állítja. Ahogy korábban említettük, ez a rövidített leírás nem szükséges, és használhatjuk magukat a dokumentum-csomópontokat is a következő két sor XPath-kifejezéseiben. A for záradék egy dupla beágyazott ciklust tartalmaz. A `$s1` változó hatóköre a `filmek.xml` fájl összes `Színész` eleme, a `$s2` szerepe ugyanez a `Színészek.xml` fájlra.

```
let $filmek := doc("filmek.xml"),
    $színészek  := doc("színészek.xml")
for $s1 in $filmek/Filmek/Film/Verzió/Színész,
    $s2 in $színészek/Színészek/Színész
where data($s1) = data($s2/Név)
return $s2/Cím/Város
```

12.12. ábra. A színészek városainak megkeresése

A where záradék a `$s1` és `$s2` elemek értékeként szolgáló szövegek kivonatolására a beépített `data` funkciót használja. Végül a return záradék előállítja a `Város` elemet. □

12.2.5. Az XQuery összehasonlító operátorai

Most egy másik olyan keresztrejtvényt tekintsünk, amely nem az általunk várt módon működik. A célunk, hogy megtaláljuk azokat a színészeket a 12.7 (a) ábra `színészek.xml` fájlban, akiknek a lakcíme 123 Maple St., Malibu. Első kísérletünk a 12.13. ábrán látható.

```
let $színészek := doc("színészek.xml")
for $s in $színészek/Színészek/Színész
where $s/Cím/Utca = "123 Maple St." and
      $s/Cím/Város = "Malibu"
return $s/Név
```

12.13. ábra. Egy hibás kísérlet arra, hogy megtaláljuk azokat, akik 123 Maple St., Malibu címen laknak

A where záradékban az `Utca` és a `Város` elemet szövegként hasonlítjuk össze, ám ezek pontosan úgy működnek, ahogy elvárjuk, azaz ha egy szöveg értékű elem találkozik egy szöveggel, az összehasonlítás pontosan akkor lesz sikeres, amikor elvárjuk. A probléma akkor keletkezik, amikor a `$s` értékül kapja a 12.7. ábra 3. és 13. sorok közötti `Színész` elemét. Ekkor a `$s/Cím/Utca` XPath-kifejezés a 6. és 10. sorok kételemű szekvenciáját állítja elő. Mivel az = operátor minden tételpárra igaz értéket szolgáltat, amikor a két oldalról egy-egy tétel egyenlő, az első összehasonlítás értéke igaz lesz; a 6. sor egyenlő a „123 Maple St." szöveggel. Hasonlóan, a második feltétel a 7. és 11. sorokban található két `Város` elem listáját hasonlítja össze a „Malibu" szöveggel, és egyenlőséget talál a 11. sorban. Így eredményként a Carrie Fisher `Név` elemét (4. sor) kapjuk vissza.

Ellenben Carrie Fisher nem a 123 Maple St., Malibu címen lakik. Az ő címe 123 Maple St., Hollywood, és a másik lakcím valahol Malibuban van. Az összehasonlítások egzisztenciális jellege okozott hibát azzal, hogy az utcát és a várost különböző címekből kaptuk.

Az XQuery összehasonlító operátorok egy olyan csoportját biztosítja, amelyekkel csak olyan szekvenciákat tudnak összehasonlítani, amelyek egyetlen tételből állnak, és elbuknak, ha valamelyik operandus több tételből álló szekvencia. Ezek az operátorok az összehasonlítások kétbetűs rövidítései: `eq`, `ne`, `lt`, `gt`, `le` és `ge`. Használjuk az `eq` operátort a = helyén abban az esetben, amikor az összehasonlítandó utcák és városok valójában különböző szövegekben vannak. Az átdolgozott lekérdezést mutatja a 12.14. ábra.

Ez a lekérdezés nem fogja átengedni a Carrie Fisher elemet a where záradék ellenőrzésén, mivel az `eq` operátor bal oldala nem egy egyszerű adat, ennek következtében az összehasonlítás elbukik. Sajnos nem kapunk meg azokról a színészekről sem semmit, akiknek kettő vagy több lakcímük van, még akkor sem, ha ezek között a lakcímek között szerepel a 123 Maple St., Malibu. Egy megfelelő lekérdezés megírása trükkös, függetlenül attól, hogy az összehasonlító operátorok melyik verzióját használjuk. A megfelelő lekérdezés megírását gyakorlásnak hagyjuk.

```
let $színészek := doc("színészek.xml")
for $s in $színészek/Színészek/Színész
where $s/Cím/Utca eq "123 Maple St." and
      $s/Cím/Város eq "Malibu"
return $s/Név
```

12.14. ábra. A második hibás kísérlet azok megtalálására, akik 123 Maple St., Malibu címen laknak

12.2.6. Az ismétlődések kiszűrése

Az XQuery lehetőséget biztosít számunkra a `distinct-values` beépített függvény alkalmazásával arra, hogy bármilyen szekvenciából kiszűrhessük a duplikátumokat. Egy finomságot azonban meg kell jegyeznünk: pontosabban szólva, a `distict-values` az egyszerű típusokra alkalmazható. Leválaszthatjuk a jelölőket egy olyan elemről, amely egy jelölt szöveg, de nem tudjuk visszahelyezni őket. Így a `distinct-values` bemenete lehet elemek egy listája, míg a kimenet szövegek listája.

12.15. példa. A 12.11. ábra összegyűjtötte az összes film összes `Színész` elemét és egy szekvenciaként adta őket vissza. Ám azok a színészek, akik több filmben is szerepeltek, a szekvenciában is többször szerepelnének. A `distinct-values` alkérdés alkalmazásával – amely a `$színészSeq` változó értéke lesz – ki tudunk választani egyet minden egyes `Színész` elem előfordulásaiból. Az új lekérdezést mutatja a 12.15. ábra.

```
let $színészSeq := distinct-values(
    let $filmek := doc("filmek.xml")
    for $m in $filmek/Filmek/Film
    return $m/Verzió/Színész
)
return <Színészek>{$színészSeq}</Színészek>
```

12.15. ábra. Többszörösen előforduló színészek duplikátumainak eltüntetése

Figyeljük meg, hogy ez nevek listáját állítja elő, amelyet körülvettünk `Színészek` jelölőkkel az alábbi módon:

```
<Színészek>"Fay Wray" "Jeff Bridges" ··· </Színészek>
```

Összehasonlítva a 12.11. ábra szerinti változattal, a 12.11. ábra eredménye

```
<Színészek><Színész>Fay Wray</Színész> <Színész>Jeff
        Bridges</Színész> ··· </Színészek>
```

ami azonban duplikátumokat is eredményezhet. □

12.2.7. Kvantifikálás az XQueryben

Vannak olyan kifejezések, amelyek eredményükben azt mondják, hogy „mindegyik" és „létezik". Ezek a szerkezetek egyenként az alábbiak:

every *változó* in *kifejezés1* satisfies *kifejezés2*
some *változó* in *kifejezés1* satisfies *kifejezés2*

Itt a *kifejezés1* tételek szekvenciáját állítja elő, és a változó ciklusban megkapja értékül az egyes tételeket. Ezeknek az értékeknek mindegyikére a *kifejezés2* (amely normál esetben tartalmazza a változót) kiértékelésre kerül, és egy logikai értéket állít elő.

Az „every" változatban az egész kifejezés eredménye hamis, ha van olyan, a *kifejezés1* által előállított tétel, amely hamissá teszi a *kifejezés2*-t; ellenkező esetben igaz lesz az eredmény. A „some" változatban az egész kifejezés eredménye igaz, ha van olyan, a *kifejezés1* által előállított tétel, amely igazzá teszi a *kifejezés2*-t; ellenkező esetben hamis lesz az eredmény.

```
let $színészek := doc("színészek.xml")
for $s in $színészek/Színészek/Színész
where every $c in $s/Cím/Város satisfies
    $c = "Hollywood"
return $s/Név
```

12.16. ábra. A csak Hollywoodban lakó színészek keresése

12.16. példa. A 12.7 (a) ábra `Színészek.xml` fájlját használva keressük azokat a színészeket, akik csak Hollywoodban laknak. Nem számít, hogy hány lakcímük van, de ezeknél a városnak Hollywoodnak kell lennie. A 12.16. ábra mutatja, hogyan kell megírni ezt a lekérdezést. Figyeljük meg, hogy a `$s/Cím/Város` előállítja a `$s` színész `Város` elemeinek szekvenciáját. A where záradék így akkor és csak akkor kerül kielégítésre, ha a lista minden eleme `<Város>Hollywood</Város>`.

Egyébként lecserélhetjük az „every"-t „some"-ra és akkor azokat a színészeket találjuk meg, akiknek legalább egy otthona van Hollywoodban. A „some" változat használata azonban csak ritkán szükséges, mivel az XQuery legtöbb kiértékelése egyébként is meghatározza a létezést. Például a

```
let $színészek := doc("színészek.xml")
for $s in $színészek/Színészek/Színész
where $s/Cím/Város = "Hollywood"
return $s/Név
```

kérdés előállítja a „some" kifejezés használata nélkül azokat a színészeket, akiknek van hollywoodi otthona. Visszautalunk a 12.2.5. alfejezetben tárgyaltakra,

az =-hez hasonló összehasonlítások működésére abban az esetben, amikor az egyik vagy mindkét oldalon egynél több elemet tartalmazó szekvencia áll. Ez akkor lesz igaz, ha megfelel neki egy-egy tetszőleges elem az egyes oldalakról. □

12.2.8. Összesítések

Az XQuery beépített függvényeket biztosít az olyan szokásos összesítések kiszámítására, mint a count, sum vagy a max. Ezek bármilyen szekvenciát kaphatnak argumentumként, azaz bármilyen XQuery-kifejezés eredményére alkalmazhatók.

```
let $filmek := doc("filmek.xml")
for $m in $filmek/Filmek/Film
where count($m/Verzió) > 1
return $m
```

12.17. ábra. A több verzióval rendelkező filmek keresése

12.17. példa. Vizsgáljuk meg az adatokat a 12.7 (b) ábra `filmek.xml` fájljában és állítsuk elő azokat a `Film` elemeket, amelyeknek egynél több verziójuk van. A 12.17. ábra elvégzi ezt a munkát. A `$m/Verzió` XPath-kifejezés előállítja a `$m` film `Verzió` elemeinek szekvenciáját. A szekvenciában található adatokat megszámoljuk. Amennyiben a számlálás 1-nél nagyobb eredményt szolgáltat, a where záradék teljesül, és a film elemet hozzáfűzzük az eredményhez. □

12.2.9. Elágazás az XQuery-kifejezésekben

Egy XQuery if-then-else kifejezés formája

`if (`*kifejezés1*`) then` *kifejezés2* `else` *kifejezés3*

Ennek a kifejezésnek a kiértékeléséhez először kiértékeljük a *kifejezés1*-et. Amennyiben ez igaz, akkor kiértékeljük a *kifejezés2*-t, amely a teljes kifejezés eredménye lesz. Ha a *kifejezés1* hamis, akkor az egész kifejezés eredménye a *kifejezés3* lesz.

Ezek a kifejezések nem utasítások – az XQueryben nincsenek utasítások, csak kifejezések. A C nyelvben szereplő analóg kifejezés az `?:`, nem pedig az if-then-else utasítás. A C-beli kifejezésekhez hasonlóan nincs mód az „else" rész elhagyására. Azonban használhatjuk a *kifejezés3*-at üres kifejezésként, amelyet a `()` jelöl. Ez a választás biztosítja, hogy egy feltételes kifejezés üres szekvenciát készítsen, ha a feltétel nem kielégíthető.

12.18. példa. Ebben a példában az a célunk, hogy előállítsuk a *King Kong* összes verzióját, a legfrissebb verziót `Legfrissebb`-bel, a régebbieket pedig `Régebbi`-vel jelölve. Az 1. sorban beállítjuk a `$kk` változót úgy, hogy értékül

a `King Kong` `Film` elemét kapja. Figyeljük meg, hogy *egy* XPath-feltételt használtunk ebben a sorban azért, hogy biztosak lehessünk abban, hogy csak ezt az egy elemet állítsuk elő. Természetesen amennyiben különböző `Film` elemek is vannak `King Kong` címmel, akkor ezen elemek szekvenciája lenne `$kk` értéke, és a lekérdezésnek nem lenne értelme. Azonban feltételezzük, hogy a filmcím egy kulcs ebben a struktúrában, így egyértelműen csoportosíthatók az azonos című filmek verziói.

```
1)  let $kk :=
            doc("filmek.xml")/Filmek/Film[@filmcím = "King Kong"]
2)  for $v in $kk/Verzió
3)  return
4)      if ($v/@év = max($kk/Verzió/@év))
5)      then <Legfrissebb>{$v}</Legfrissebb>
6)      else <Régebbi>{$v}</Régebbi>
```

12.18. ábra. A *King Kong* verzióinak jelölése

A 2. sor eredményeként a `$v` végighalad a *King Kong* összes verzióján. Mindegyik verzióra egyet kapunk vissza a lehetséges két elem közül. Ahhoz, hogy megmondjuk melyiket, ki kell értékelnünk a 4. sor feltételét. Az egyenlőségjel jobb oldalán a *King Kong* összes verziója közül a legnagyobb évszámmal rendelkező évszáma szerepel, a bal oldalon pedig a `$v` évszáma. Amennyiben ezek egyenlők, akkor a `$v` a legfrissebb verzió, és az 5. sor elemét állítjuk elő. Ha nem, akkor a `$v` egy korábbi verzió, és a 6. sor elemét állítjuk elő. □

12.2.10. Lekérdezés eredményének rendezése

Lehetséges egy FLWR-lekérdezés eredményének rendezése, amennyiben egy order záradékot hozzáveszünk a return záradék előtt. Valójában azt a lekérdezésformát, amivel itt foglalkozunk, általában FLWOR-nek hívják (és ezt is „flower"-nek ejtjük), elfogadva az order záradék opcionális jelenlétét. Ennek a záradéknak a formája:

`order` *list of kifejezések listája*

A rendezés kiindulópontja az első kifejezés értéke, majd az egyezőeket a második kifejezés értéke szerint rendezzük, és így tovább. Az alapértelmezett rendezés a növekvő, de amennyiben `descending` kulcsszó követ egy kifejezést, akkor a rendezés fordított sorrendben történik.

Az, hogy mi történik egy rendezés jelenlétekor, analóg ahhoz, ami az SQL-ben történik. A lekérdezés feldolgozásában a kimenet összegyűjtésének lépése előtt (a `SELECT` záradék az SQL-ben; a return záradék az XQueryben), az előző záradék eredménye összegyűjtésre és rendezésre kerül. Az SQL esetében a

közbülső eredmény a sorváltozók halmazának sorokhoz történő hozzárendelése, ahol a változók befutják a FROM záradék relációit. Speciálisan ezek azok a hozzárendelések, amelyek átmentek a WHERE záradék tesztjén.

Az XQueryben úgy gondolhatjuk, hogy a közbülső eredmény változókhoz történő érték-hozzárendelések szekvenciája. A változók azokban a for és let záradékokban definiáltak, amelyek megelőzik az order záradékot, és a szekvencia azokat a hozzárendeléseket tartalmazza, amelyek átmentek a where záradék tesztelésén. Ezen a hozzárendelések mindegyike kiértékelésre kerül az order záradékban, és ezeknek a kifejezéseknek az értéke határozza meg a hozzárendelés helyét a hozzárendelések sorrendjében. Amikor rendelkezésünkre áll a hozzárendelések sorrendje, akkor használjuk őket a return záradék kifejezésének kiértékelésére.

12.19. példa. Tekintsük az összes film összes verzióját, rendezzük őket évszám szerint, és állítsuk elő Film elemek szekvenciáját a cím és évszám attribútummal. A szokásos módon az adatok a 12.7 (b) filmek.xml fájlból jönnek. A lekérdezést a 12.19. ábra mutatja.

```
let $filmek := doc("filmek.xml")
for $m in $filmek/Filmek/Film,
    $v in $m/Verzió
order $v/@év
return <Film filmcím = "{$m/@filmcím}" év = "{$v/@év}" />
```

12.19. ábra. Az évszám szerint rendezett filmcím-évszám párok szekvenciájának elkészítése

Amikor elérjük az order záradékot, a hozzárendelések a következő három változónak szolgáltatnak értéket: $filmek, $m, $v. Ezen hozzárendelések mindegyikében a doc("filmek.xml") érték van hozzákötve a $filmek-hez. Azonban a $m és $v változók értékei minden párosításban egy filmet és ennek a filmnek egy verzióját tartalmazzák, ezek lesznek egy hozzárendelései ezeknek a változónak. Például ezek közül a hozzárendelések közül az első a $m-hez a 12.7 (b) ábra 17. és 26. sorok közötti elemet rendeli, a $v-hez pedig a 18. és 20. sorok közötti elemet.

Ezek a hozzárendelések a $v-hez kötött elem év attribútumának értéke szerint kerülnek rendezésre. Előfordulhat, hogy vannak azonos évszámmal rendelkező filmek, és a rendezés nem specifikálja, hogyan kell ezeket rendezni. Az eredményről annyit tudhatunk, hogy egy adott év film-verzió párjaiból fog állni ebben a sorrendben, és az évszámok csoportosítása az évszámok növekvő sorrendjében történik. Amennyiben szeretnénk specifikálni ezen hozzárendeléseknek egy totális rendezését, adjunk egy második tagot az order záradék listájához, például:

```
order $v/@év, $m/@filmcím
```

az egyezések megszüntetésére a címek ábécérendbe rendezésével.

A hozzárendelések rendezése után mindegyik hozzárendelés továbbmegy a return záradékra a választott sorrendben. A return záradékban történő változóhelyettesítéssel mindegyik hozzárendelésből előállítunk egyetlen `Film` elemet. □

12.2.11. Feladatok

12.2.1. feladat. A 12.4. és 12.5. ábrák termékadatainak használatával írjuk meg a következőket XQueryben.

a) Keressük meg azokat a `Nyomtató` elemeket, amelyeknek az ára 100 dollárnál kevesebb.

b) Keressük meg azokat a `Nyomtató` elemeket, amelyeknek az ára 100 dollárnál kevesebb, és állítsuk elő ezen elemek szekvenciáját úgy, hogy körülvesszük őket az `<OlcsóNyomtatók>` jelölővel.

! *c)* Keressük meg azoknak a gyártóknak a nevét, akik nyomtatókat és laptopokat is gyártanak.

! *d)* Keressük meg azoknak a gyártóknak a nevét, akik legalább két olyan PC-t gyártanak, amelyek sebessége 3.00 vagy annál nagyobb.

! *e)* Keressük meg azokat a gyártókat, akik csak olyan PC-ket gyártanak, amelyek ára nem több 1000 dollárnál.

!! *f)* Állítsuk elő elemek szekvenciáját a

`<Laptop><Modell>`x`</Modell><Gyártó>`y`</Gyártó></Laptop>`

formában, ahol x egy modellszám, y pedig a laptop gyártójának neve.

12.2.2. feladat. Használjuk a 12.6. ábra csatahajóadatait, és írjuk meg a következőket XQueryben.

a) Keressük meg azoknak a hajóosztályoknak a nevét, amelyek legalább 10 ágyúval rendelkeznek.

b) Keressük meg azoknak a hajóknak a nevét, amelyek legalább 10 ágyúval rendelkeznek.

c) Keressük meg az elsüllyesztett hajóknak a nevét.

d) Keressük meg azoknak a hajóosztályoknak a nevét, amelyek legalább 3 hajóval rendelkeznek.

! *e)* Keressük meg azoknak a hajóosztályoknak a nevét, amelyeknek nincs olyan hajója, amely csatában volt.

!! *f)* Keressük meg azoknak a hajóosztályoknak a nevét, amelyekben legalább két olyan hajó van, amelyeket ugyanabban az évben bocsátottak vízre.

!! *g)* Állítsuk elő adatok szekvenciáját a

$$\texttt{<Csata név = } x\texttt{><Hajó név = } y \texttt{ />}\cdots\texttt{</Csata>}$$

formában, ahol x egy csata neve, y pedig a csatában részt vett hajónak a neve. A szekvenciában egynél több `Hajó` elem is lehet.

! **12.2.3. feladat.** Oldjuk meg a 12.2.5. alfejezet problémáját. Írjunk egy lekérdezést, amely megkeres egy megadott lakcímen élő színészt (színészeket) – akkor is, ha több címmel is rendelkeznek –, kihagyva azokat a színészeket, akik nem a megadott címen laknak.

! **12.2.4. feladat.** Döntsük el, hogy létezik-e olyan E és F kifejezés, hogy az

$$\texttt{every \$x in } E \texttt{ satisfies } F$$

kifejezés igaz, de a

$$\texttt{some \$x in } E \texttt{ satisfies } F$$

hamis? Adjunk egy példát, vagy magyarázzuk meg, hogy ez miért lehetetlen.

12.3. XSLT

Az XSLT (Extensible Stylessheet Language for Transformations – Kiterjeszthető Stíluslap Nyelv) a World-Wide-Web Consortium egy szabványa. Az eredeti szándék annak biztosítása volt, hogy XML-dokumentumok HTML-be vagy más hasonló formába transzformálhatók legyenek annak érdekében, hogy a dokumentumok megtekinthetők vagy nyomtathatók legyenek. A gyakorlatban azonban az XSLT az XML egy másik lekérdező nyelve lett. Az XPath-hoz vagy az XQueryhez hasonlóan az XSLT-t is arra használjuk, hogy a dokumentumokból adatokat nyerjünk ki, illetve arra, hogy egy dokumentumot másik formába alakítsunk.

12.3.1. XSLT-alapismeretek

Az XML-sémához hasonlóan az XSLT specifikációja is XML-dokumentum. Ezeket a specifikációkat általában *stíluslapok*nak nevezzük. Az XSLT-ben használható jelölők a `http://www.w3.org/1999/XSL/Transform` névtérben találhatók. Így a legfelső szinten a stíluslap a 12.20. ábrához hasonlóan néz ki.

```
<? xml version = "1.0" encoding = "utf-8" ?>
<xsl:stylesheet xmlns:xsl =
        "http://www.w3.org/1999/XSL/Transform">
    ...
</xsl:stylesheet>
```

12.20. ábra. Egy XSLT-stíluslap alakja

12.3.2. Sablonok

A stíluslapok egy vagy több *sablon*nal rendelkeznek. Ahhoz, hogy egy stíluslapot egy XML-dokumentumra alkalmazzunk, lefelé kell haladnunk a sablonok listáján addig, amíg találunk egy olyat, amely illeszkedik a gyökérre. A feldolgozás során gyakran kell keresnünk illeszkedő sablonokat a dokumentumba ágyazott elemek számára. Mivel keresnünk kell a sablonok listájában az illeszkedési szabályoknak megfelelő illeszkedést, így ebben a részben ezt kell tanulmányoznunk. A sablon jelölőjének legegyszerűbb formája:

`<xsl:template match = "`*XPath-kifejezés*`">`

Az XPath-kifejezés – amely egyaránt lehet abszolút (egy / jellel kezdve) vagy relatív – az XML-dokumentumnak a sablon által feldolgozandó elemeit írja le. Ha a kifejezés abszolút, akkor a sablon a dokumentum minden olyan elemére alkalmazandó, amely illeszkedik a megadott útra. Relatív kifejezéseket akkor alkalmazunk, ha egy T sablon egy `<xsl:apply-templates>` jelölő belsejében van. Ebben az esetben az adott elem gyerekei között kell keresnünk olyat, amelyre a T alkalmazható. Ezzel a módszerrel keresztülhaladhatunk az XML-dokumentum fáján a lentről felfelé módszerrel, bonyolult transzformációkat végrehajtva a dokumentumban.

A sablon tartalma legegyszerűbb esetben egy szöveg lehet, tipikusan HTML. Amikor egy sablon illeszkedik egy dokumentumra, akkor a sablondokumentum belsejében levő szöveg áll elő kimenetként. A szöveg tartalmazhat hívásokat sablonalkalmazásokra a gyerekekre és/vagy dokumentumon belül elérhető értékekre vonatkozóan, például az aktuális elem attribútumaira.

12.20. példa. A 12.21. ábrán egy rendkívül egyszerű stíluslap található. Ez bármilyen dokumentumra alkalmazható, és ugyanazt a HTML-dokumentumot állítja elő a bemenettől függetlenül. Ez a HTML-dokumentum félkövér betűkkel az „Ez egy dokumentum" szöveget jeleníti meg.

A 4. sor bevezeti az egyetlen sablont a stíluslapba. A `match` attribútum értéke `"/"`, amely csak a gyökérre illeszkedik. A sablon törzse az 5. és 9. sorok között egy egyszerű HTML. Miután ezek a sorok kimenetként előállnak, az eredményül kapott fájl HTML-ként kezelhető és megjeleníthető egy böngészővel vagy más HTML-feldolgozóval. □

```
1)   <? xml version = "1.0" encoding = "utf-8" ?>
2)   <xsl:stylesheet xmlns:xsl =
3)           "http://www.w3.org/1999/XSL/Transform">
4)       <xsl:template match = "/">
5)           <HTML>
6)               <BODY>
7)                   <B>Ez egy dokumentum</b>
8)               </body>
9)           </html>
10)      </xsl:template>
11)  </xsl:stylesheet>
```

12.21. ábra. Nyomtatási kimenet tetszőleges dokumentumokhoz

```
<? xml version="1.0" encoding="utf-8" standalone="yes" ?>
<Filmek>
    <Film filmcím = "King Kong">
        <Verzió év = "1933">
            <Színész>Fay Wray</Színész>
        </Verzió>
        <Verzió év = "1976">
            <Színész>Jeff Bridges</Színész>
            <Színész>Jessica Lange</Színész>
        </Verzió>
        <Verzió év = "2005" />
    </Film>
    <Film filmcím = "Gumiláb">
        <Verzió év = "1984">
            <Színész>Kevin Bacon</Színész>
            <Színész>John Lithgow</Színész>
            <Színész>Sarah Jessica Parker</Színész>
        </Verzió>
    </Film>
        ... további filmek
</Filmek>
```

12.22. ábra. A `filmek.xml` fájl

12.3.3. XML-adatokból elérhető értékek

Szokatlan, hogy a kimenetként kapott dokumentum semmilyen módon nem függ a transzformáció bemenetétől, ahogy az a 12.20. példa esetében történt. A legegyszerűbb módja annak, hogy adatokat nyerjünk ki, a bemenetből a `value-of` jelölővel érhető el. Ennek a jelölőnek az alakja:

```
<xsl:value-of select = "kifejezés" />
```

A kifejezés egy olyan XPath-kifejezés, amely egy szöveget állít elő értékként. Egyéb értékek esetén – például amikor egy elem tartalmazza a szöveget –, nyilvánvaló módon ezt szöveggé kell alakítanunk.

12.21. példa. A 12.22. ábrán reprodukáltuk azt a `filmek.xml` fájlt, amelyet a 12.2. alfejezetben használtunk példaként. Ebben a stíluslapról szóló példában a `value-of` műveletet fogjuk használni arra, hogy elérjük az összes film címét, és kiírjuk azokat soronként egyesével. A stíluslapot a 12.23. ábra mutatja.

A 4. sorban azt látjuk, hogy a sablon minden `Film` elemre illeszkedik, így egyesével feldolgozhatjuk azokat. Az 5. sorban a `value-of` műveletet alkalmazzuk egy `@filmcím` XPath-kifejezéssel. Ez azt jelenti, hogy menjünk minden egyes `Film` elem `filmcím` attribútumához és szerezzük meg az értékét ennek az attribútumnak. Ez az érték kerül kimenetként előállításra, majd a következő 6. sorban egy HTML „új sor" jelölő található, így a következő filmcím a következő sorban kerül kiírásra. □

```
1)  <? xml version = "1.0" encoding = "utf-8" ?>
2)  <xsl:stylesheet xmlns:xsl =
3)          "http://www.w3.org/1999/XSL/Transform">
4)      <xsl:template match = "/Filmek/Film">
5)          <xsl:value-of select = "@filmcím" />
6)          <BR/>
7)      </xsl:template>
8)  </xsl:stylesheet>
```

12.23. ábra. A filmcímek kiírása

12.3.4. Sablonok rekurzív használata

A legérdekesebb és legerősebb transzformációkat várhatóan a sablonok rekurzív alkalmazása jelenti a bemenet különböző elemeire. Egy kiválasztott sablont alkalmazva a bemeneti dokumentum gyökerére kérhetjük, hogy alkalmazzunk egy sablont annak minden gyerekelemére is az `apply-templates` jelölő használatával. Amennyiben egy kiválasztott sablont csak a gyerekelemek egy részhalmazára – például azokra, amelyek egy kiválasztott jelölővel rendelkeznek – szeretnénk alkalmazni, használhatjuk a `select` kifejezést, mégpedig:

```
<xsl:apply-templates select = "kifejezés" />
```

Amikor egy ilyen jelölővel találkozunk a sablonban, megkeressük az aktuális elem illeszkedő gyerekelemeit (az elemeket, amelyekre a sablon alkalmazandó). Minden gyerekelem esetén megkeressünk az első illeszkedő sablont és alkalmazzuk azt a gyerekelemre.

```
1)  <? xml version = "1.0" encoding = "utf-8" ?>
2)  <xsl:stylesheet xmlns:xsl =
3)          "http://www.w3.org/1999/XSL/Transform">

4)      <xsl:template match = "/Filmek">
5)          <Filmek>
6)          <xsl:apply-templates />
7)          </Filmek>
8)      </xsl:template>

9)      <xsl:template match = "Film">
10)         <Film filmcím = "
11)         <xsl:value-of select = "@filmcím" />
12)         ">
13)         <xsl:apply-templates />
14)         </Film>
15)     </xsl:template>

16)     <xsl:template match = "Verzió">
17)         <xsl:apply-templates />
18)     </xsl:template>

19)     <xsl:template match = "Színész">
20)         <Színész név = "
21)         <xsl:value-of select = "." />
22)         " />
23)     </xsl:template>

24) </xsl:stylesheet>
```

12.24. ábra. A `filmek.xml` fájl transzformálása

12.22. példa. Ebben a példában az XSLT-t arra fogjuk használni, hogy XML-dokumentumot egy másik XML-dokumentumba transzformáljuk ahelyett, hogy egy HTML-dokumentumba transzformálnánk. Vizsgáljuk meg a 12.24. ábrát. Négy sablont találunk benne, és ezek együtt fogják feldolgozni a 12.22. ábra filmadatait. Az első sablon, a 4. és 8. sorok között a gyökérre illeszkedik. Ez azt mondja, hogy a kimenet legyen a `<Filmek>` szöveg, majd pedig alkalmazzuk a sablont a gyökérelem összes gyerekére. Specifikálhatnánk, hogy a sablont csak azokra a gyerekekre alkalmazzuk, amelyek a `<Film>`-mel vannak jelölve, és semmilyen más jelölőt nem várunk a gyerekek közül, ezért nem így specifikáltuk:

```
6)      <xsl:apply-templates select = "Film" />
```

Figyeljük meg, hogy <Film> gyerekekre vonatkozó sablonok alkalmazása után (ami az eredményben több elem kiírását eredményezi), a kimenetben lezártuk a <Filmek> elemet a megfelelő záró jelölővel a 7. sorban. Figyeljük meg azt is, hogy meg tudjuk különböztetni a kimenetben szereplő jelölőket – amilyenek az 5. és 7. sorokban vannak –, az XSLT-jelölőktől, mivel mindegyik XSLT-jelölőnek az xsl névtérből kell származnia.

Most nézzük meg, hogy a <Film> elemekre milyen sablonokat alkalmazunk. Az első (és egyetlen) sablon, amely ezekre az elemekre illeszkedik, a második sablon, amely a 9. és 15. sorok között található. Ez a sablon a kimenetet a <Film filmcím= " szöveggel kezdi a 10. sorban. Ez követően a 11. sor megadja a film címét és átadja azt a kimenet számára. A 12. sor befejezi az idézőjeles attribútumot és a <Film> jelölőt a kimenetben. A 13. sor alkalmazza a sablont a film összes gyerekére, amelyek a verziók lehetnek. Végezetül a 14. sor adja a </Film> záró jelölőt.

Amikor a 13. sor meghívja a sablont, hogy alkalmazza ezt a film összes verziójára, az egyetlen illeszkedő sablon a 16. és 18. sorok közötti, amely azon kívül nem tesz semmit azért, hogy a sablont alkalmazza a verzió összes gyerekére, amelyek a <Színész> elemek lehetnek. Így az, hogy mi kerül előállításra az egyes <Film> jelölők és a nekik megfelelő záró jelölő között, a 19. és 23. sorok közötti utolsó sablon határozza meg. Ez a sablon minden egyes <Színész> elemre vonatkozik.

A bemenet színészelemeit transzformáltuk a kimenetre. Ahelyett, hogy a színészek nevei szövegek lennének – ahogy a 12.22. ábrán –, a 19. sorban kezdődő sablon előállít egy <Színész> elemet a névvel, mint attribútummal. A 21. sor azt mondja, hogy válasszuk ki a <Színész> elemet magát (a pont jelentése „önmaga", ahogy az XPath-ban is), mint értéket a kimenet számára. Azonban a teljes kimenet szöveg, így az elemek jelölői nem részei a kimenetnek. Az eredmény pontosan az, amit akartunk, azaz a név attribútum értéke szöveg lesz és nem egy elem. Az üres <Színész> eleme a 22. sorban lesz teljes. Például, ha a bemenet a 12.22. ábrán adott, akkor a kimenetet a 12.25. ábra mutatja. □

12.3.5. Iterációk XSLT-ben

A sablonokba ciklusokat is elhelyezhetünk, amelyek szabadságot biztosítanak nekünk abban, hogy egy elem meglátogatott gyerekelemei közül melyikre alkalmazzuk a sablont. A for-each ciklus az alábbi alakú:

<xsl:for-each select = "*kifejezés*">

A kifejezés egy XPath-kifejezés, amelynek értéke tételek szekvenciája. Bármi is álljon a nyitó <for-each> jelölő és a neki megfelelő záró jelölő között, az mind végrehajtásra kerül ciklusléptenként az egyes tételekre.

12.23. példa. A 12.26. ábrán a színészek.xml dokumentumunk másolata látható. Szeretnénk ezt áttranszformálni egy olyan HTML-listába, amely az összes

```
<Filmek>
    <Film filmcím = "King Kong">
        <Színész név = "Fay Wray" />
        <Színész név = "Jeff Bridges" />
        <Színész név = "Jessica Lange" />
    </Film>
    <Film filmcím = "Gumiláb">
        <Színész név = "Kevin Bacon" />
        <Színész név = "John Lithgow" />
        <Színész név = "Sarah Jessica Parker" />
    </Film>
            ... további filmek
</Filmek>
```

12.25. ábra. A 12.24. ábra transzformációjának kimenete

színész nevét tartalmazza, és amelyet egy HTML-lista követ, amely az összes olyan várost tartalmazza, ahol színészek élnek. A 12.27. ábra az ezt a feladatot megoldó sablont tartalmazza.

Szerepel egy sablon, amely a gyökérre illeszkedik. Az első, ami az 5. sorban található `<OL>` HTML-jelölő hatására történik, hogy elkezdődik egy rendezett lista. Ezután a 6. sorban egy ciklus kezdődik az összes `<Színész>` gyerekelemre vonatkoztatva. A 7. és 9. sorok között egy listaelem kerül előállításra a színész nevével. A 11. sor lezárja a nevek listáját és elkezdi a városok listáját. A 12. és 16. sorok között található második ciklus végigfut az egyes `<Cím>` elemeken, és előállít egy-egy listaelemet az egyes városokhoz. A 17. sor zárja a második listát. □

12.3.6. Feltételes módok az XSLT-ben

A sablonjainkba elágazásokat vezethetünk be az `if` jelölő használatával. A jelölő alakja:

`<xsl:if test = "`*logikai kifejezés*`">`

Bármi is van ezen jelölő és a neki megfelelő záró jelölő között, az akkor és csak akkor kerül kiértékelésre, ha a logikai kifejezés igaz. Nincs else záradék, de ez után a kifejezés után, ha akarunk, el tudunk helyezni egy másik `if`-et ellentétes tesztfeltétellel.

12.24. példa. A 12.28. ábrán egy stíluslap látható, amely egy egyoszlopos táblát ír ki a „Színészek" fejléccel. Szerepel benne egy sablon, amely a gyökérre illeszkedik. A sablon által végrehajtott első tevékenység a fejléc sorának kiírása az 5. sorban. A 6. és 12. sorok között található for-each ciklus egy ciklus az összes színészre nézve. A 7. sorban található feltétel azt vizsgálja, hogy a

```
<? xml version="1.0" encoding="utf-8" standalone="yes" ?>
<Színészek>
    <Színész>
        <Név>Carrie Fisher</Név>
        <Cím>
            <Utca>123 Maple St.</Utca>
            <Város>Hollywood</Város>
        </Cím>
        <Cím>
            <Utca>5 Locust Ln.</Utca>
            <Város>Malibu</Város>
        </Cím>
    </Színész>
          ... további színészek
</Színészek>
```

12.26. ábra. A színészek.xml dokumentum

```
1)    <? xml version = "1.0" encoding = "utf-8" ?>
2)    <xsl:stylesheet xmlns:xsl =
3)            "http://www.w3.org/1999/XSL/Transform">
4)        <xsl:template match = "/">
5)            <OL>
6)            <xsl:for-each select = "Színészek/Színész" />
7)                <LI>
8)                <xsl:value-of select = "Név">
9)                </li>
10)           </xsl:for-each>
11)           </ol><P/><OL>
12)           <xsl:for-each select = "Színészek/Színész/Cím" />
13)               <LI>
14)               <xsl:value-of select = "Város">
15)               </li>
16)           </xsl:for-each>
17)           </ol>
18)       </xsl:template>
19)   </xsl:stylesheet>
```

12.27. ábra. A színészek neveinek és városainak kiírása

```
1)  <? xml version = "1.0" encoding = "utf-8" ?>
2)  <xsl:stylesheet xmlns:xsl =
3)          "http://www.w3.org/1999/XSL/Transform">
4)      <xsl:template match = "/">
5)          <TABLE border = "5"><TR><TH>Színészek</th></tr>
6)          <xsl:for-each select = "Színészek/Színész" />
7)              <xsl:if test = "Cím/Város = 'Hollywood'">
8)                  <TR><TD>
9)                  <xsl:value-of select = "Név" />
10)                 </td></tr>
11)             </xsl:if>
12)         </xsl:for-each>
13)         </table>
14)     </xsl:template>
15) </xsl:stylesheet>
```

12.28. ábra. A hollywoodi színésznevek keresése

színésznek van-e legalább egy lakása Hollywoodban. Emlékezzünk arra, hogy az egyenlőségjel egy olyan összehasonlítást reprezentál, amely akkor lesz igaz, ha van olyan adat a bal oldalon, amely a jobb oldalon is megtalálható. Ez az, amit szeretnénk, mivel mi azt kérdeztük, hogy van-e a színésznek otthona Hollywoodban. A 8. és 10. sorok közötti rész egy sort ír a táblázatba. □

12.3.7. Feladatok

12.3.1. feladat. Tételezzük fel, hogy bemenetként szolgáló XML-dokumentumunk a 12.4. és 12.5. ábra gyártási adatait tartalmazza. Írjunk XSLT-stíluslapokat, amelyek a következő dokumentumokat állítják elő.

a) Egy HTML-fájlt, amelynek fejléce „Gyártók" és egy felsorolási listát tartalmaz azoknak a gyártóknak a neveivel, akik a bemenetben szerepelnek.

b) Egy HTML-fájlt, amely egy táblázatot tartalmaz soronként az egyes PC-kkel, továbbá a „Modell" és „Ár" fejléccel. Egy sornak a PC-hez tartozó modellt és árat kell tartalmazni.

! *c)* Egy HTML-fájlt, amely egy táblázatot tartalmaz minden egyes laptopra a „Modell", „Ár", „Sebesség" és „Memória" fejléccel, majd ezt követően egy másik táblát ugyanazon fejléccel a PC-kre.

d) Egy XML-fájlt `<PC-k>` jelölőjű gyökérelemmel és `<PC>` jelölőjű gyerekelemekkel. Ennek a címkének az attribútumai legyenek `modell`, `ár`, `sebesség` és `memória`. A kimenetben egy-egy `<PC>` elemnek kell tartoznia a bemeneti fájl minden `<PC>` eleméhez, és az attribútumok értékeit a megfelelő bemeneti elemekből kell vennie.

!! *e)* Egy XML-fájlt `<Termékek>` jelölőjű gyökérelemmel és `<Termék>` jelölőjű gyerekelemekkel. Az egyes `<Termék>` elemek rendelkezzenek `típus`, `gyártó`, `modell` és `ár` attribútumokkal, ahol a gyártmány típusa `"PC"`, `"Laptop"` vagy `"Nyomtató"`. A bemeneti fájl minden egyes PC, laptop és nyomtató eleméhez egy `<Termék>` elemnek kell szerepelnie a kimeneten, és az attribútumok értékeit a bemeneti fájl megfelelő elemiből kell venni.

! *f)* Ismételjük meg a *b)* részt, de a kimenet egy LaTeX-fájl legyen.

12.3.2. feladat. Tegyük fel, hogy a bemenetként szolgáló XML-dokumentumunk a 12.6. ábra csatahajók adatait tartalmazza. Írjunk XSLT-stíluslapokat, amelyek a következő dokumentumokat állítják elő.

a) Egy HTML-fájlt, egy-egy fejléccel az egyes hajóosztályokhoz. Minden egyes fejléc alatt egy-egy táblázat szerepeljen „Név" és „Felavatva" oszlopfejlécekkel és az adott hajóosztályhoz tartozó hajók megfelelő adataival.

b) Egy XML-fájlt `<Vesztesek>` jelölőjű gyökérelemmel és `<Hajó>` jelölőjű gyerekelemekkel, amelyek értékei egyenként az elsüllyedt hajók neve.

! *c)* Egy XML-fájlt `<Hajók>` jelölőjű gyökérelemmel és az egyes hajókra `<Hajó>` jelölőjű gyerekelemekkel. Ezen elemek olyan `név`, `hajóosztály`, `ország` és `ágyúkSzáma` attribútumokkal rendelkezzenek, amelyek értékei a bemeneti fájl megfelelő értékeiből származnak.

! *d)* Ismételjük meg a *c)* részt, de csak azok a hajók szerepeljenek a listában, amelyek legalább egy csatában részt vettek.

e) Egy, a bementi fájlnak megfelelő XML-fájlt, de a `<Csata>` elemek legyenek üresek, és a csata kimenetele, illetve a csata neve két attribútumként szerepeljen.

12.4. Összefoglalás

- *XPath:* Ez a nyelv egy egyszerű módja az XML-adatokkal kapcsolatos lekérdezések kifejezésére. Utak írhatók le vele a jelölők szekvenciájával a dokumentum gyökerétől indulva. Az út egy attribútumban és egy elemben is végződhet.
- *Az XPath-adatmodell:* Minden XPath-érték tételek szekvenciája. Egy tétel vagy egy egyszerű érték vagy egy elem. Egy elemhez egy nyitó XML-jelölő, és a neki megfelelő záró jelölő tartozik, továbbá minden, ami e kettő között található.
- *Tengelyek:* Ahelyett, hogy lefelé haladnánk a fában, egy másik tengelyt is követhetünk, amely ugrásokat tartalmazhat valamely leszármazottra, szülőre vagy testvérre.

- *XPath-feltételek:* Az út bármely lépése olyan feltételhez köthető, amely egy logikai értékű kifejezés. Ez a kifejezés szögletes zárójelek között jelenik meg.
- *XQuery:* Ez a nyelv az XML-dokumentumok lekérdező nyelvének egy elterjedt formája. Az XPath-adatmodellt használja. Az XQuery egy funkcionális nyelv.
- *FLWR-kifejezések:* Az XQuery számos lekérdezése tartalmaz let, for, where és return záradékot. A „let" ideiglenes változók definícióit vezeti be, a „for" ciklusokat készít, a „where" teszteli a feltételeket, a „return" pedig definiálja a lekérdezés eredményét.
- *Összehasonlító operátorok XQueryben és XPath-ben:* A szokásos összehasonlító operátorok, amilyen a <, alkalmazható az adatok szekvenciájára, és „létezik olyan" jelentéssel bír. Ezek az összehasonlítások igazak, ha a megadott reláció fennáll a tételek bármely olyan párjára, amelyek az egyes listákból származnak. Annak biztosítására, hogy csak egy-egy tétel kerüljön összehasonlításra, a betűkkel jelölt operátorokat kell alkalmaznunk, amilyen az `lt` a „kisebb mint" jelentéssel.
- *Egyéb XQuery-kifejezések:* Az XQueryben számos olyan művelet van, amely hasonlít az SQL-ben lévőkre. Ezek az operátorok magukban foglalják az egzisztenciális és univerzális kvantorokat, az összesítéseket, a duplikátumok megszüntetését és az eredmény rendezését.
- *XSLT:* Ezt a nyelvet az XML-dokumentumok transzformációjára tervezték, ennek ellenére úgy is használható, mint egy lekérdező nyelv. Egy ezen a nyelven írt „program" alakja olyan, mint egy XML-dokumentumé speciális névtérrel, amely biztosítja számunkra a transzformációk leírására szolgáló jelölők használatát.
- *Sablonok:* Az XSLT szíve a sablon, amely a bemeneti dokumentum egyes elemeire illeszkedik. A sablon leírja a kimeneti szöveget, és képes arra, hogy értékeket kapjon a bementi dokumentumból a kimenetben való elhelyezés céljából. Egy sablon meghívhat újabb sablonokat az adott elem gyerekeire való rekurzív alkalmazással.
- *XSLT-programkonstrukciók:* Egy sablon tartalmazhat XSLT-szerkezeteket, amelyek úgy viselkednek, mint egy iteratív programozási nyelv. Ezek a szerkezetek a for ciklusok és az if-szerkezetek.

12.5. Irodalomjegyzék

A World-Wide-Web Consortium oldala az XPath definíciójára [2]. Az XQuery oldala [3], az XSLT oldala pedig [4].

[1] tartalmazza a bevezetést az XQuery nyelvbe. Oktatási oldalak az XPathhoz, az XQueryhez és az XSLT-hez [5]-ben találhatók.

[1] D. D. Chamberlin, „XQuery: an XML Query Language," *IBM Systems Journal* **41**:4 (2002), pp. 597–615. Lásd még:

`www.research.ibm.com/journal/sj/414/chamberlin.pdf`

[2] World-Wide-Web Consortium `http://www.w3.org/TR/xpath`

[3] World-Wide-Web Consortium `http://www.w3.org/TR/xquery`

[4] World-Wide-Web Consortium `http://www.w3.org/TR/xslt`

[5] W3 Schools, `http://www.w3schools.com`

Tárgymutató

F

G

Gy

H

Ny

O, Ó

Ö, Ő

P

Q

R

U, Ú

Ü, Ű

V

W

X

Z

A Stanford Egyetem Adatbázisrendszerek csoportja által készített és használt könyv világszerte az adatbázis-kezelés legnépszerűbb tankönyvei közé tartozik az informatikus-képzésben. Az adatbázisrendszereknek azt a két legfontosabb oldalát hangsúlyozza, amelyek a legtöbb informatikus hallgató számára a leghasznosabbak: az adatbázis-tervezést és az adatbázis-programozást. A könyv bármely felsőfokú informatikai képzéshez az adatbázisok témakörben az elvárható alapismereteket foglalja egységbe.

Az első kiadás megjelenése után rövid időn belül a hazai felsőoktatásban alaptankönyvvé vált. A felsőoktatás új, kétlépcsős rendszerében a három informatikus szak mindegyikén az adatbázis-kezelés tantárgyakhoz ezt a könyvet az oktatási intézmények szinte kivétel nélkül kötelező irodalomként adták meg.

A második kiadásban igen jelentős változtatást hajtottak végre a szerzők – a könyv legalább ötven százalékban újnak tekinthető. Felépítése az oktatás célszerűsége érdekében lényegesen változott. A kötet a bevezető tankönyvek minden követelményét kielégíti: a formális definíciók mellett didaktikusan tervezett példarendszer is az olvasó rendelkezésére áll.

PANEM
www.panem.hu
6900 Ft
ISBN 978-9-635454-81-5
9 789635 454815