Jeffrey D. Ullman – Jennifer Widom

Adatbázisrendszerek

Alapvetés

Jeffrey D. Ullman – Jennifer Widom

Adatbázisrendszerek

Alapvetés

Panem–Prentice-Hall

A mű eredeti címe: A First Course in Database Systems.
Authorized translation from the English language edition published by Prentice Hall, Inc. A Simon & Schuster Company. Upper Saddle River, New Jersey 07458.

Hungarian language edition published by Panem Kft.

Panem Kft.
1385 Budapest, Pf. 809
Hungary

Ez a könyv a Művelődési és Közoktatási Minisztérium támogatásával a Felsőoktatási Pályázatok Irodája által lebonyolított felsőoktatási tankönyv-támogatási program keretében jelent meg.

ISBN 963 545 190 3

Szerkesztette: dr. Benczúr András

Fordította: Cserges Enikő, Csizmazia Balázs, Gyenizse Pál,
Hajas Csilla, Kónya László, Kovács György, Nikovits Tibor
Lektorálta: dr. Kiss Attila és dr. Márkus Tibor
Műszaki szerkesztő: Érdi Júlia
Borítóterv: Érdi Júlia

A kiadásért felel a Panem Könyvkiadó Kft. ügyvezetője, Budapest, 1998

A Panem könyvek megrendelhetők a 06-30/9488-488 hívószámú telefonon,
illetve a 1385 Budapest, Pf. 809 levélcímen.
panem@mail.datanet.hu
http://www.datanet.hu/panem

Tartalomjegyzék

Előszó a magyar kiadáshoz

A magyar kiadás alig egy évvel követi a könyv amerikai megjelenését. Amikor a Művelődési és Közoktatási Minisztérium felsőoktatási tankönyvpályázatán fordításának támogatására pályáztunk, csupán sejtettük, a fordítás elkészítése és alapos átnézése után azonban megbizonyosodtunk arról, hogy jól választottunk a több hasonló tematikájú könyv közül.

Valóban olyan könyv jelenik meg ezzel a hazai informatikai, számítástechnikai szakirodalomban és a tankönyvek sorában, amely hiánypótló, és fontos alkalmazási terület számára jelent alaptankönyvet, amely a legfontosabb fogalmak és használati lehetőségek összefoglalását adja. Bemutatja egy jéghegy csúcsát, nevezetesen az adatbázis-kezelő rendszerekét.

Ez a jéghegy, mint minden hasonló, nagy alkalmazási szoftveregyüttes esetén, állandó változásban van, építkezik és kopik egyszerre, ahogy az új technológiák megjelennek és részben vagy egészében kiszorítják a régebbieket, vagy mélyebbre süllyesztik a felszín alá.

Amit a könyv bemutat, az a jelenleg kereskedelmi forgalomban beszerezhető tipikus, legjellemzőbb adatbázis-kezelő rendszerek, a relációs adatbázisok használati ismeretei, valamint a kialakult, illetve kialakulóban lévő – többek között az objektum-orientált – szabványok alapmegoldásai. A könyv szerzői az egyik legjelentősebb, a Stanford Egyetem Jeffrey D. Ullman által vezetett adatbázis-elméleti csoportjának teljes elméleti hátterére építhettek. A jéghegy-hasonlatot folytatva teljes mélységig és részletekig ismerik azt is, mi van a vízfelszín alatt, és mi van a jéghegy belsejében. Sőt, azt is tudják, hogyan épült és épül a jéghegy. Mindez jól visszatükröződik a könyvön, s annak ellenére, hogy a tárgyalás gyakorlatias jellegű, minden teljes tisztasággal került bemutatásra, az elméleti háttérre való utalások pedig jelzik, hogy a 30 éves fejlődés mögött mi húzódhat.

A könyv olyan, amilyennek egy bevezető tankönyvnek lennie kell. 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, a fogalmak elmélyítéséhez didaktikusan tervezett példarendszer is rendelkezésére áll. A könyv tárgyalásmódja mind a megértéshez, mind a megjegyzéshez elegendő szabadságot biztosít az olvasó számára.

Nyugodtan kijelenthető, hogy bármely felsőfokú informatikai, számítástechnikai képzés során a könyv tartalma az adatbázisok témakörben az elvárható minimális ismeretrendszert jelenti. Ezért a könyv a hazai felsőoktatásban főiskolai szinten az adatbázisok tantárgyhoz alapkönyvként használható. Tekintettel arra, hogy a tudományegyetemek programozó és programtervező matematikus szakán ezen a tananyagon túlmenő ismereteket csak azok kapnak, akik a programtervező szakon adatbázisokkal foglalkozó tárgykört választanak, a programozó matematikus szakon az adatbázisok tárgy keretében szintén ez a könyv ajánlható tankönyvként.

A magyar nyelvű változat gyors megjelenéséért köszönet illeti a fordítókat és lektorokat a rendkívül rövid határidő betartásáért és a jó minőségű fordításért. A fordítást végző csapat az Eötvös Loránd Tudományegyetem Információs Rendszerek Tanszék oktatóiból, jelenlegi és volt doktorandus hallgatóiból és egy-egy, a tárgyat oktató debreceni és szegedi kollégából alakult.

Dr. Benczúr András

Előszó

Ez a könyv a Stanford Egyetem „Bevezetés az adatbázisokba" című (CS145 azonosítójú) kurzus jegyzete alapján készült. Ez az első az öt szemeszteres sorozatból. A további négy rendre: adatbázisrendszerek alapjai, adatbázis megvalósítási projektkurzus, tranzakciók és osztott adatbázisok, adatbázis-elmélet. Oktatását Arthur Keller kezdte el, és az oktatás során a tananyag folyamatosan fejlődött, alakult. Jelenleg az adatbázisrendszerek azon két legfontosabb oldalára fekteti a hangsúlyt, amelyek az informatikus, számítástechnikus hallgatók többsége számára a leghasznosabb: az adatbázis-tervezésre és az adatbázis-programozásra. A tanulmányaik során a hallgatók egy projekt készítésében is részt vesznek, ahol adatbázist terveznek, majd alkalmazzák is azt a gyakorlatban. A projekt feladatkiírásai, egyéb házi feladatok, vizsgakérdések, kiegészítő anyagok hozzáférhetők a könyv honlapján (lásd a Támogatás a World Wide Weben című szakaszban).

A könyv használata

A könyv egyszemeszteres kurzushoz készült. Negyedéves kurzus számára, mint amilyen a CS145, ki kellett hagynunk, illetve le kellett rövidítenünk néhány részt. Természetesen az előadóra tartozik, mit hagyhat el leginkább, de a legnyilvánvalóbb kihagyási lehetőségek között a Datalog, az SQL programozás haladó témakörei és az SQL3 részletes tárgyalása említhető.

Amennyiben projektkészítés is része a kurzusnak, igen fontos, hogy az SQL utasítások előbb kerüljenek bevezetésre, mint ahogy a könyvben szerepelnek. Amit könnyen későbbre helyezhetünk, azok között a Datalog, az 5. és 6. fejezetekből az SQL3-mal foglalkozó szakaszok, valamint a 3. fejezet néhány elméleti része említendő. (Szüksége lesz a hallgatóknak azonban a normalizálásra, valószínűleg a többértékű függőségekre is ahhoz, hogy jó relációs tervet készíthessenek, mielőtt megkezdik az SQL programozást.)

Előismeretek

A könyvet olyan kurzusokon használtuk, amelyeket egyaránt felvehettek haladó „undergraduate" (az amerikai rendszerben az első felsőfokú képzési lépcső, a magyar főiskolai szintet közelíti) és kezdő „graduate" (második lépcső, egyetemi diplomával zárul) hallgatók. A tárgy felvételének formális követelménye a következő másodéves szintű 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 kereső fák, objektum-orientált programozás alapfogalmai és programozási környezetek. Hiszünk abban azonban, hogy az első év végére a számítástudományt hallgató diákok 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. Néhány feladatot, vagy annak egy részét, csillag jelöléssel is ellátunk. Arra törekszünk, hogy ezeknek a megoldásait hozzáférhetővé tegyük a könyv web-lapján. Ezek a megoldások nyilvánosan elérhetők, és segítenek az önellenőrzésben. Felhívjuk még a figyelmet arra, hogy vannak olyan esetek, amikor egy B példa igényli az olvasó által egy korábbi A feladatra adott válasz módosítását, vagy felhasználását. Ilyenkor, ha A valamely részére van megoldás, az olvasó biztos lehet abban, hogy B megfelelő részére is van.

Támogatás a Word Wide Weben

A könyv honlapjának címe:

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

Megtalálhatók itt a csillaggal jelölt feladatok megoldásai, a sajtóhibák, amint tudomást szerzünk róluk, és háttéranyagok. Reményeink szerint hozzáférhetővé tesszük minden meghirdetett CS145 kurzusunkhoz a kiosztott jegyzeteket, ahogy éppen tanítjuk, beleértve a házi feladatokat, a megoldásokat és a projektkiírásokat.

J. D. U.
J.W.

1. fejezet

Az adatbázisrendszerek világa

Ebben a könyvben az olvasó az adatbázis-kezelő rendszerek tényleges használatát fogja megismerni, beleértve az adatbázisok tervezését és az adatbázis-műveletek programozását is. Ez a fejezet arra szolgál, hogy bevezetést nyújtson számos olyan fogalomba, amelyek az adatbázisokhoz kapcsolódnak. A téma rövid történeti áttekintése után megtudhatjuk, hogy mi az, ami az adatbázisrendszereket más szoftverfajtáktól megkülönbözteti. Szintén e fejezet szolgál háttér-információkkal az adatbázisokat támogató adatbázis-kezelő rendszerek megvalósítását és használatát illetően is. A „színfalak mögött" zajló események megértése fontos ahhoz, hogy tisztán lássuk miért vannak úgy megtervezve az adatbázisok ahogyan vannak, és hogy miért vannak bizonyos korlátozások az adatbázisokon végrehajtható műveletekre vonatkozóan. Végül áttekintünk néhány fogalmat, mint például az objektumorientált programozás, amelyek valószínűleg ismerősek az olvasó számára, de amelyek nélkülözhetetlenek az elkövetkező fejezetekben.

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* (DBMS – Database Management System) kezel. Szokás ezeket a rendszereket egyszerűen csak *adatbázisrendszernek* nevezni. 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 nyelvnek* nevezzük.

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

3. Támogassa nagyon nagy mennyiségű adat (gigabájtok vagy még több adat) hosszú időn keresztül való tárolását, garantálja az adatok biztonságát a meghibásodásokkal és az illetéktelen felhasználókkal szemben, és tegye lehetővé a hatékony adathozzáférést a lekérdezések és az adatbázis-módosítások számára.

4. 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 lekérdezőnyelvet az adatokhoz. Az adatséma megadására vonatkozó lehetőségeik (1. pont) az állományokat tartalmazó könyvtárszerkezet megadására korlátozódnak. Végül pedig a 4. pont elvárásait sem elégítik ki. 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. Az alábbiakban felsorolunk néhány példát ezek közül:

Repülőgép-helyfoglalási rendszerek

Ebben az esetben az adatelemek a következők lehetnek:

1. Egy vevőnek egy járatra szóló helyfoglalása, amelybe beletartoznak olyan információk, mint az ülőhely sorszáma vagy a választott menü.

2. A járatokra vonatkozó információk, mint például az indulás és érkezés időpontja, hogy melyik reptérről indul és hova érkezik a járat, vagy hogy melyik repülőgép szállítja az utasokat.

3. A jegyárakra, az igényekre és a még kapható jegyekre vonatkozó információk.

A leggyakoribb lekérdezések, hogy egy adott városból egy másikba mely járatok indulnak egy hozzávetőlegesen megadott időpontban, mely ülőhelyek szabadok még, és mennyi a jegy ára. Jellegzetes adatmódosítások lehetnek például: egy vevő helyfoglalása egy járatra, egy ülőhely kiadása a vevőnek, vagy a menüválasztás bejegyzése. Egy adott időpontban sok különböző (jegyeladással foglalkozó) ügynökség érheti el az adatok egy-egy részét. Az adatbázis-kezelő rendszernek lehetővé kell tennie az ilyen konkurens hozzáféréseket, de meg kell akadályoznia az olyan problémákat, hogy két ügynök ugyanazt az ülőhelyet adja ki egyidejűleg. Meg kell továbbá akadályoznia a felvitt adatok elvesztését abban az esetben, ha a rendszer hirtelen valamilyen hiba folytán leállna.

Banki rendszerek

Itt az adatelemek lehetnek: az ügyfelek nevei és címei, a folyószámlák és hitelszámlák egyenlegei, vagy az ügyfelek és a számlák között fennálló kapcsolatok. Ilyen kapcsolat lehet például az, hogy kinek, melyik számla felett van aláírási joga. Gyakoriak a számlák egyenlegére vonatkozó lekérdezések, de még gyakrabbak az olyan módosítások, amelyek egy számlára vonatkozó befizetést vagy kifizetést jelentenek.

Ahogyan a repülőgép-helyfoglalási rendszernél, úgy itt is az várható, hogy egy időben több bankpénztáros és bankjegykiadó automatát használó ügyfél fogja az adatokat lekérdezni és módosítani. Alapvetően fontos, hogy az egyidejű hozzáférések miatt ne veszhessenek el egy automatánál végrehajtott művelet adatai. A hibák itt most nem megengedhetők. Ha például a pénzt már kiadta az automata, akkor a pénzfelvételt a banknak akkor is rögzítenie kell, ha közben hirtelen áramszünet lesz. Másrészről azonban a bank azt sem engedheti meg magának, hogy rögzítse a pénzfelvételt, de a pénzt nem adja ki az automata egy áramszünet miatt. Az ilyen műveletek helyes kezelése nem is olyan egyszerű. Az adatbázis-kezelő rendszerek egyik legfőbb erénye éppen az ilyen helyzetek megoldása.

Vállalati nyilvántartások

Az első alkalmazások közül jó néhánynak az volt a feladata, hogy egy vállalat különböző nyilvántartásait kezelje. Az adatok vonatkozhattak az eladásokra, a kimenő és bejövő számlákra, vagy éppen a dolgozókra. Ez utóbbi adatok lehetnek például a név, cím, fizetés, nyereségrészesedés, adóbesorolás stb. A lekérdezések olyan jelentések kinyomtatásából állnak, mint például a kinnlevő számlák, vagy a dolgozók havi fizetése. Minden egyes eladás, vásárlás, számlakibocsátás, számlakifizetés, vagy dolgozók felvétele, elbocsátása és előléptetése, az adatbázis módosítását jelenti.

A fájlkezelő rendszerekből kialakuló első adatbázis-kezelő rendszerek arra ösztökélték a felhasználót, hogy az adatokat olyan vizuális formában ábrázolja, ahogyan azok tárolva vannak. Ezek a rendszerek különböző adatmodelleket használtak az adatbázisban tárolt információk szerkezetének ábrázolásához. A legfontosabb ezen model-

lek 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] Mind a hierarchikus, mind a hálós modellt bemutatjuk a 2.7. szakaszban, bár ezek ma már csak történeti érdekességként említhetők.

Ezekkel a korai modellekkel és rendszerekkel a gond az volt, hogy nem támogattak semmilyen magas szintű lekérdezőnyelvet. Például a CODASYL lekérdezőnyelvnek 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. Relációs adatbázis-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 *relációknak*. 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. A 3. fejezetben ismertetjük a legalapvetőbb relációs fogalmakat. Az SQL nyelvet (Structured Query Language – Strukturált Lekérdezőnyelv), a relációs modell legfontosabb lekérdezőnyelvét az 5. fejezettől kezdődően tárgyaljuk. Egy rövid bevezetés a relációkba azonban már most érzékeltetheti az olvasóval a modell egyszerűségét, egy SQL példa pedig megmutatja, hogy a relációs modell hogyan támogatja a lekérdezések megírását egy magas szintű nyelven, amikor az adatbázisban való navigálás részleteivel nem kell törődnünk.

1.1. példa: A relációk mint táblák. Az oszlopok fejlécében vannak az *attribútumok*, amelyek az oszlopban szereplő értékek jellegét megadják. Például egy `Számlák` nevű reláció, amelyben bankszámlákat, azok egyenlegét és típusát tároljuk, a következőképpen nézhet ki:

[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.

Számlaszám	*Egyenleg*	*Típus*
12345	1000,00	betétszámla
67890	2846,92	folyószámla
...	...	...

Az oszlopok fejlécében szereplő három attribútum: `számlaszám`, `egyenleg`, `típus`. Az attribútumnevek alatt láthatók a reláció sorai. A példában két sort adtunk meg minden adatával, és az alattuk levő pontok azt jelzik, hogy a relációnak lehetnek további sorai. A bankban levő minden bankszámlának egy további sor felel meg. Az első sor azt az információt jelenti, hogy az 12345 számú számla típusa betétszámla és egyenlege 1000 dollár. A második sor azt mondja meg, hogy a 67890 számú számla egy folyószámla, amelynek egyenlege 2846,42 dollár.

Tegyük fel, hogy a 67890 számú számla egyenlegére vagyunk kíváncsiak. Ezt a lekérdezést SQL-ben a következőképpen fogalmazhatjuk meg:

```
SELECT egyenleg
FROM Számlák
WHERE számlaszám = 67890;
```

Egy másik példa lehet egy lekérdezésre a következő, amelyben azoknak a betétszámláknak a számlaszámát kérdezzük le, amelyeknek az egyenlege negatív:

```
SELECT számlaszám
FROM Számlák
WHERE típus = 'betétszámla' AND egyenleg < 0;
```

Természetesen nem gondoljuk, hogy ez a két példa elegendő lenne ahhoz, hogy az olvasót gyakorlott SQL programozóvá képezze, de a példák mindenesetre ízelítőt adhatnak abból, hogy mennyire magas szintű az SQL nyelv select-from-where utasítása. Ezek a példák alapjában véve a következőket kérik az adatbázisrendszertől:

1. Vizsgálja meg a `FROM` után szereplő reláció – jelen esetben a `Számlák` reláció – minden sorát.

2. Válassza ki azokat a sorokat, amelyek bizonyos követelményeknek megfelelnek, ahol a követelményeket a `WHERE` után adjuk meg.

3. Adja meg végeredményképpen ezeknek a soroknak bizonyos attribútumait, ahol a kívánt attribútumokat a `SELECT` után soroljuk fel.

A gyakorlatban a rendszernek optimalizálnia kell a lekérdezést, és egy hatékony végrehajtási módot kell találnia még akkor is, ha a lekérdezésben szereplő relációk nagyon nagyok lennének. □

Az IBM volt az első olyan cég, amelyik relációs, és a relációs modell előtti modelleket támogató adatbázis-kezelő rendszereket is árusított. Később újabb és újabb cégek alakultak, amelyek relációs adatbázis-kezelők megvalósításával és árusításával foglalkoztak. Ma már ezen cégek közül néhány a világ legnagyobb szoftverkereskedő cégei közé tartozik.

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 egy gigabájt ráfér egyetlen lemezre, és az is lehetséges, hogy egy adatbázis-kezelőt személyi számítógépen futtassunk. Így a relációs modellen alapuló adatbázisrendszerek egészen kis számítógépekre is megvásárolhatók ma már. Mára ezek a rendszerek ugyanolyan elterjedt számítógépes eszközökké kezdenek válni, mint korábban a táblázatkezelők és a szövegszerkesztők.

1.1.4. Egyre nagyobb rendszerek

Másik oldalról szemlélve a dolgot, egy gigabájtnyi adat nem is olyan sok. A vállalati adatbázisok gyakran több száz gigabájtnyi helyet foglalnak el, és ahogy a tárolókapacitás egyre olcsóbbá válik, az emberek újabb indokokat találnak ki a nagy adatmennyiségek tárolására. Például a kiskereskedelmi hálózatok gyakran terabájtnyi adatmennyiségeket (1 *terabájt* = 1000 gigabájt, vagyis 10^{12} bájt) tárolnak, megőrizve minden egyes eladás adatait hosszú időszakon keresztül. (Minderre a raktárkészlet tervezéséhez van szükség, erről bővebben szólunk majd az 1.3.4. szakaszban.) Az adatbázisok ma már nemcsak olyan egyszerű adatelemek tárolására alkalmasak, mint amilyenek az egész számok vagy a karakterláncok. Képek, audió, videó és sok más egyéb fajta adat is tárolható bennük, amelyek összehasonlíthatatlanul több helyet igényelnek. Egyórányi videofilm például egy gigabájt körüli helyet foglal el. A műholdak képeit tároló adatbázisok 2000-re várhatóan néhány petabájtnyi (1 *petabájt* = 1000 terabájt, vagyis 10^{15} bájt) információt fognak tárolni.

Az ilyen hatalmas méretű adatbázisok kezeléséhez néhány technológiai újításra is szükség volt. Manapság már a nem túl nagy méretű adatbázisokat is lemezkötegeken tárolják, amelyeket szokás másodlagos tárolóeszköznek is nevezni. (Az elnevezés a memóriával való összehasonlításból származik, amelyet elsődleges tárolónak nevezünk.) Sőt, tulajdonképpen elmondhatjuk, hogy ami a legjobban megkülönbözteti az adatbázisrendszereket más szoftverektől, az az a tény, hogy ezek eleve feltételezik, hogy az adatok nem férnek el a memóriában, és ezért lemezen kell őket tárolni a feldolgozás teljes ideje alatt. Az alábbi két irányvonal lehetővé teszi az adatbázisrendszerek számára, hogy egyre nagyobb mennyiségű adatot egyre gyorsabban kezeljenek.

Harmadlagos tárolás

A legnagyobb adatbázisokhoz manapság már nem elegendőek a lemezek. Mára kifejlesztettek néhány fajta úgynevezett *harmadlagos tárolóeszközt*, amelyek egyenként akár egy terabájtnyi adatot is képesek tárolni. Ezeken az eszközökön egy adattétel elérése sokkal több időt vesz igénybe, mint a lemezek esetén. Míg egy lemezen egy adatot elérhetünk 10–20 ezredmásodperc alatt, addig egy harmadlagos tárolón ugyanez akár néhány másodpercig is eltarthat. Egy harmadlagos tárolóeszköz magában foglalja az adatokat hordozó objektumokat, és az azokat az olvasóeszközig mozgató mechanikai részeket is. Ez a mechanikai mozgatás általában valamilyen robottechnikával valósul meg.

Egy ilyen eszközben az adathordozók lehetnek például CD-k, amelyeket egy mozgó állványzatra szerelt kar kezel. A kar elmegy egy CD-ért, kiveszi a helyéről, elviszi az olvasóegységig és betölti oda.

Párhuzamos számítás

A hatalmas adatmennyiségek tárolása nagyon fontos, de mit érnénk vele, ha nem tudnánk a nagy adatmennyiségeket gyorsan elérni. Ezért a nagy adatbázisok kezeléséhez sebességnövelő módszerekre is szükség van. Egyik ilyen módszer az indexstruktúrák használata, amiről majd az 1.2.1. és 5.7.7. szakaszokban szólunk még. Egy másik lehetőség arra, hogy több adatot dolgozzunk fel adott idő alatt, a párhuzamosság. A párhuzamos feldolgozás különféle módokon valósulhat meg.

Például, mivel az adatoknak lemezről való olvasása meglehetősen lassú – mindössze néhány megabájt másodpercenként – ezért a feldolgozást felgyorsíthatjuk, ha több lemezt használunk, és azokat párhuzamosan olvassuk. (Ha az adatok eredetileg harmadlagos tárolón helyezkednek is el, azok mindenképpen lemezre olvasódnak, mielőtt az adatbázis-kezelő elérné őket.) Maguk a lemezek lehetnek akár egy párhuzamos gép részei, akár egy elosztott rendszer komponensei, amelyben az egyes gépek az adatbázis egyes részeit kezelik, és szükség esetén egy nagy sebességű hálózaton kommunikálnak egymással.

Természetesen önmagában sem az adatok gyors elérhetősége sem a nagy adatmennyiség tárolási lehetősége nem garantálja azt, hogy a lekérdezésekre gyors választ kapjunk. Szükségünk van még olyan algoritmusok használatára is, amelyek úgy darabolják szét a lekérdezéseket, hogy a párhuzamos számítógépek vagy a hálózatba kötött számítógépek hatékonyan tudják felhasználni a rendelkezésükre álló erőforrásokat. Ezért a nagy adatbázisok párhuzamos és elosztott feldolgozása továbbra is nyitott kutatási és fejlesztési terület marad.

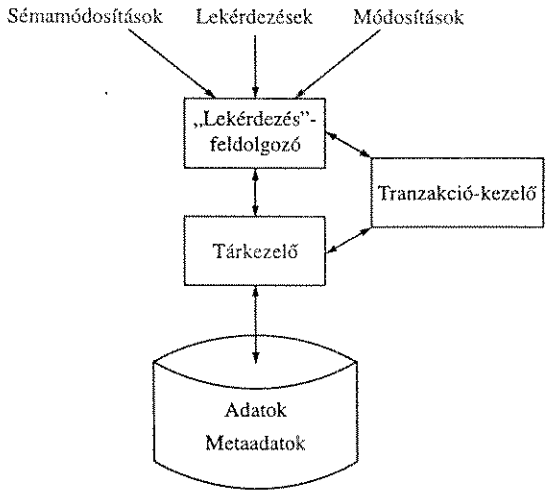
1.2. Adatbázis-kezelő rendszerek felépítése

Ebben a szakaszban felvázoljuk, hogy általában hogyan néz ki egy adatbázis-kezelő rendszer felépítése. Azt is megvizsgáljuk, hogy milyen lépéseket hajt végre a rendszer egy lekérdezés vagy más adatbázis-művelet végrehajtása során. Végül megemlítünk néhány problémát, amelyek nagy mennyiségű adatokat feldolgozó, és nagyon sok lekérdezésre választ adó adatbázisrendszer tervezésekor merülnek fel. Az adatbázis-kezelő rendszerek megvalósításának technológiája nem témája e könyvnek, mindössze arra fordítunk figyelmet, hogy hogyan lehet az adatbázisokat megtervezni és hatékonyan használni.

1.2.1. Adatbázis-kezelő rendszerek főbb részei

Az 1.1. ábrán az adatbázis-kezelő rendszerek legfontosabb részeit láthatjuk. Az ábra alján az adatok tárolására szolgáló helyet ábrázoltuk. Szokás szerint az adattárolásra szolgáló részt lemez alakú ábra jelöli. Megjegyezzük, hogy az ábrán azt is jelöltük, hogy ez a rész nem csak adatokat, hanem úgynevezett *metaadatokat* is tartalmaz, ami az adatok szerkezetét írja le. Például ha az adatbázis-kezelő relációs, akkor a metaadatok között szerepelnek a relációk nevei, a relációk attribútumainak nevei és az attribútumok adattípusai (mint például egész vagy 20 hosszúságú karakterlánc).

Az adatbázis-kezelő rendszerek gyakran indexeket használnak az adatok elérésére. Az index olyan adatstruktúra, amely lehetővé teszi, hogy az adatelemeket gyorsan megtaláljuk, ha ismerjük a hozzájuk tartozó értékek bizonyos részét. A leggyakrabban



1.1. ábra. *Egy adatbázis-kezelő főbb részei*

Hogyan valósítják meg az indexeket?

Az olvasó korábban talán már megtanulta valamilyen adatstruktúrákkal foglalkozó kurzuson, hogy az indexek készítésénél a hash tábla nagyon hatékony eszköz. A korábbi adatbázis-kezelők széles körben használták is azokat. Manapság a leggyakrabban alkalmazott adatstruktúra a *B-fa,* ahol a B betű a kiegyensúlyozottságra utal.* (Balanced = kiegyensúlyozott) A B-fa a kiegyensúlyozott bináris keresőfa általánosítása. Míg azonban a bináris fa csomópontjainak legfeljebb két gyermeke lehet, addig a B-fa csomópontjainak sok gyermeke. A B-fák általában a lemezen vannak és nem a memóriában, ezért azokat úgy tervezik, hogy egy csomópont egy teljes lemezblokkot elfoglaljon. Általában a legtöbb rendszer 4096 bájt (2^{12} bájt) méretű blokkot használ, ezért a B-fa egy csomópontjában több száz gyermekekre mutató pointer is lehet. Így a B-fában való keresés ritkán mélyebb három szintnél.

A lemezműveletek tényleges költsége általában arányos az elért lemezblokkok számával. Ezért a B-fában való keresés, amelyik legtöbbször csak három lemezblokk elérését igényli, sokkal hatékonyabb, mint amilyen a bináris fában való keresés lenne, hiszen ott sok különböző lemezblokkon elhelyezkedő csomópontot kellene bejárnunk. A B-fák és a bináris keresőfák között meglévő imént említett különbözőség csak egy példa a sok közül arra, hogy a lemezen tárolt adatok számára legmegfelelőbb adatstruktúrák és a memóriában futó algoritmusok által használt adatstruktúrák mennyire különbözőek.

előforduló példa egy olyan index, amelyiknek a segítségével egy reláció azon sorait kereshetjük meg, amelyekben az egyik attribútum értéke adott. Ha például egy relációban a számlaszámokat és az egyenlegeket tároljuk, és a relációnak van indexe a számlaszámra vonatkozóan, akkor az adott számlaszám egyenlegét gyorsan megtaláljuk ennek az indexnek a segítségével. Az indexek a tárolt adatok közé tartoznak, az az információ azonban, hogy mely attribútumokra léteznek indexek, a metaadatok részét képezi.

Az 1.1. ábrán láthatjuk még a *tárkezelőt,* amelynek feladata a kért információk beolvasása a tárolóhelyről, illetve az adatok módosítása, amennyiben a felette levő rendszerszintek ilyen irányú kérést fogalmaznak meg. Láthatunk még egy olyan komponenst, amelyet *lekérdezésfeldolgozónak* neveztünk, habár ez az elnevezés nem teljesen helyes. Ez a rész ugyanis nemcsak a lekérdezéseket kezeli, hanem az adatok és a metaadatok módosítására vonatkozó kéréseket is ez adja ki. Ennek a komponensnek kell megtalálnia egy művelet legjobb végrehajtási módját, és ez adja ki a tárkezelőnek szóló utasításokat is.

---

* Szerkesztői megjegyzés: A B-fa elnevezés eredete homályos. Bevezetésüket 1972-ben Bayer és McCreight javasolta, de nem adták meg, honnan származik az elnevezés. Lásd: Algoritmusok. Műszaki Könyvkiadó, 1997. A B-fák hatékonysága nem csak a sok gyereken múlik, hanem telítetlenségük miatt lehetséges gyors módosíthatóságukon.

A *tranzakció-kezelő* rész felelős a rendszer sértetlenségéért. Neki kell biztosítania, hogy az egy időben futó lekérdezések és módosítások ne ütközzenek össze egymással, és hogy a rendszer még rendszerhiba esetén se veszíthessen adatokat. Kapcsolatot tart a lekérdezésfeldolgozóval, hiszen a konfliktusok elkerüléséhez tudnia kell, hogy az aktuális lekérdezések éppen mely adatokon dolgoznak, és épp e konfliktusok megakadályozásáért késleltethet bizonyos lekérdezéseket vagy műveleteket. Kapcsolatot tart a tárkezelővel is, mert az adatok védelmére szolgáló módszerek többnyire magukban foglalják a módosítások naplózását is. Ha a műveleteket a megfelelő sorrendben végzi el a rendszer, akkor a naplóban benne lesznek a változtatások, és így egy rendszerhiba után még azok a módosítások is újra végrehajthatók lesznek, amelyeknek a lemezre írása eredetileg nem volt sikeres.

Az 1.1. ábra tetején az adatbázis-kezelő rendszer három különböző fajta inputját láthatjuk:

1. *Lekérdezések.* Ezek az adatokra vonatkozó kérdések, amelyek két különböző módon jöhetnek létre.

   a) Egy *általános lekérdezőinterfészen* keresztül. Például a rendszer megengedi a felhasználónak, hogy SQL lekérdezéseket gépeljen be, amelyeket aztán a lekérdezésfeldolgozó fog megkapni és végrehajtani.

   b) Egy *alkalmazói program interfészén* keresztül. Az adatbázis-kezelő rendszerek általában megengedik a programozónak, hogy olyan programot írjon, amelyik az adatbázis-kezelőnek szóló hívásokon keresztül kérdezi le az adatbázist. Például egy repülőgép-helyfoglalási rendszert használó ügynök egy olyan programot futtat, amelyik a különböző járatok adatait kérdezi le az adatbázisból. A lekérdezések egy speciális interfészen keresztül adhatók meg, ahol tulajdonképpen mezőket kell kitölteni városok neveivel, időpontokkal vagy egyéb adatokkal. Ezen az interfészen keresztül nem tehetünk fel tetszőleges kérdéseket, de amit lekérdezhetünk azt általában egyszerűbben kérdezhetjük le ezen az interfészen keresztül, mintha nekünk kellene megírnunk a lekérdezést SQL-ben.

2. *Módosítások.* Ezek a műveletek az adatok módosítására szolgálnak. Ugyanúgy, ahogyan a lekérdezéseket, ezeket is kiadhatjuk egy általános interfészen, vagy egy alkalmazás interfészén keresztül.

3. *Sémamódosítások.* Ezeket az utasításokat általában csak az arra illetékes személyek adhatják ki – *adatbázis-adminisztrátoroknak* hívjuk őket –, akik megváltoztathatják az adatbázis sémáját, vagy akár új adatbázist hozhatnak létre. Például, amikor a bankokat arra kötelezték, hogy a kamatkifizetéseket az ügyfelek személyi számával együtt jelentsék az adóhatóság felé, akkor minden bizonnyal egy új attribútummal, a személyi számmal, kellett bővíteniük azt a relációt, amelyik az ügyfelek adatait tárolta.

1.2.2. A tárkezelő

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 két részből áll, a pufferkezelőből és a fájlkezelőből.

1. A *fájlkezelő* tartja nyilván a fájlok lemezen való elhelyezkedését és olvassa be egy állomány blokkjait, ha a pufferkezelő erre kéri. Emlékeztetünk rá, hogy a lemezek általában *lemezblokkokra* vannak osztva, amelyek nagyméretű összefüggő tárolóterületek, melynek mérete akár 4000 vagy 16 000 bájt is lehet. (2^{12}, illetve 2^{14} bájt)

2. A *pufferkezelő* a memóriát kezeli. Az adatblokkokat a lemezről a fájlkezelő segítségével olvassa be, és kiválaszt egy memóriaterületet, ahol az adott blokkokat tárolni fogja. Egy lemezblokkot a memóriában tarthat egy ideig, de ha a blokk által foglalt memóriára egy másik blokknak van szüksége, akkor a blokkot visszaírja a lemezre. A blokkokat akkor is lemezre írja, ha a tranzakció-kezelő kéri erre. (Lásd az 1.2.4. szakaszt)

1.2.3. A lekérdezésfeldolgozó

A lekérdezésfeldolgozó feladata, hogy a lekérdezéseket, illetve egyéb adatbázis-műveleteket, amelyek gyakran egy magas szintű nyelven vannak megfogalmazva (mint pl. egy SQL lekérdezés esetén) egyszerű utasítások sorozatává alakítsa. Ezek az egyszerű utasítások a relációknak vagy az indexeknek bizonyos soraira vonatkozó kérések. Gyakran a lekérdezésfeldolgozás legnehezebb része a *lekérdezések optimalizálása,* vagyis egy megfelelő *végrehajtási terv* kiválasztása. A végrehajtási terv a tárolórendszernek szóló olyan kérések sorozata, amelyek választ adnak a lekérdezésre.

1.2. példa: Tegyük fel, hogy egy bank adatbázisában a következő két reláció található:

1. Az `Ügyfelek` tábla tárolja az egyes ügyfelek nevét, címét és személyi számát.

2. A `Számlák` tábla tárolja az egyes számlák számlaszámát, egyenlegét és a számla tulajdonosának személyi számát. Látható, hogy minden számlának van egy elsődleges tulajdonosa, akinek a személyi számát a bank az adóhatóság felé történő bevallásokban szerepelteti. Emellett egy számlának más tulajdonosai is lehetnek, de a fenti két relációból ezt nem tudjuk megállapítani.

Tegyük fel, hogy a következő lekérdezésre szeretnénk választ kapni: Adjuk meg az összes számla egyenlegét, amelynek elsődleges tulajdonosa Sally Jones. A lekérdezésfeldolgozónak ekkor egy olyan végrehajtási tervet kell találnia a két relációra vonatkozóan, amelyik választ ad a kérdésre. Minél kevesebb lépésből áll egy végrehajtási

terv annál jobb. Általában azok a költséges műveletek, amikor a tárkezelő egy lemezblokkot másol a lemezről a memóriában levő pufferbe, vagy amikor a pufferből egy memóriablokkot lemezre ír. Ezért célszerű egy végrehajtási terv költségeinek becslésénél csak az ilyen lemezblokk műveleteket számolni.

A lekérdezés megválaszolásához először meg kell keresnünk az `Ügyfelek` táblában Sally Jones személyi számát. (Feltesszük, hogy csak egy ilyen nevű ügyfél van, bár a gyakorlatban akár több is lehetne.) Ezután a `Számlák` táblában meg kell keresnünk minden olyan sort, ahol ez a személyi szám szerepel, és ezeknek a számláknak az egyenlegét kell kinyomtatnunk.

Egy egyszerű, de meglehetősen költséges terv az lenne, hogy végignézzük az `Ügyfelek` tábla minden sorát egészen addig, amíg nem találunk egy olyan sort, ahol az ügyfél neve Sally Jones. Ekkor átlagosan a sorok felét kell megvizsgálnunk, hogy megtaláljuk a keresett sort. Mivel egy bankban általában elég sok ügyfél van, az `Ügyfelek` tábla sok lemezblokkot foglal el és így ez a lépés elég költséges lehet. Ha megtaláltuk Sally Jones személyi számát még nem vagyunk készen. Még meg kell találnunk a `Számlák` tábla azon sorait, amelyek ezt a személyi számot tartalmazzák. Mivel több ilyen sor is lehet, ezért az összes sort végig kell néznünk. A bankokban nagyon sok számla van, így a `Számlák` tábla is sok lemezblokkot foglalhat el, amelyek végigolvasása szintén nagyon költséges.

Ha az `Ügyfelek` táblához van indexünk az ügyfél nevére, akkor jobb végrehajtási tervet is készíthetünk. A teljes `Ügyfelek` tábla végignézése helyett használhatjuk az indexet, hogy megtaláljuk azt az egyetlen lemezblokkot, amelyen a Sally Jonesra vonatkozó sor van. Ahogy korábban az 1.2.1. szakasz bekeretezett részében említettük, egy B-fa indexben általában három blokk olvasásával megtalálhatjuk, amit keresünk.[3] Még egy további blokk olvasással megkapjuk a Sally Jonesra vonatkozó sort.

Persze még ekkor is hátra van a második lépés: meg kell találnunk a `Számlák` tábla összes olyan sorát, amelyben az adott személyi szám szerepel. Ez a lépés általában megint csak sok lemezműveletet igényel. Ha azonban a `Számlák` táblának van indexe a személyi számra, akkor az adott személyi számhoz tartozó sorokat tartalmazó blokkokat megtalálhatjuk az index segítségével is. Az indexben való kereséshez két vagy három lemezolvasási műveletre van szükség, ahogy már korábban az `Ügyfelek` táblánál említettük. Ha a keresett sorok mind különböző lemezblokkon helyezkednek el, akkor az összes ilyen blokkot be kell olvasnunk. Egy ügyfélnek általában nincs túl sok számlája, ezért ez a lépés valószínűleg csak néhány lemezolvasási műveletet igényel. Ha a fenti két index létezik, akkor a lekérdezést megválaszolhatjuk körülbelül 6–10 lemezolvasási művelettel. Ha egyik vagy mindkét index hiányzik és ezért az egyszerűbb végrehajtási tervet kell alkalmaznunk, akkor a lemezblokkolvasások száma elérheti akár a több százezret is, mivel egy nagyméretű teljes táblát végig kell néznünk. □

[3] Valójában a B-fa gyökerét tartalmazó blokkot a rendszer általában a memóriában, valamelyik blokkpufferben tartja, mivel arra mindig szükség van, ha az indexben keresünk. Ezért tulajdonképpen két blokkolvasás elegendő lesz.

Az 1.2. példából úgy tűnhet, hogy egy lekérdezés optimalizálása mindössze annyiból áll, hogy használjunk indexeket, ha léteznek. Valójában a dolog ennél sokkal bonyolultabb. A bonyolult lekérdezések sokszor lehetővé teszik, hogy átrendezzük a műveleteket, és így nagyon sok lehetséges végrehajtási terv adódik. Ezek száma exponenciálisan nagy lehet a lekérdezés méretének függvényében. Néha két index közül is választhatunk, hogy melyiket használjuk, de mindkettőt egyszerre nem használhatjuk. Az adatbázis-kezelő rendszerek megvalósításának fontos része ez a téma, ennek tanulmányozása azonban túlmutat e könyv keretein.

1.2.4. A tranzakció-kezelő

Amint azt az 1.1. szakaszban említettük, van néhány fontos szempont, amit az adatbázis-kezelőnek garantálnia kell azok számára, akik az adatbázison műveleteket hajtanak végre. Egyik példaként szerepelt annak a jelentősége, hogy egy művelet hatása akkor sem tűnhet el, ha valamilyen komoly rendszerhiba következne be. Az adatbázis-kezelők többnyire megengedik a felhasználóknak, hogy egy vagy több lekérdezést vagy módosítást egy *tranzakcióba* csoportosítsanak. A tranzakció tulajdonképpen olyan műveletek egy csoportja, amelyeket egymás után egy egységként kell végrehajtani.

Az adatbázisrendszerek gyakran sok tranzakció egyidejű végrehajtását engedik meg. Például egy bank bankjegykiadó automatáinál egyszerre különböző dolgok történhetnek. Annak biztosítása, hogy mindezek a tranzakciók helyesen fussanak le, a *tranzakció-kezelő* feladata. Hogy pontosan mit is értünk a tranzakciók „helyes" lefutásán, azt a következő négy alapvető elvárás megfogalmazásával részletezzük.

- *Atomosság*. Megköveteljük, hogy a tranzakció vagy teljes egészében hajtódjon végre, vagy semmi ne hajtódjon végre belőle. Például az automatából történő pénzfelvétel és a hozzá kapcsolódó megterhelés az ügyfél számláján egyetlen atomi tranzakciót kell hogy alkosson. Nem megengedhető, hogy a pénzkiadás megtörténjen, de a megterhelés ne legyen regisztrálva, vagy fordítva, hogy a megterhelés megtörténjen, de a pénzt az automata ne adja ki.

- *Következetesség*. Egy adatbázisban általában beszélhetünk a „következetes állapotok" fogalmáról, amelyekben az adatok megfelelnek bizonyos elvárásoknak. Például egy repülőgép-helyfoglalási adatbázisban egy megfelelő következetességi feltétel lehet az, hogy egyetlen ülőhelyet se rendeljünk hozzá két különböző utashoz. Noha ezt a feltételt megsérthetjük egy rövid időre egy tranzakció alatt (pl. amíg az utasokat áthelyezzük az ülőhelyek között), a tranzakció-kezelőnek kell biztosítania, hogy a tranzakciók befejeződése után az adatbázis ismét következetes állapotba kerüljön, vagyis elégítse ki az összes következetességi feltételt.

- *Elkülönítés*. Amikor két vagy több tranzakció egyidejűleg fut, azok kihatását el kell különíteni egymástól. Ez azt jelenti, hogy semmiféle olyan eredményt vagy kiha-

tást nem tapasztalhatunk az adatbázisban, amit a két tranzakció egyidejű futása okozott, és ami nem fordult volna elő, ha a két tranzakció egymás után fut le. Ha például két ügynök éppen ugyanarra a járatra ad el jegyet és a járaton már csak egy hely van, akkor az egyik kérést teljesíteni kell, és a másikat vissza kell utasítani. Sem az nem elfogadható, hogy a jegyet kétszer adjuk el, sem az, hogy egyszer sem, az egyidejű műveletek miatt.

- *Tartósság*. Ha egy tranzakció befejezte a munkáját, akkor annak eredménye nem veszhet el rendszerhiba esetén sem, még akkor sem, ha a rendszer közvetlenül a tranzakció befejezése után hibásodik meg.

Arról, hogy a tranzakciókat hogyan valósítják meg, hogy a fenti négy feltételnek megfeleljenek, egy külön könyvet lehetne írni, és ezért ezt a témát itt most nem is próbáljuk meg bemutatni. Azt azonban, hogy az SQL nyelvben hogyan adható meg, hogy mely műveletek tartoznak egy tranzakcióba, és hogy az SQL-programozó milyen garanciákra számíthat a tranzakciókkal kapcsolatban, azt a 7.2. szakaszban fogjuk tárgyalni. Továbbá még ebben a szakaszban vázoljuk majd röviden azokat a technikákat, amelyek a fenti négy elvárás érvényre juttatásában segítenek.

Zárolás

Ha a tranzakciók kihatása nem különül el egymástól annak az alapvető oka az, hogy két vagy több tranzakció írja vagy olvassa ugyanazt az adattételt. Ha például két tranzakció próbálja meg egyidejűleg módosítani ugyanannak a számlának az egyenlegét, akkor az egyik felül fogja írni a másikat és így az első írás eredménye el fog veszni. Ezért a legtöbb adatbázis-kezelőben a tranzakció-kezelő zárolhatja a tranzakció által elérni kívánt adattételt. Amíg egy tranzakció zárolva tart egy tételt, addig a többi tranzakció nem érheti el azt. Így az előző példa esetén az első tranzakció, amelyik zárolja az 12345 számú számla egyenlegét, az olvashatja és írhatja felül ezt az egyenleget. Egy másik tranzakció csak ezután érheti azt el, és így már az új egyenleget fogja olvasni. Ennek eredményeképpen a két tranzakció nem lesz hatással egymásra.

A zárolások finomsága

A különböző adatbázis-kezelők eltérnek abban, hogy milyen fajta adattételek zárolását teszik lehetővé. Van amelyik sorok, van amelyik lemezblokkok, és van amelyik teljes relációk zárolását megengedi. Minél nagyobb a zárolt objektum, annál nagyobb az esélye annak, hogy egy tranzakciónak egy másikra kell várnia, még akkor is, ha a két tranzakció ténylegesen nem ugyanazt az adatelemet szeretné elérni. Viszont minél kisebb a zárolható adattétel, annál terjedelmesebb és bonyolultabb a zárolást kezelő mechanizmus.

Naplózás

A tranzakció-kezelő az összes megkezdett tranzakciót, a tranzakciók által az adatbázisban végzett módosításokat, és a tranzakciók végét feljegyzi egy naplóba. A napló mindig olyan tárolóeszközre íródik, ami nem érzékeny az esetleges áramkimaradásra. Ilyenek például a lemezek. Ezért noha a tranzakció a munkájának egy részéhez a sokkal érzékenyebb memóriát használja, a napló minden esetben azonnal lemezre íródik. Az összes művelet naplózása fontos szerepet játszik a tartósság biztosításában.

Tranzakciók érvényesítése

A tartósság és az atomosság érdekében a tranzakciók egyfajta „puhatolózás" jelleggel kerülnek végrehajtásra, ami azt jelenti, hogy az adatbázisbeli módosítások kiszámításra kerülnek, de ténylegesen az adatbázisban még nem történnek meg. Amikor a tranzakció készen áll a befejezésre, arra, hogy *érvényesítse* az elvégzett munkát, a módosítások a naplóba kerülnek. Először ezek a naplóbejegyzések íródnak lemezre, és csak ezután történik meg az adatbázis tényleges módosítása.

Így még ha a rendszer a két lépés közben omlik is össze, akkor is látható lesz majd a naplóból, amikor a rendszer ismét helyreáll, hogy a változtatásokat még el kell végeznünk az adatbázisban. Ha a rendszer még azelőtt omlik össze, hogy az összes módosítást a naplóba bejegyeztük volna, akkor a tranzakciót nyugodtan visszagörgethetjük. Ezzel tudjuk biztosítani, hogy még véletlenül se adhassuk el kétszer ugyanazt a jegyet, vagy ne terheljük meg kétszer tévedésből ugyanazt a bankszámlát.

1.2.5. Kliens-szerver felépítés

Manapság nagyon sok különböző szoftverfajta használja a *kliens-szerver* felépítést, ami azt jelenti, hogy az egyik folyamat (a *kliens*) kéréseit egy másik folyamat (a *szerver*) hajtja végre. Ez alól a modell alól az adatbázis-kezelők sem jelentenek kivételt, és így az 1.1. ábra komponenseit szokás egy szerver folyamatra és egy vagy több kliens folyamatra osztani.

A legegyszerűbb kliens-szerver felépítésben az egész adatbázis-kezelő mindössze egyetlen szerver, eltekintve a lekérdező interfészektől, amelyek a felhasználóval kommunikálnak, és a lekérdezéseket, illetve az egyéb utasításokat eljuttatják a szerverhez. A relációs rendszerek például az SQL nyelvet használják a klienstől a szervernek szóló kérések megfogalmazására. Az adatbázisszerver a választ egy tábla formájában küldi vissza a kliensnek. A kliens és a szerver közti kapcsolat jóval bonyolultabb is lehet, különösen amikor a válaszok nagyon nagyok. Erről majd az 1.3.3. szakaszban részletesebben is szólunk. Manapság van egy olyan törekvés is, hogy minél több munkát a kliens oldalon végezzünk el, hiszen sok egyidejű felhasználó esetén a szerver túlterheltsége lehet az egész rendszer szűk keresztmetszete.

1.3. Adatbázisrendszerek jövője

Napjainkban számos különböző irányzat létezik az adatbázis-kezelők világában, amelyek mind más-más irányba viszik el ezt a tudományágat. Néhány ezek közül teljesen új technológiát jelent, amelyek megváltoztatják a hagyományos adatbázis-kezelés természetét. Ilyenek például az objektumorientált programozás, a megszorítások és triggerek, a multimédia-adatok, vagy a World Wide Web. Más irányzatok újfajta alkalmazástípusok kifejlesztését tűzik ki célul, mint amilyen például az adattárház, vagyis az információk egységesítése. Ebben a szakaszban rövid bevezetést nyújtunk a jövő adatbázis-kezelőinek legfőbb irányzataiba.

1.3.1. Típusok, osztályok, objektumok

Az objektumorientált programozást széles körben tekintik olyan eszköznek, ami a programok jobb szervezését és ezáltal a megbízhatóbb programok írását segíti elő. Először a Smalltalk nyelvben jelent meg a módszer, majd a C++ nyelv kifejlesztésével vált igazán elterjedtté. Ezt követően nagyon sok korábban C-ben írt alkalmazást írtak át C++-ra. Legújabban a Java nyelv az, amelyik az objektumorientált programozás felé irányítja sokak figyelmét. Ez a nyelv alkalmas arra, hogy a programokat megoszthassuk a hálózaton. Az adatbázisok világa is elmozdult az objektumorientáltság felé, és néhány cég már olyan adatbázis-kezelőt forgalmaz, amelyet „objektumorientáltnak" nevez. Ebben a szakaszban áttekintjük az objektumorientáltság mögött rejlő fogalmakat.

A típusrendszer

Egy objektumorientált programozási nyelv a típusok széles választékát kínálja a felhasználónak. Az *alaptípusokból* kiindulva – amelyek általában az egészek, valós számok, logikai értékek és a karakterláncok – újabb típusokat hozhatunk létre a *típuskonstruktorok* segítségével. A típuskonstruktorok segítségével általában a következő típusok hozhatók létre:

1. *Rekordstruktúrák.* Ha adott a $T_1, T_2, \ldots, T_n$ típusokból álló lista és a hozzájuk tartozó $f_1, f_2, \ldots, f_n$ mezőnevekből álló lista (a Smalltalk ezeket nevezi *példányváltozóknak*), akkor létrehozhatunk egy n komponensből álló rekordtípust. Az i-edik komponens típusa T_i és a megfelelő f_i mezőnévvel hivatkozhatunk rá. A rekordstruktúrák pontosan megfelelnek a C és C++ nyelv „struct" típusának.

2. *Kollekciótípusok.* Ha adott egy T típus, akkor új típusokat hozhatunk létre úgy, hogy valamilyen *kollekcióoperátort* alkalmazunk T-re. A különböző nyelvek különböző kollekcióoperátorokat használnak, de van néhány általánosan használt, mint a tömbök, listák és a halmazok. Így ha T az egész típus, akkor létrehozhatjuk az „egészekből álló tömb", az „egészekből álló lista" és az „egészekből álló halmaz" kollekciótípusokat.

3. *Hivatkozástípus.* A T típusra való hivatkozás egy olyan típus, amelynek értékei segítségével megkereshetünk egy T típusú értéket. A C-ben vagy a C++-ban a hivatkozás egy mutató, amelyben a mutatott érték memóriabeli címét tároljuk. A mutatók példáján keresztül könnyebb megérteni a hivatkozásokat. Az adatbázisrendszerekben azonban, ahol az adatok jó néhány lemezen, sokszor különböző számítógépeken helyezkednek el, a hivatkozás sokkal bonyolultabb dolog, mint a mutató. Itt a hivatkozás tartalmazhatja a számítógép nevét, a lemezegység számát, azon belül egy blokk azonosítóját és a blokkban egy pozíciót, ahol a keresett érték tárolva van.

Természetesen a rekordstruktúra- és a kollekcióoperátorokat többször is alkalmazhatjuk, és ezáltal egészen bonyolult típusokat hozhatunk létre. Például definiálhatunk egy rekordstruktúra típust, melynek az első komponensét `ügyfél`-nek hívjuk, és legyen ennek típusa karakterlánc, a második komponens neve pedig `számlák`, amelynek típusa egészekből álló halmaz. Egy ilyen típus segítségével összerendelhetjük a bank ügyfeleit a saját számláik halmazával.

Osztályok és objektumok

Egy osztály egy típusból és opcionálisan egy vagy több függvényből vagy eljárásból áll (ezeket *metódusoknak* nevezzük), amelyeket végrehajthatunk az osztály objektumaira. Az osztály objektumai vagy a megadott típusú értékek (ezek az *állandó objektumok*), vagy olyan változók, amelynek értékei az adott típusúak lehetnek (ezek a *változó objektumok*). Például, ha definiáljuk a C osztályt, amelynek típusa „egészekből álló halmaz" akkor a $\{2, 5, 7\}$ egy állandó objektuma az osztálynak, míg az s változót deklarálhatjuk úgy, hogy típusa a C osztály és értékül adhatjuk neki a $\{2, 5, 7\}$ halmazt.

Objektumazonosító

Feltesszük, hogy minden objektumnak van egy *objektumazonosítója* (OID). Két különböző objektumnak nem lehet azonos az azonosítója, és minden objektumnak egyetlen azonosítója van. Az objektumazonosító az az érték, amivel egy, az objektumra való hivatkozás rendelkezik. Gondolhatunk úgy az objektumazonosítóra, mint egy memóriabeli mutatóra, amelyik az objektumra mutat, de amint azt már a hivatkozástípusnál említettük, az adatbázisok esetében ez az azonosító jóval bonyolultabb is lehet. Nevezetesen annyi bitből állhat, amennyi elegendő ahhoz, hogy az objektumot megtaláljuk jó néhány gép másodlagos és harmadlagos tárolóeszközein. Továbbá, mivel az adatok itt állandónak tekinthetők, ezért az azonosítónak érvényesnek kell lennie egészen addig, amíg az adat létezik.

Metódusok

Az osztályokhoz általában hozzá vannak rendelve bizonyos függvények, amelyeket *metódusoknak* hívunk. Egy *C* osztályhoz tartozó metódusnak van legalább egy argumentuma, egy *C* osztálybeli objektum. Ezen kívül még további argumentumai is lehetnek, amelyek akármelyik osztályba tartozhatnak, akár *C*-be is. Például egy „egészekből álló halmaz" típusú osztálynak lehet olyan metódusa, amelyik előállítja egy adott halmaz hatványhalmazát, olyan metódusa, amelyik két halmaz unióját állítja elő, vagy olyan metódusa, amelyik egy logikai értéket ad vissza annak függvényében, hogy a halmaz üres vagy nem.

Absztrakt adattípusok

Sok esetben az osztályok egyben absztrakt adattípusok is, ami azt jelenti, hogy *tokba foglalják* az objektumaikat, vagy más szóval megakadályozzák az azokhoz való tetszőleges hozzáférést. Csak az osztály metódusai módosíthatják az osztálybeli objektumokat közvetlenül. Ez a megszorítás garantálja, hogy az objektumokat nem lehet megváltoztatni olyan módon, amit az osztály megalkotója ne tervezett volna előre. Ez a koncepció az egyik legfontosabb eszköze a megbízható szoftverek fejlesztésének.

Osztályhierarchiák

Egy *C* osztályt deklarálhatunk úgy, mint egy másik, *D* osztály *alosztályát*. Ilyenkor a *C* osztály *örökli* a *D* összes tulajdonságát, mind a típusát, mind a függvényeit. Ezenkívül *C*-nek lehetnek további tulajdonságai is. Új metódusokat definiálhatunk a *C* osztálybeli objektumokra, amelyek akár helyettesíthetik a *D* metódusait, vagy akár bővíthetik azok halmazát. Sőt még arra is van lehetőségünk, hogy a *D* típusát valamilyen értelemben bővítsük. Nevezetesen, ha *D* típusa valamilyen rekordstruktúra típus, akkor ehhez újabb mezőket adhatunk, amelyek csak a *C* típusú objektumokban fognak szerepelni.

1.3. példa: Vegyünk egy osztályt, amelynek az objektumai bankszámlaszámok. Az osztály típusát a következőképpen adhatjuk meg:

```
CLASS Számla ={ számlaszám: integer;
                egyenleg: real;
                tulajdonos: REF Ügyfél;
              }
```

Vagyis a `Számla` osztály típusa egy rekordstruktúra, amelynek három mezője van: az egész típusú számlaszám, a valós típusú egyenleg és a tulajdonos, ami egy `Ügyfél` osztálybeli objektumra való hivatkozás. (Erre az `Ügyfél` osztályra is szükség van az adatbázisban, de ennek a típusát itt most nem írtuk le.)

Miért jók az objektumok?

Az objektumorientált programozás számos fontos lehetőséget kínál az adatbázisrendszerek számára.

- A gazdag típusrendszer segítségével az adatokat olyan formában kezelhetjük, amelyik természetesebb a relációknál vagy a korábbi adatmodelleknél. A relációs modell meglehetősen szűk típusrendszert kínál. A relációk rekordokból álló halmazok, és a rekordok struktúrája olyan, hogy a mezők (amelyeket a relációs modellben attribútumoknak nevezünk) típusa csak valamilyen alaptípus lehet.
- Az osztályok és az osztály-hierarchia segítségével könnyebben megoszthatjuk, és többször felhasználhatjuk a sémákat és a programokat, mint a hagyományos rendszerek esetében.
- Az absztrakt adattípusok segítségével megakadályozhatjuk az adatok helytelen felhasználását, hiszen az adatokhoz csak néhány alaposan megtervezett függvény segítségével lehet hozzáférni, amelyek az adatokat bizonyosan helyesen fogják kezelni.

Definiálunk néhány metódust is az osztályhoz. Legyen az alábbi egy olyan metódus, amelyik a `Számla` típusú *a* objektum `egyenleg`-ét megnöveli *m*-mel.

```
Növel(a: Számla, m: real)
```

Végül adjuk meg néhány alosztályát a `Számla` osztálynak. A lekötött számlának például lehet egy további mezője a `lejáratDátuma`. Ez az az időpont, amikor a tulajdonos felveheti a számláján levő pénzt. A `LekötöttSzámla` alosztálynak lehet egy további metódusa, a kamatveszteség.

```
Kamatveszteség(a: LekötöttSzámla)
```

Ez kiszámolja a lejáratnál korábbi pénzfelvétel esetén az ügyfelet érő kamatveszteséget. Ehhez felhasználja a paraméterül kapott `LekötöttSzámla` típusú `a` objektum `lejáratDátuma` mezőjét és az aktuális dátumot. Ez utóbbit a rendszer dátumból kaphatja meg. □

Az adatbázisrendszerek objektumorientált megközelítéseit mélyrehatóan fogjuk tanulmányozni a könyvben. A 2.1. szakaszban bemutatjuk az objektumorientált adatbá-

zis-tervező nyelvet, az ODL-t, a 8. fejezetet pedig az objektumorientált lekérdezőnyelveknek szenteljük. Ott majd áttekintjük az OQL nyelvet, amelyik lassan szabvánnyá kezd válni az objektumorientált lekérdezőnyelvek terén, valamint az SQL nyelvnek az objektumorientált irányba történő tervezett bővítéseit.

1.3.2. Megszorítások és triggerek

Egy másik, napjainkban terjedő irányzat az, hogy az adatbázisrendszerek úgynevezett aktív elemeket használnak. Az aktív alatt itt azt értjük, hogy az adatbázis adott komponense mindig elérhető és készen áll arra, hogy a rendszer végrehajtsa, amikor szükség van rá. Az adatbázisrendszerekben alkalmazott aktív elemek két fajtája a következő:

1. *Megszorítások.* A megszorítások logikai értékű függvények, amelyek által visszaadott értéktől azt várjuk el, hogy igaz legyen. Például egy banki adatbázisban megadhatunk egy olyan megszorítást, amelyik azt mondja ki, hogy az egyenleg nem lehet negatív. Az olyan módosításokat, amelyek ezt a megszorítást megsértik az adatbázis-kezelő rendszer visszautasítja. Ilyen lehetne például egy pénzfelvétel, amelyik után az egyenleg 0 alá csökkenne.

2. *Triggerek.* A trigger egy programkódrészlet, amelyik egy *esemény* bekövetkezésére vár. A lehetséges események például a beszúrás, a módosítás vagy a törlés. Amikor az esemény bekövetkezik, a megfelelő műveletsorozat végrehajtódik, vagy más szóval *kiváltódik.* (trigger = kivált, elsüt) Például a repülőgép-helyfoglalási rendszerben lehet egy olyan szabály, amelynek kiváltó oka az lehet, ha egy járat státusát töröltre módosítják. A végrehajtódó művelet pedig egy lekérdezés, amelyik megkeresi az adott járat utasainak telefonszámait, hogy értesíteni lehessen őket. Egy még bonyolultabb művelet lehetne az, hogy megpróbáljuk az utasokat automatikusan más járatokon elhelyezni.

Az aktív elemek alkalmazása nem teljesen új ötlet, hiszen ezek már megjelentek a PL/I programozási nyelv úgynevezett „ON-feltételeiben". Szintén előfordultak ilyenek a mesterséges intelligenciarendszerekben már évekkel ezelőtt, és az operációs rendszerekben alkalmazott démonok is hasonlók ezekhez. Ha azonban az adatok mérete, amelyen az aktív elem dolgozik nagyon nagy, vagy az aktív elemek száma nagy, akkor ezek megvalósítása komoly technikai problémákat jelent. Éppen ezért az aktív elemek egészen a 90-es évek elejéig nem váltak szabványos részévé az adatbázis-kezelő rendszereknek. Az aktív elemeket a 6. fejezetben tárgyaljuk majd.

1.3.3. Multimédia-adatok

Szintén fontos új irányzatnak tekinthető az adatbázis-kezelő rendszerekben a multimédia-adatok kezelése. Multimédián általában a legkülönfélébb formában tárolt információkat értjük. A leggyakoribb formái a multimédia-adatoknak a videó, audió, radarjelek, műholdas felvételek, és különféle módon kódolt dokumentumok és képek. Ezekben az a közös vonás, hogy sokkal nagyobb adatmennyiséget jelentenek, mint a korábbi adatformátumok – mint például az egészek, valósak vagy karakterláncok – és a méreteikben is nagyon jelentős különbségek vannak.

A multimédia-adatok tárolása az adatbázis-kezelő rendszerek különböző irányú bővítésével járt. Például a multimédia-adatokon végrehajtott műveletek nem olyan egyszerűek, mint a hagyományos adatformák műveletei. Így például egy banki adatbázisban a negatív egyenlegű számlákat megkereshetjük úgy, hogy az összes egyenleget összehasonlítjuk a 0 számmal, de egy képeket tároló adatbázisban egy adott arcképhez „hasonló" képek megkeresése már nem nagyon megvalósítható. Ezért az adatbázis-kezelőnek lehetővé kell tennie a felhasználó számára, hogy saját függvényeket definiálhasson, amiket a multimédia-adatokra alkalmazhat. Gyakran az objektumorientált megközelítést alkalmazzák ezeknek a kiterjesztéseknek a megvalósításánál még a relációs rendszerek esetében is.

A multimédia-objektumok mérete is arra készteti az adatbázis-kezelőket, hogy a tárkezelőt alkalmassá tegyék gigabájt méretű sorok kezelésére. A nagyméretű adatelemek számos problémát vetnek fel, amelyek közül az egyik a lekérdezés eredményének a továbbítása. A hagyományos relációs adatbázisban az eredmény egy sorhalmaz, amit a szerver teljes egészében továbbít a kliens felé.

Most azonban képzeljük el, hogy a lekérdezés végeredménye egy gigabájt méretű videoklip. Itt már nem megvalósítható, hogy a szerver az egészet továbbítsa a kliens felé. Egyrészt ez túl sokáig tartana, és megakadályozná a szervert abban, hogy más kérésekkel foglalkozzon. Másrészt lehet, hogy a kliens csak egy kis részét szeretné megkapni a filmnek, de pontosan nem tudja megfogalmazni, hogy melyik részét anélkül, hogy látná a film elejét. És ha még a teljes filmet szeretné is megkapni a kliens, mondjuk azért, hogy lejátssza egy képernyőn, akkor is elegendő lenne azt valamilyen ütemben továbbítani egy teljes óra leforgása alatt. (Körülbelül ennyi ideig tart egy gigabájtnyi tömörített videó lejátszása.) Ezért a multimédia-adatbázisok tárolórendszerét fel kell készíteni arra, hogy az eredményt interaktív módon továbbítsák a kliens felé. Ez történhet úgy, hogy a válasznak mindig egy kis részét külön kérésre kapja meg a kliens, vagy történhet valamilyen rögzített ütemezéssel is.

1.3.4. Adatok egységesítése

Ahogy az információ szerepe egyre alapvetőbbé válik a munkában és a szórakozásban, azt figyelhetjük meg, hogy a meglevő információforrásokat egyre többféle módon hasznosítjuk. Vegyünk például egy vállalatot, amelyik az összes termékét egy állandóan elérhető katalógusba szeretné gyűjteni, hogy a vevők a World Wide Web

segítségével bármikor megnézhessék és megrendelhessék azokat. Egy nagyvállalatnak sok részlege van, amelyek mindegyike külön adatbázissal rendelkezik a saját termékeiről. A részlegek használhatnak különböző adatbázis-kezelőket, különböző információstruktúrákat, még az is előfordulhat, hogy különböző terminológiát használnak ugyanarra a dologra, vagy, hogy ugyanazt a terminológiát használják különböző dolgok megnevezésére.

1.4. példa: Képzeljünk el egy vállalatot néhány részleggel, amelyik lemezeket gyárt. Az egyik részleg katalógusában a forgási sebesség percenként van megadva, egy másikéban másodpercenként. Egy harmadik részleg katalógusa esetleg ezt az adatot egyáltalán nem tartalmazza. Egy hajlékonylemezeket készítő részleg „lemez"-ként hivatkozik a termékeire, miközben egy merevlemezeket gyártó részleg szintén „lemez"-nek nevezi a termékeit. A lemezen levő sávok számára az egyik részleg „sávok"-ként hivatkozik, míg egy másik „cilinderek"-ként. □

A központi ellenőrzés nem jelent mindig megoldást. A részlegek lehet, hogy sok pénzt fordítottak korábban a saját adatbázisuk létrehozására akkor, amikor még az egységesítés kérdése nem merült fel. Egy részleg lehet, hogy korábban önálló vállalat volt, amit csak nemrég vett meg a cég. Ezért vagy más egyéb okok miatt ezek az úgynevezett „örökölt" adatbázisok nem cserélhetők le olyan könnyen. Ezért a cégnek az ilyen adatbázisok fölé kell építenie valamilyen struktúrát, hogy a vevők felé egy egységes termékkínálatot mutathasson.

Az egyik népszerű megközelítés az *adattárház* létrehozása. Itt a különböző adatbázisokból származó adatokat megfelelő átalakítás után egy központi adatbázisba másolják. Ahogy az egyes adatbázisok változnak, úgy módosítják az adattárházat is. Ezt azonban nem feltétlenül kell azonnal elvégezni. Általában ezt éjszakánként végzik, amikor az egyes adatbázisok kevésbé vannak leterhelve.

A különálló adatbázisok így továbbra is elláthatják azt a feladatot, amire létrehozták őket. Az új funkciókat – mint amilyen például az on-line-katalógus a hálózaton keresztül – az adattárház szintjén valósítják meg. Az adattárházak a tervezés és elemzés céljait is szolgálhatják. Például a vállalati elemzők az adattárházból kérdezhetik le az eladások alakulását, és ezáltal a készletezés és a gyártás jobban tervezhetővé válik. Az *adatbányászat,* amely az adatok között lévő érdekes és szokatlan minták keresésével foglalkozik, szintén az adattárházak segítségével alakulhatott ki. A növekvő eladások érdekében ma már komoly igényként merül fel ezeknek a mintáknak a vizsgálata.

1.4. A könyv felépítése

Az adatbázisrendszerekkel kapcsolatos fogalmak három fő területre oszthatók:

1. *Adatbázis-tervezés.* Hogyan alakítsunk ki egy jól használható adatbázist? Mely információk kerüljenek be az adatbázisba? Milyen legyen az adatok szerkezete? Milyen feltételezésekkel éljünk az adatelemek típusára és értékeire vonatkozóan? Hogyan kapcsolódjanak egymáshoz az adatelemek?

2. *Adatbázis-programozás.* Hogyan fejezzük ki az adatbázisra vonatkozó lekérdezéseket és egyéb műveleteket? Hogyan használjuk ki az adatbázis-kezelő rendszerek egyéb lehetőségeit, mint amilyenek a tranzakciók vagy a triggerek?

3. *Adatbázis-megvalósítás.* Hogyan készítsünk el egy adatbázis-kezelő rendszert? Hogyan valósítsuk meg a lekérdezésfeldolgozót, a tranzakció-kezelőt? Hogyan szervezzük meg az adatok tárolását, hogy az adathozzáférés hatékony legyen?

Noha az adatbázisok megvalósítása egy jelentős szelete a szoftveriparnak, mégis azok száma, akik adatbázis-tervezéssel, vagy az adatbázisok használatával foglalkoznak jóval nagyobb azokénál, akik az adatbázisrendszereket megvalósítják. E könyv célja az, hogy bevezetést adjon az adatbázisrendszerekbe, ezért úgy látjuk jónak, ha az első két területre, a tervezésre és a programozásra összpontosítunk. Ebben a fejezetben megpróbáltunk egy rövid bepillantást adni a harmadik területbe, a megvalósításba, de a könyv folyamán ezt a témát már nem fogjuk érinteni. A hátralévő fejezetekben szereplő anyagot a tervezés és programozás kérdéseinek szenteltük a következő felosztásban.

1.4.1. Tervezés

A második és harmadik fejezet a tervezésről szól. A második fejezet elején két magas szintű adatbázis-tervező nyelvet mutatunk be. Az egyik az ODL (Object Definition Language = Objektum Definíciós Nyelv), egy objektumorientált nyelv az osztályok deklarálására. A másik az E/K vagy Egyed/Kapcsolat modell, ami egy grafikus jelölési rendszer az adatbázis szerkezetének a leírására.

Sem az ODL-nek, sem az E/K modellnek nem az a célja, hogy közvetlenül használjuk az adatbázis-szerkezet megadására, habár az ODL nagyon közel áll egy adatdefiníciós nyelvhez, ha az adatbázis-kezelő rendszer objektumorientált. A cél inkább az, hogy az ezekben a modellekben megadott tervet később lefordítsuk bármilyen formális jelölési rendszerre, amit az adott adatbázis-kezelő adatdefiníciós nyelve használ. Mivel a legtöbb adatbázis-kezelő relációs, ezért az ODL-nek és az E/K modellnek a relációs modellre történő lefordítására fogunk koncentrálni. A harmadik fejezetet a relációs modellnek és ennek a fordítási folyamatnak szenteltük. Később az 5.7. szakasz-

ban megmutatjuk majd, hogy a relációs adatbázissémákat hogyan lehet formálisan átírni az SQL nyelv adatdefiníciós részére.

A harmadik fejezetben bevezetjük a függőségek fogalmát, amelyek tulajdonképpen formálisan megfogalmazott elvárások a reláció sorai között fennálló kapcsolatokra vonatkozóan. A függőségek segítségével tudjuk a relációkat kedvező módon részekre bontani. Ezt a folyamatot a relációk normálformára hozásának vagy normalizálásnak nevezzük. A függőségeket és a normalizálást a 3.5. és az azt követő szakaszokban tárgyaljuk. Mindezek fontos részét képezik a relációs adatbázisok tervezésének. E téma hasznos azok számára is, akik közvetlenül a relációs modellben tervezik meg az adatbázist, és azok számára is akik egy ODL vagy E/K tervet alakítanak át relációkká.

1.4.2. Programozás

A negyediktől a nyolcadikig tartó fejezetek az adatbázis-programozásról szólnak. A negyedik fejezetet a relációs modell lekérdezéseinek absztrakt tárgyalásával kezdjük, és bevezetjük a relációkon értelmezett műveleteknek egy családját, amelyeket együttesen relációs algebrának nevezünk. Ezután bemutatunk egy másik módszert, amelyik szintén a lekérdezések leírására szolgál. Ez logikai kifejezéseken alapul, Datalog a neve.

Az ötödiktől a hetedikig tartó fejezeteket az SQL programozásnak szenteljük. Ahogy korábban már említettük, ma az SQL az általánosan használt lekérdezőnyelv. Az ötödik fejezet bevezeti az SQL lekérdezéseknél és az adatbázissémák SQL-beli megadásánál használt alapvető fogalmakat. Az ötödik és az azt követő két fejezet csaknem teljes egészében az SQL-nek egy szabványos változatán, az SQL2-n alapul. Az SQL programozásnak azonban vannak bizonyos részei, amelyeket néhány adatbázis-kezelő rendszerben már megvalósítottak, de nem részei az SQL2-nek. Ezekben az esetekben egy későbbi szabványt használunk, az SQL3-at, amelyet még formálisan nem fogadtak el.

A hatodik fejezet az SQL-nek a triggerekkel és megszorításokkal kapcsolatos lehetőségeiről szól. Mivel az SQL2 lehetőségei e területeken elég korlátozottak, ezért a triggereket és megszorításokat külön-külön tárgyaljuk az SQL2-n és az SQL3-on belül.

A hetedik fejezetben az SQL programozás bizonyos speciális kérdéseire térünk ki. Mivel az SQL programozás legegyszerűbb modellje egy önálló, általános lekérdező interfész, ezért a gyakorlatban az SQL programokat általában beágyazzuk valamilyen hagyományos nyelven megírt programba, mint amilyen például a C. A hetedik fejezetben megmutatjuk, hogy hogyan lehet az SQL utasításokat a környező program utasításaival összekapcsolni, valamint azt, hogy hogyan valósítható meg az adatcsere az adatbázis és a program változói között. Szintén a hetedik fejezetben mutatjuk meg az SQL olyan lehetőségeit, amelyekkel tranzakciókat definiálhatunk, kapcsolatot teremthetünk a kliens és a szerver között, vagy az adatbázishoz való hozzáférést korlátozhatjuk a felhasználók számára.

A nyolcadik fejezetben az adatbázis-programozás kialakulóban lévő objektumorientált szabványaival foglalkozunk. Itt majd két különböző irányzatot vizsgálunk meg. Az első az OQL (Object Query Language = Objektum Lekérdezőnyelv), egy olyan kísérletnek tekinthető, amelyik a C++ nyelvet szeretné kompatíbilissé tenni a magas szintű adatbázis-programozás követelményeivel. A másik irányzat, az SQL3-ban megtalálható objektumorientált lehetőségek, olyan kísérletnek tekinthető, amelyik megpróbálja a relációs adatbázisokat és az SQL-t az objektumorientált programozással kompatíbilissé tenni. Bizonyos mértékig ez a két megközelítés közös alapon nyugszik. Van azonban néhány dolog, amiben alapvetően különböznek egymástól.

1.5. Összefoglalás

- *Adatbázis-kezelő rendszerek:* Egy adatbázis-kezelő rendszer lehetővé teszi a tervezőknek, hogy az információkat megfelelő szerkezetben ábrázolják, lehetővé teszi a felhasználóknak, hogy az információkat lekérdezzék és módosítsák, segít a nagy mennyiségű adatok kezelésében és az adatokon történő egyidejű műveletek kezelésében.

- *Adatbázisnyelvek:* Külön nyelvi elemek szolgálnak az adatbázis szerkezetének a megadására (adatdefiníciós nyelv) és az adatok lekérdezésére, illetve módosítására (adatmanipulációs nyelv).

- *Relációs adatbázisrendszerek:* Manapság a legtöbb adatbázisrendszer a relációs adatmodellen alapul, amelyik az információkat táblákba szervezetten tárolja. Ezekben a rendszerekben az SQL a leggyakrabban használt nyelv.

- *Objektumorientált adatbázisrendszerek:* Néhány újabb adatbázisrendszer az objektumorientált adatmodellezésben szokásos fogalmakat használja. Ilyenek az osztályok, a fejlett típusrendszer, az objektumazonosítás és a tulajdonságok öröklése az alosztályok által. A jövőben a legtöbb adatbázis-kezelő rendszer – köztük a relációsok is – valószínűleg támogatni fogja ezeket a fogalmakat.

- *Másodlagos és harmadlagos tárolás:* A nagy adatbázisokat másodlagos tárolóeszközökön tárolják, amelyek általában lemezek. A még nagyobb adatbázisok már harmadlagos tárolóeszközöket igényelnek, amelyeknek a kapacitása néhány nagyságrenddel nagyobb mint a lemezeké, de az elérésük is néhány nagyságrenddel lassabb.

- *Az adatbázis-kezelő részei:* Az adatbázis-kezelő rendszer legfontosabb részei a lekérdezésfeldolgozó, a tranzakció-kezelő és a tárkezelő.

- *A tárkezelő:* A tárkezelő kezeli a másodlagos tárolóeszközökön lévő adatállományokat és a memóriapuffereket, amelyekben az adatok egy részét tároljuk. Az adatbázis-kezelő rendszerek általában indexeket is használnak, amelyek az adatok hatékonyabb elérését elősegítő adatszerkezetek.

- *A lekérdezésfeldolgozó:* A lekérdezésfeldolgozó egyik fontos feladata a lekérdezések optimalizálása, vagyis a lekérdezés megválaszolásához egy jó algoritmus keresése.

- *A tranzakció-kezelő:* A tranzakciók az adatbázison végzett alapvető munkaegységek. A tranzakció-kezelő több tranzakció egyidejű futását engedi meg és eközben biztosítja, hogy a tranzakciók megfeleljenek a következő követelményeknek: atomosság, elkülönítés, következetesség és tartósság.

- *Kliens-szerver rendszerek:* Az adatbázis-kezelő rendszerek általában a kliens-szerver felépítést követik, ahol a legfontosabb adatbázis-komponensek a szerver oldalon futnak, a kliens pedig a felhasználóval való kommunikációt kezeli.

- *Aktív adatbáziselemek:* A modern adatbázisrendszerek támogatják az aktív elemek valamilyen formáit, leggyakrabban a triggereket és a megszorításokat.

- *A jövő rendszerei:* A legfőbb új irányzatok az adatbázisrendszerek terén a nagyméretű multimédia-objektumok, mint például a videók és képek támogatása, valamint az információk egységesítése különálló információforrásokból egyetlen adatbázisba.

1.6. Irodalomjegyzék

Számos könyv foglalkozik az adatbázisok megvalósításának legfontosabb kérdéseivel. [3]-ban és [5]-ben a tranzakció-kezelő megvalósításáról olvashatunk. Az osztott adatbázis-kezelők megvalósításának kérdéseiről szintén e könyvekben, valamint [7]-ben találunk információt. A fájlkezelők megvalósítását [11] tárgyalja. Az objektumorientált adatbázis-kezelőkkel [2], [4] és [6] foglalkozik. Az adatbázisrendszerek elméletét [1], [9] és [10] tárgyalja. Végül pedig sok olyan publikáció megtalálható [8]-ban, amelyek jelentősen befolyásolták az adatbázisok világát.

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

2. Bancilhon, F., C. Delobel, P. Kanellakis, *Building an Object-Oriented Database System,* Morgan-Kaufmann, San Francisco, 1992.

3. Bernstein, P. A., V. Hadzilacos, N. Goodman, *Concurrency Control and Recovery in Database Systems,* Addison-Wesley, Reading, MA, 1987.

4. Cattell, R. G. G., *Object Data Management,* Addison-Wesley, Reading, MA, 1994.

5. Gray, J. N. és A. Reuter, *Transaction Processing: Concepts and Techniques,* Morgan-Kaufmann, San Francisco, 1993.

6. Kim, W. (szerk.), *Modern Database Systems: The Object Model, Interoperability, and Beyond,* ACM Press, New York, 1994.

7. Oszu, M. T., P. Valduriez, *Principles of Distributed Database Systems,* Prentice Hall, Englewood Cliffs, NJ, 1991.

8. Stonebraker, M. (szerk.), *Readings in Database Systems,* Morgan-Kaufmann, San Francisco, 1994.

9. Ullman, J. D., *Principles of Database and Knowledge-Base Systems, Volume I,* Computer Science Press, New York, 1988.

10. Ullman, J. D., *Principles of Database and Knowledge-Base Systems, Volume II,* Computer Science Press, New York, 1989.

11. Wiederhold, G., *Database Design,* McGraw-Hill, New York, 1993.

2. fejezet

Adatmodellezés

Egy adatbázis tervezése a feldolgozandó információk elemzésével és ezen információk komponensei közötti kapcsolatok meghatározásával kezdődik. Az adatbázis struktúráját *adatbázissémának* nevezzük. Ennek a struktúrának leírására különböző nyelvek, modellek és jelölésrendszerek léteznek. Az elemzés után a terv készül el valamilyen modellben, amit utána felhasználunk az adatbázis fizikai megvalósításához.

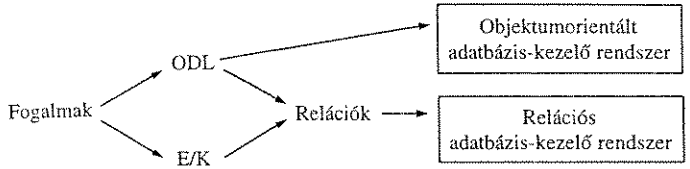
Ebben a könyvben az adatbázis-tervezéshez kétféle modellt fogunk használni. A hagyományosabb modell az egyed-kapcsolat (E/K) diagram[1], amely téglalapokkal és nyilakkal reprezentálja a lényeges adatelemeket és a köztük lévő kapcsolatokat. Párhuzamosan bevezetjük az ODL-t (Object Definition Language), amely egy objektumorientált megközelítése az adatbázis-tervezésnek úgy, hogy illeszkedik a szabványos objektumorientált adatbázisrendszerekhez. Ez a fejezet tartalmaz két további modellt, a hálós és a hierarchikus modelleket, amelyeknek elsődlegesen történelmi érdekességük van. Ezek bizonyos értelemben csökkentett verziói az ODL-nek. Már a 70-es években implementálták és használták őket a kereskedelmi adatbázisrendszerekben.

A 3. fejezetben a relációs adatmodellel foglalkozunk, ahol a valós világot táblák segítségével reprezentáljuk. A relációs adatmodell némi megszorítást tartalmaz a struktúrákra vonatkozólag. Azonban a modell meglehetősen egyszerű és hasznos, így a nagy kereskedelmi adatbázis-kezelő rendszerek manapság ezen alapszanak. Gyakran kezdődik az adatbázis-tervezés azzal, hogy kifejlesztik a sémát E/K diagram vagy objektum alapú modell segítségével, és utána ezt transzformálják relációs adatmodellé.

A 2.1. ábra mutatja a fejlesztési eljárást. A modellezni kívánt információval kapcsolatos fogalmakból indulunk ki. Ezeket a fogalmakat lefordítjuk egy tervező nyelvre, például E/K diagramra vagy ODL-re. (Természetesen más lehetőségek is vannak.) Az esetek legnagyobb részében a tervet végül egy relációs adatbázis-kezelő rendszerrel valósítjuk meg. Ha így van, akkor egy meglehetősen mechanikus eljárással, amit a 3. fejezetben tárgyalunk, az absztrakt tervet konvertáljuk egy konkrét relációs adatbázisba.

Van egy másik lehetőség is, ahol az ODL tervet egy objektumorientált adatbázis-

[1] Fordító megjegyzése: Az angol irodalomban az E/K diagram jele: E/R (Entity-Relationship) diagram.



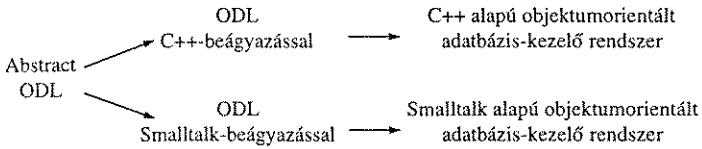
2.1. ábra. *Az adatbázis-modellezés és implementálás eljárása*

kezelő rendszer számára konvertáljuk. Ebben az esetben a transzformálás még automatikusabb, ugyanis nagyon egyszerű egy ODL stílushoz hozzárendelni egy másik objektumorientált programozási nyelvet, mint a C++-t vagy a Smalltalkot.

2.1. Bevezetés az ODL-be

Az ODL egy javasolt szabvány, amely segítségével adatbázisok struktúráját specifikálhatjuk objektumorientált terminológiával úgy, ahogy ezt más nyelveknél, mint a C++ vagy Smalltalk, megszoktuk. Az ODL egy kiterjesztése az IDL-nek (Interface Description Language), illetve egy komponense a CORBA-nak (Common Object Request Broker Architecture). Ez utóbbi egy szabvány az osztott objektumorientált számításokhoz.

Az ODL elsődleges célja, hogy támogassa az adatbázisok objektumorientált tervezését, és utána ennek a tervnek a közvetlen transzformálását objektumorientált adatbázis-kezelő rendszerek (OODBMS) deklarációiba. Mivel az ilyen rendszerekben az elsődleges nyelv rendszerint a C++ vagy a Smalltalk, így az ODL-t ezen nyelvek egyikének deklarációira kell transzformálni. Az ODL hasonlít ezekre a nyelvekre (bár a C++-ra jobban), így a 2.2. ábra által javasolt transzformálás meglehetősen egyszerű. Összehasonlításképpen, mind az ODL, mind az egyed-kapcsolat diagram transzformálása relációs adatbázis-kezelő rendszer (RDBMS) deklarációiba sokkal összetettebb.



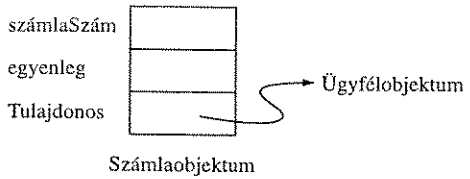
2.2. ábra. *ODL konvertálása egy objektumorientált adatbázis-kezelő rendszer deklarációiba*

2.1.1. Objektumorientált tervezés

Egy objektumorientált terv a valós világot modellezi *objektumok* segítségével, amelyek valamilyen fajta észlelhető egyedek. Például tekinthetjük objektumoknak az embereket, a bankszámlákat, a repülőjegyeket, a főiskolai kurzusokat, épületeket stb. Az

objektumokról feltételezzük, hogy van egy egyértelmű *objektumazonosítójuk* (OID), ami megkülönbözteti őket más objektumoktól, mint az 1.3.1. részben már említettük.

Az információszervezéshez rendszerint csoportosítjuk az objektumokat hasonló tulajdonságú objektumok *osztályaiba*. Az objektum és az osztály fogalma az adatbázisok esetében lényegében ugyanaz, mint az objektumorientált programozási nyelveknél, például C++ vagy Smalltalk (idézzük fel az objektumorientált fogalmakat az 1.3.1. részből). Azonban, amikor ODL objektumorientált tervezésről beszélünk, akkor egy osztályban a „hasonló tulajdonságú" objektumokra kell gondolnunk a következő két különböző módon:



2.3. ábra. *Egy számlát reprezentáló objektum*

- Egy osztály objektumai által reprezentált valós fogalmaknak hasonlóaknak kell lenniük. Például értelmezi, hogy egy bank összes ügyfelei egy osztályban vannak, míg a banknál lévő számlák egy másik osztályban vannak. Ugyanakkor nincs értelme, hogy egy osztályban legyenek az ügyfelek és a számlák, mert nagyon kevés vagy inkább semmi közös sincs bennük, és lényegesen különböző a szerepük a banki világban.

- Egy osztályban az objektumok tulajdonságainak ugyanazoknak kell lenni. Amikor programozunk egy objektumorientált nyelven, gyakran tekintjük az objektumokat

Sémák és adatok

Az ODL egy nyelv, amellyel séma-, illetve adatbázis-struktúra specifikálható. Az ODL nem ad lehetőséget az adatbázis aktuális tartalmának megadására, sem a lekérdezésekre, sem pedig az adatok manipulálására. Mint már említettük az 1.1. részben, a sémát specifikáló nyelveket, mint az ODL-t, gyakran hívják *adatdefiníciós nyelveknek*, míg azokat a nyelveket, amelyek segítségével megadhatjuk az adatbázis tartalmát, vagy lekérdezhetünk, vagy pedig módosíthatjuk az adatokat, *adatmanipulációs nyelveknek* nevezzük. Az adatmanipulációs nyelveket nem tárgyaljuk addig, amíg meg nem vizsgáltuk az adatbázisokat a felhasználó szempontjából a 4. fejezetben. Az adatdefiníciós nyelvek viszont a tervező szempontjából az adatbázisok vizsgálatának a lényegét jelentik.

rekordoknak, mint ahogyan a 2.3. ábra mutatja. Az objektumoknak vannak mezői, amelyekben értékeket helyezhetünk el. Ezeknek az értékeknek lehet típusa, úgymint egész, karakterlánc vagy tömb, vagy hivatkozás egy másik objektumra. Sőt, még lehetnek metódusok is, azaz függvények, amelyeket alkalmazhatunk az objektumra. Itt azonban nem helyezünk hangsúlyt az ODL metódusaira, amelyek hasonlóak bármely más objektumorientált programozási nyelvben megszokotthoz. Az ODL metódusok használatához a 8.1. részben térünk vissza.

Bár gyakran segít, ha az objektumokra úgy tekintünk, mint rekordstruktúrára, ez a fejezet a tervezés egy absztrakt szintjét tárgyalja. Így, először egy osztálynak és a tulajdonságainak az absztraktabb fogalmával foglalkozunk, anélkül hogy meg vizsgálnánk a megvalósítás részleteit, például, ha az objektumot valójában egy rekordstruktúrával lehet reprezentálni, akkor sem különösebben fontos számunkra a rekord mezőinek sorrendje.

Amikor specifikálunk egy ODL osztályt, akkor a következő három fajta jellemzőjét adjuk meg:

1. *Attribútumok*, amelyek a tulajdonságok, és ezeknek a típusa egyszerű típusokból, mint egész, karakterlánc stb., épül fel. Speciálisan, egy attribútumnak olyan típusa van, amely nem foglal magában egy osztályt. Az ODL-ben az attribútumok típusai korlátozottak. Erről részletesebben a 2.1.7. részben beszélünk.

2. *Kapcsolatok*, amelyek szintén jellemzők, melyek típusa vagy valamely osztály egy objektumára való hivatkozás, vagy ilyen hivatkozások gyűjteménye (például halmaza).

3. *Metódusok*, amelyek függvények. Ezeket lehet alkalmazni az osztály objektumaira. Mint már korábban említettük, a metódusok használatát most nem fogjuk hangsúlyozni.

Az objektumazonosító természete

Mint már az 1.3.1. részben megjegyeztük, az objektumorientált adatbázisok gyakran olyan nagyok, hogy az objektumazonosítók száma túlnő a címtartományban lévő címek számán. Így az objektumorientált rendszerek általában valamilyen algoritmus alapján hoznak létre egy egyértelmű karakterláncot, amit hozzárendelnek minden egyes objektumhoz. Gyakran ezek a karakterláncok meglehetősen hosszúak, elérhetik a 16 bájtot is. Például, egy objektum azonosítóként kaphatja azt az időpontot, amikor létrehozták (elegendően kis egységben kell mérni, hogy egyedi gépen ne lehessen létrehozni két objektumot egy időben), hozzá véve annak az azonosítóját, aki létrehozta (arra az esetre, ha az adatbázisrendszer osztott).

2.1.2. Interfész-deklarációk

ODL-ben egy osztálynak a deklarációja a következőket tartalmazza:

1. Az `interface` kulcsszót.
2. Az interfész nevét (azaz az osztály nevét).
3. Kapcsos zárójelek között az osztály jellemzőinek a listáját. Ezek a jellemzők lehetnek attribútumok, kapcsolatok és metódusok.

Azaz egy egyszerű formája az interfész-deklarációnak:

```
interface <név> {
   <jellemzők listája>
}
```

2.1.3. Attribútumok az ODL-ben

A jellemzők legegyszerűbb fajtái az *attribútumok*. Ezek a tulajdonságok írják le egy objektum néhány aspektusát azáltal, hogy az objektumhoz valamilyen egyszerű típusú értéket rendelnek. Például, a személy objektumnak lehet egy attribútuma a `név`, amelynek a típusa karakterlánc és az értéke az adott személy neve. A személy objektumnak lehet egy másik attribútuma a `születésiDátum`, aminek a típusa egy egész számokból álló hármas (azaz rekordstruktúra), amely reprezentálja a születési évet, hónapot és napot.

2.1. példa: A 2.4. ábrán látható a filmek osztályának deklarációja. Ez a deklaráció nem teljes, de később meg fogjuk adni a teljes deklarációt. Az első sor deklarálja a `Film` osztályt. Az `interface` kulcsszó jelzi ODL-ben az osztályt[2]. A következő négy sor deklarálja a `Film` objektum 4 attribútumát.

Az első attribútum, a második sorban, a `cím`. A típusa karakterlánc, ami azt jelenti, hogy a hossza ismeretlen. Ez az attribútum jelenti az adott `Film` objektumban a filmnek a címét. A következő két attribútum, az `év` és a `hossz` a harmadik és negyedik sorokban, reprezentálják a film készítésének évét, illetve a film hosszát percekben. Az ötödik sorban lévő `szalagFajta` attribútum pedig azt mutatja, hogy az adott filmet színes vagy fekete-fehér szalagra rögzítették-e. Ennek a típusa egy *felsorolás* típus, amelynek a neve `Szalag`. A felsorolás típusú attribútumok az értékeiket *literálok*

[2] Technikailag az ODL-ben egy osztály az egy interfész az adatstruktúrái és metódusai megvalósításával együtt. Ebben a fejezetben nem tárgyaljuk az ODL interfészek megvalósítását, de ennek ellenére az interfész-deklarációkat úgy tekintjük, mint osztályok definiálását.


```
1) interface Film {
2)    attribute string cím;
3)    attribute integer év;
4)    attribute integer hossz;
5)    attribute enum Szalag {színes,feketeFehér} szalagFajta;
  };
```

2.4. ábra. *A `Film` osztály deklarációja ODL-ben*

egy listájából vehetik, mint a példában a `szalagFajta`, amely vagy `színes` vagy `feketeFehér` értéket vehet fel.

Egy `Film` osztálybeli objektum tekinthető úgy, mint egy rekord. Például,

```
("Elfújta a szél", 1939, 231, színes)
```

egy `Film` objektum. □

2.2. példa: A 2.1. példában az összes típus egyszerű típus volt. Használhatunk olyan attribútumokat is, amelyeknek a típusa struktúra vagy struktúrák valamilyen gyűjteménye. A típusokat részletesen a 2.1.7. részben tárgyaljuk. Most adunk egy példát, amelyben szerepel nem egyszerű típus is.

Definiáljuk a `Színészek` osztályát:

```
1) interface Színész {
2)    attribute string név;
3)    attribute Struct Cím
        {string város, string utca} lakcím;
  };
```

A második sor specifikálja a színész nevét, a harmadik 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 mezőket névvel és típussal. □

2.1.4. Kapcsolatok az ODL-ben

Amikor egy objektumot az attribútumok szempontjából vizsgálunk, időnként fontos lehet, hogy milyen módon kapcsolódik az objektum ugyanannak, vagy egy másik osztálynak az objektumaihoz.

2.3. példa: Tegyük fel, hogy szeretnénk a 2.1. példában lévő `Film` osztályba felvenni egy jellemzőt, ami színészek egy halmaza. Mivel a színészek maguk is egy osztályt alkotnak, lásd a 2.2. példában, így ez az információ nem lehet egy attribútuma a `Film`

osztálynak, ugyanis egy attribútum típusa nem lehet egy osztály, illetve egy osztályokból felépített típus. Így a filmek színészeinek halmaza egy *kapcsolat* 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 sor bárhová kerülhet a 2.4. ábrában a kapcsos zárójelek között. 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`. A `relationship` kulcsszó mutatja, hogy a `szereplők` egy hivatkozás más objektumokra, a `Set` kulcsszó pedig azt jelenti, hogy a `szereplők` a `Színész` osztály objektumainak egy halmazára hivatkozik és nem csak egy objektumra. Általában ODL-ben a `Set` kulcsszóval definiálható olyan típus, amely valamely T típus elemeinek egy halmazát jelenti úgy, hogy a T-t a kulcsszó után kisebb és nagyobb jelek közé tesszük.

Fizikailag úgy képzelhetjük el a `szereplők` halmazát, mint mutatók egy listáját, amelyek mindegyike egy `Színész` objektumra mutat. Tehát a `Színész` objektumok nincsenek fizikailag benne a `Film` objektumaiban. Habár az adatbázis tervezési fázisában a fizikai reprezentáció nem ismert, egy kapcsolat szempontjából mégis fontos, hogy egy `Film` objektumban a neki megfelelő szereplőket meg tudjuk találni. □

A 2.3. példában mutattunk egy olyan kapcsolatot, ami objektumoknak egy halmazát, a szereplőket rendeli hozzá egy objektumhoz, azaz egy filmhez. Lehetséges olyan kapcsolatot is deklarálni, amely egy objektumot rendel hozzá egy másik osztály egy objektumához. Például, a 2.3. példában elhagyva a `Set` kulcsszót a következőt kapjuk:

```
relationship Színész szereplője;
```

Ez a kapcsolat minden `Film` objektumhoz egy `Színész` objektumot rendel. Ez a megközelítés nyilván nem megfelelő, mert általában egy filmnek több szereplője van. Azonban fogunk még látni példákat, ahol az egyértékű kapcsolat valóban szükséges.

2.1.5. 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 2.2. példához. Azonban ez a sor és a hasonló deklaráció a `Film` osztályban nem veszi tekintetbe egy nagyon fontos aspektusát a filmek és a

színészek közötti kapcsolatnak. Ugyanis elvárjuk, 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.

2.4. példa: A 2.5. ábrán látható a `Színész` osztály teljes deklarációja, amelyben megadjuk a `szerepelBenne` kapcsolat inverz kapcsolatát is, azaz a `Film` osztály `szereplők` kapcsolatát. A negyedik sorban nem csak a `szerepelBenne` kapcsolat deklarációja van, hanem az inverze is: `Film::szereplők`. Mivel a `szereplők` kapcsolatot egy másik osztályban definiáltuk, így minősíteni kell az osztálya nevével, közéjük pedig két kettőspontot kell tenni. Ez a jelölés általános, amikor egy másik osztály egy tulajdonságára hivatkozunk. □

```
1) interface Színész {
2)    attribute string név;
3)    attribute Struct Cím
         {string város, string utca} lakcím;
4)    relationship Set<Film> szerepelBenne
         inverse Film::szereplők;
   };
```

2.5. ábra. *A `Színész` osztály egy kapcsolata és annak inverze*

A 2.4. példában a két inverz kapcsolat egy objektumhoz rendelt (film vagy színész objektumhoz), objektumok egy halmazát. Mint említettük lehetnek olyan kapcsolatok, amelyekben egy objektum kapcsolódik egy másik osztály egy objektumához. Az inverz kapcsolat fogalma ott is ugyanez. Általában mondhatjuk, egy C osztály R kapcsolata esetén, ha a C osztály egy x objektumában az R kapcsolat az $y_1, y_2, \ldots, y_n$ objektumokat jelenti, akkor minden y_i objektumnál R inverzkapcsolatában az x benne van. Néha segít elképzelni egy R kapcsolatot a C és D osztályok között, ha pároknak egy listájaként fogjuk fel. Minden pár egyik tagja egy C osztálybeli x objektum, a másik tagja pedig a hozzárendelt D osztálybeli y objektum, azaz:

C	D
x_1	y_1
x_2	y_2
...	...

Ha R típusa `Set<D>`, akkor minden C-beli objektum egynél több párban is benne lehet. Ha viszont R típusa `D`, akkor egy adott C-beli objektum csak egy párban szerepelhet.

Inverz kapcsolatok megkövetelése

Az ODL, mint egy absztrakt tervező nyelv, megköveteli, hogy a kapcsolatoknak legyen inverze. Ennek a követelménynek a magyarázata az, hogy bármikor, amikor létezik egy objektumhoz, például egy film objektumhoz, elérési út a hozzákapcsolt (színész) objektumok halmazába, akkor ez a kapcsolat visszafele is lehetséges, azaz a színésztől eljutunk azokhoz a filmekhez, amelyekben szerepelt. Azaz, ha adott egy színész, mondjuk Charlton Heston, akkor megvizsgálhatjuk az összes film objektumot, és ellenőrizhetjük a szereplőit, majd kiválaszthatjuk azokat, amelyeknek a szereplője volt. Ez azonban meg fog egyezni azokkal a filmekkel, amelyekben szerepelt. Az ODL megköveteli, hogy ennek a fordított műveletnek is legyen kapcsolat neve.

Azonban, amikor átfordítjuk az ODL deklarációt egy valódi programozási nyelv deklarációjára, például C++-ra, akkor megtehetjük, hogy csak a film objektumba teszünk hivatkozást és nem hivatkozunk a filmekre a színész objektumokban. Tehát a C++ megengedi az egyirányú kapcsolatokat. Mivel azonban itt a tervezésről van szó és nem a megvalósításról, elvárjuk, hogy a kapcsolatoknak legyen inverze.

Nyilván R inverz kapcsolata ugyanazon párok halmaza, csak a komponenseket meg kell fordítani, azaz:

D	C
y_1	x_1
y_2	x_2
...	...

Megjegyezzük, hogy ez a szabály akkor is működik, ha C és D ugyanazok az osztályok. Vannak olyan kapcsolatok, amelyek saját osztályukra hivatkoznak, mint például a „Személy" osztály „gyereke" kapcsolata hivatkozik magára, azaz a „Személy" osztályára.

2.1.6. Kapcsolattípusok

Gyakran lényeges, hogy egy kapcsolatban egy adott objektumhoz csak egy objektumot rendelünk, vagy kapcsolódhat hozzá több objektum is. Az ODL-ben ezeket az opciókat specifikálni tudjuk azzal, hogy használjuk vagy sem a gyűjtő operátorokat, mint a `Set`. Így, ha van egy inverz kapcsolatunk, akkor négy opció lehetséges: a kapcsolat valamelyik irányban egyelemű, mindkét irányban egyelemű vagy egyik irányban sem egyelemű.


```
1)  interface Film {
2)     attribute string cím;
3)     attribute integer év;
4)     attribute integer hossz;
5)     attribute enum Szalag { színes, feketeFehér} szalagFajta;
6)     relationship Set<Színész> szereplők
            inverse Színész::szerepelBenne;
7)     relationship Stúdió gyártó
            inverse Stúdió::gyárt;
    };

8)  interface Színész {
9)     attribute string név;
10)    attribute Struct Cím
          {string város, string utca} lakcím;
11)    relationship Set<Film> szerepelBenne
            inverse Film::szereplők;
    };

12)  interface Stúdió {
13)     attribute string név;
14)     attribute String cím;
15)     relationship Set<Film> gyárt
            inverse Film::gyártó;
    };
```

2.6. ábra. *Néhány ODL osztály és kapcsolataik*

A színészek és filmek közötti kapcsolat egyik irányban sem egyelemű, mint azt korábban láttuk. Azaz egy filmnek általában több szereplője van, és egy színész általában több filmben szerepelt. A következő példa az előzőek kidolgozása, illetve kiegészítése egy `Stúdió` osztállyal, ami a filmkészítő stúdiókat reprezentálja.

2.5. példa: A 2.6. ábrán megadtuk a `Film`, a `Színész` és a `Stúdió` osztályokat. Az első két osztályt már bevezettük a 2.1. és 2.2. példákban. A stúdió objektumoknak `név` és `cím` attribútumaik vannak, amelyek 13. és 14. sorban találhatók. Itt a címek típusa karakterlánc, ellentétben a `Színész` osztály `lakcím` attribútumával, amely a 10. sorban található. Természetesen az sem jelent semmilyen problémát, amikor azonos nevű attribútumokat használunk, de különböző típusúak és különböző osztályban vannak.

A hetedik sorban látható a `gyártó` kapcsolat a filmek és stúdiók között. Mivel a kapcsolat típusa `Stúdió` és nem `Set<Stúdió>`, így minden filmet csak egy stúdió gyártott. Ennek az inverz kapcsolata a `gyárt` kapcsolat. Ennek a típusa `Set<Film>`, ami azt jelenti, hogy minden stúdió filmek halmazát gyártotta, 0-t, 1-t, de lehet, hogy sokkal többet. □

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.

Megjegyezzük azt is, hogy ha azt mondjuk, hogy az *R* kapcsolat sok-sok típusú, akkor az valójában azt jelenti, hogy *R*-nek lehetősége van sok-sok kapcsolatúnak lenni. Azaz, ahogy *R* folyamatosan változik, van amikor éppen sok-egy kapcsolat, vagy van amikor éppen egy-egy kapcsolat. Hasonlóan, egy *R* sok-egy kapcsolat is lehet valamikor éppen egy-egy kapcsolat.

Egy kapcsolat és az inverze közötti számossági megfeleltetést a kapcsolat *kapcsolattípusának* nevezzük. A következő három kapcsolattípus létezik:

1. A *sok-sok* kapcsolat[3], amely egy *C* és egy *D* osztály esetén azt jelenti, hogy minden *C*-beli objektumhoz *D*-beli objektumok egy halmazát rendeljük hozzá, illetve az inverz kapcsolatban, minden *D*-beli objektumhoz *C*-beli objektumok egy halmazát rendeljük. Például a 2.6. ábrán a `szereplők` és `szerepelBenne` sok-sok kapcsolatok a `Film` és `Színész` osztályok között. Itt mindkét irányban megengedett az üres halmaz is, ami azt jelenti, hogy lehetnek filmek, amelyekben nincs szereplő.

2. A *sok-egy* kapcsolat[4], amely egy *C* és egy *D* osztály esetén azt jelenti, hogy minden *C*-beli objektumhoz pontosan egy *D*-beli objektumot rendelünk, viszont minden *D*-beli objektumhoz *C*-beli objektumok egy halmazát rendeljük. A 2.6. ábrán a `gyártó` egy sok-egy kapcsolat. Az ilyen kapcsolatok inverzét *egy-sok* kapcsolatnak nevezzük. Tehát a 2.6. ábrán a `gyárt` egy-sok kapcsolat.

3. Az *egy-egy* kapcsolat[5], amely egy *C* és egy *D* osztály esetén azt jelenti, hogy minden *C*-beli objektumhoz pontosan egy *D*-beli objektumot rendelünk, és minden *D*-beli objektumhoz is pontosan egy *C*-beli objektum tartozik. Például tegyük fel, hogy a 2.6. ábrát kiegészítjük egy `Elnök` osztállyal, amely a stúdiók elnökeit tar

[3] A fordító megjegyzése: A magyar irodalomban ezt N:N kapcsolatként találjuk meg.
[4] A fordító megjegyzése: Ezt a magyar irodalomban N:1 kapcsolatnak nevezik.
[5] A fordító megjegyzése: Az egy-egy kapcsolatnak megfelel az 1:1 jelölés a magyar irodalomban.

talmazza. Elvárjuk, hogy minden stúdiónak egy elnöke legyen és minden elnök csak egy stúdiónak legyen az elnöke. Így ez a kapcsolat mindkét irányban egy-egy kapcsolat.

Van egy apró különbség a „pontosan egy" és az „egy" között a sok-egy és egy-egy kapcsolatokban. Normálisan azt várjuk, hogy a „pontosan egy" vagy „egy" elem létezzen. Például, minden filmhez valójában van egy gyártó stúdió és minden stúdiónak van elnöke. Azonban a gyakorlatban vannak okok, amiért nem létezik a várt „pontosan egy" objektum. Például a stúdió ideiglenesen lehet elnök nélkül, vagy egy bizonyos filmnél lehet ismeretlen, hogy melyik stúdió készítette.

Így megengedett, hogy bizonyos kapcsolt objektumok hiányozzanak. Később látni fogjuk, hogy az adatbázisokban a „null" érték gyakran megengedett, ahol tulajdonképpen valódi értéknek kellene lenni. Például a hagyományos programozásban is találhatunk „nil" mutatót, ahol egy objektumra mutató mutatónak kellene lennie. A 2.5. részben fogjuk tárgyalni az épségi megszorításokat, és látunk egy mechanizmust arra, hogyan kezeljük az olyan objektumokat, amelyeknek feltétel nélkül létezni kell.

2.1.7. Típusok az ODL-ben

Az ODL egy típusrendszert ajánl fel az adatbázis-tervezőnek, ami hasonló ahhoz, ami a C++-ban vagy más hagyományos programozási nyelvben található. Egy típusrendszer alaptípusokból épül fel úgy, hogy adottak az alaptípusok, és bizonyos rekurzív szabályokkal összetett típusok építhetők egyszerűbb típusokból. ODL-ben az alaptípusok a következők:

1. *Atomi típusok:* `integer`, `float`, `character`, `string`, `boolean` és `enum`. Ez utóbbi a felsorolás típus, amelyben neveknek egy listáját definiáljuk és ezeknek a neveknek, majd egész számok fognak megfelelni. Például, a 2.6. ábra 5. sorában használunk egy felsorolás típust, ahol a `színes` és `feketeFehér` neveket definiáljuk, és ezeknek a 0 és 1 egész számok felelnek meg.

2. *Interfész típusok:* olyanok, mint a `Film` vagy a `Színész` osztályok, amelyek lényegében struktúrákat reprezentáló típusok. Ezeknek a struktúráknak a komponensei felelnek meg az interfész attribútumainak és kapcsolatainak Az interfész típusok nevei az alábbi szabályok segítségével megadott összetett típust reprezentálnak, de tekinthetjük őket alaptípusoknak is.

A következő *típuskonstruáló* utasításokkal a fenti alaptípusok kombinálhatók strukturált típusokban:

1. *Halmaz.* Ha *T* egy típus, akkor `Set<T>` jelöli azt a típust, amelynek az értéke *T* típusú elemek egy véges halmaza. Például ezt használjuk a 2.6. ábra 6., 11. és 15. sorában.

Halmazok, multihalmazok és listák

A halmazok, multihalmazok és listák között következő a különbség. Egy halmaz mindig 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.

2. *Multihalmaz.* Ha T egy típus, akkor `Bag<T>` jelöli azt a típust, amelynek az értéke T típusú elemek multihalmaza. A multihalmaz megengedi, hogy egy elem többször is szerepeljen benne. Például, {1, 2, 1} egy multihalmaz, de nem halmaz, mert az 1 több, mint egyszer szerepel benne.

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. Például a `string` típus egy rövid jelölése a `List<char>` típusnak.

4. *Tömb.* Ha T egy típus és i egy egész szám, akkor `Array<T, i>` jelöli azt a típust, amelynek az értéke egy i elemű tömb, ahol az elemek típusa T. Például `Array<char, 10>` egy 10 hosszú karakterláncot jelent.

5. *Struktúra.* Ha $T_1, T_2, \ldots, T_n$ típusok és $F_1, F_2, \ldots, F_n$ mezőnevek, akkor

   `Struct N { T1 F1, T2 F2, ..., Tn Fn}`

   jelöli azt az N nevű típust, amely egy n mezőből álló struktúra. Az i mező neve F_i és típusa T_i. Például a 2.6. ábra 10. sorában használtuk a `Cím` struktúrát két mezővel. Mindkét mező típusa `string` és a mezők nevei rendre `város` és `utca`.

Az első négy típust, halmaz, multihalmaz, lista és tömb, *kollekciótípusoknak* nevezzük. Különböző szabályok vannak arra, hogy mely típusok használhatók attribútumoknál és mely típusok használhatók kapcsolatok esetében.

- Egy attribútum típusa egy olyan típus, ami egy atomi típusból vagy egy olyan struktúrából, amely mezőinek a típusa atomi típus, kiindulva épül fel. Az atomi típusra vagy a struktúrára utána valamelyik kollekciótípust vagy struktúraképzést alkalmazhatjuk.

- Egy kapcsolat típusa vagy egy interfész típus vagy egy kollekciótípus alkalmazva egy interfész típusra.

Fontos megjegyezni, hogy az interfész típusok nem szerepelhetnek egy attribútum típusában és az atomi típusok nem szerepelhetnek egy kapcsolat típusában. Ez a különbség szétválasztja az attribútumokat és a kapcsolatokat. Azt is megjegyezzük, hogy különbség van az összetett típusok felépítési módjában is az attribútumoknál és a kapcsolatoknál. Mindkét esetben megengedett a kollekciótípusok használata, mint végső operátor, azonban csak az attribútumok esetében lehet a struktúratípus lezáró operátor.

2.6. példa: Attribútumok lehetséges típusai:

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 kollekciótípussal, a negyedik pedig atomi típusok struktúrája kollekciótípussal. Attribútumok típusa csak ez a négyféle lehet.

Tegyük fel, hogy a `Film` és a `Színész` interfész típusaink adottak. Ekkor konstruálhatunk a kapcsolatoknak olyan típusokat, mint `Film` vagy `Bag<Színész>`. 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.

3. `Set<Array<Színész, 5>>`. Egy kapcsolat típusa nem tartalmazhat két kollekciótípust (sőt ez az attribútumok típusaira is igaz).

□

2.1.8. Feladatok

* **2.1.1. feladat:** Tervezzünk egy bank részére adatbázist, amely tartalmazza az ügyfeleket és azok számláit. Az ügyfelekről tartsuk nyilván a nevét, címét, telefonszámát és TAJ-számát. A számláknak legyen számlaszámuk, típusuk (pl. takarékbetét-számla, folyószámla stb.) és egyenlegük. Továbbá, meg kell jelölni azokat az ügyfeleket, akiknek van számlájuk. Adjuk meg az ODL leírását ennek az adatbázisnak. Gondosan válasszuk meg az alkalmas típusokat az attribútumok és kapcsolatok részére.

2.1.2. feladat: Módosítsuk a 2.1.1. feladat tervét a következő módon. Írjuk le a változásokat úgy, hogy nem készítünk teljesen új tervet.

a) Egy számlának csak egy tulajdonosa lehet.

b) Az a)-t egészítsük ki azzal, hogy egy ügyfélnek csak egy számlája lehet.

c) A cím három részből áll: ország, város és utca. Ráadásul egy ügyfélnek akárhány címe és telefonszáma lehet.

! d) Az ügyfeleknek akárhány címe lehet, amelyek a c)-ben leírt hármasok, és minden címhez több telefonszám is tartozhat. Azaz meg kell jegyezni egy ügyfél címeinél, hogy a címekhez mely telefonszámok tartoznak. *Megjegyzés:* vigyázzunk arra, hogy az attribútumok és kapcsolatok típusaira korlátozások vannak.

2.1.3. feladat: Adjuk meg az ODL tervét 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.

2.1.4. feladat: Módosítsuk a 2.1.3. feladatot úgy, hogy a játékosokról jegyezzük fel, hogy korábban mely csapatokban játszottak, beleértve a belépés és kilépés dátumát is.

*! **2.1.5. feladat:** Tegyük fel, hogy családfát szeretnénk összeállítani. Csak egy osztályunk van, a `Személy` osztály. 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? Írjuk le a kapcsolatokat és inverzeiket párok halmazaként.

! **2.1.6. feladat:** A 2.1.5. feladat tervét egészítsük ki a `végzettség` attribútummal. Ennek az attribútumnak az értéke a megszerzett végzettségek egy gyűjteménye. Minden végzettséghez tartozik egy név (a megszerzett fokozat neve), iskola, ahol megszerezte és 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?


```
1) interface Hajó {
2)    attribute string név;
3)    attribute int felavatáséve;
4)    relationship Különítmény hozzárendelt
         inverse Különítmény::része;
   };

5) interface Különítmény {
6)    attribute real szám;
7)    attribute string parancsnok;
8)    relationship Set<Hajó> része
         inverse Hajó::hozzárendelt;
   };
```

2.7. ábra. *A hajók és a különítmények leírása ODL-ben*

! **2.1.7. 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, ami 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 megválasztásról, illetve arról, hogy milyen segédinformációkat használunk.

2.1.8. feladat: A 2.7. ábrán van egy ODL terv a `Hajó` és `Különítmény` osztályokhoz (a *különítmény* hajók egy gyűjteménye). Módosítani szeretnénk a fenti definíciót. Minden módosítás végrehajtható úgy, hogy egy vagy több sort megváltoztatunk vagy beszúrunk a definícióba. Hajtsuk végre a következő módosításokat:

a) A `parancsnok` attribútumnak változtassuk meg a típusát karakterláncokból álló párokra úgy, hogy tároljuk a rangot és a nevet is.

b) Engedjük meg, hogy egy hajó több különítménynek is tagja lehessen.

c) *Testvérhajó* az a hajó, amely ugyanazon tervek szerint készült. Tartsuk nyilván a testvérhajókat. Minden hajónak több testvérhajója van és minden testvérhajó egy `Hajó` objektum.

*!! **2.1.9. feladat:** Milyen körülmények között lehet egy kapcsolat önmaga inverze? *Útmutatás:* Gondoljunk a kapcsolatokra, mint párok halmazára, ahogy 2.1.5. részben láttuk.

*! **2.1.10. feladat:** Van-e olyan típus, amely lehet típusa egy ODL attribútumnak és egy ODL kapcsolatnak is? Magyarázzuk meg miért.

2.2. Egyed-kapcsolat (E/K) diagramok

Az adatmodellezés grafikus megközelítését *egyed-kapcsolat diagramnak* nevezzük, amely lényegében hasonlít az ODL objektumorientált módszeréhez. Az E/K diagramnak ugyanaz a három alapeleme van, mint az ODL-nek (de vannak mindkettőben további elemek, amelyeket később tárgyalunk). Ez a három elem a következő:

1. *Egyedhalmazok,* amelyek az osztályok megfelelői. Egyedhalmazok elemei az egyedek, amelyek az ODL objektumoknak felelnek meg.

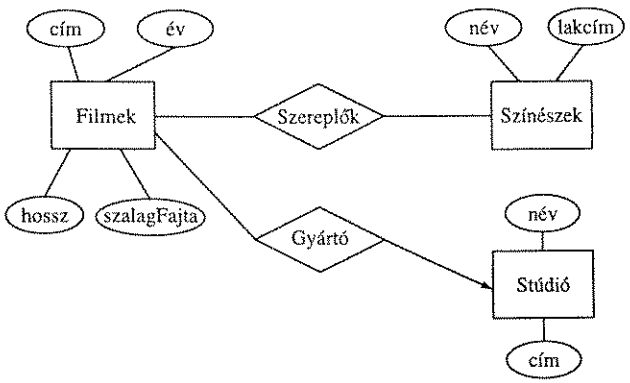
2. *Attribútumok,* amelyek értékei egy egyed tulajdonságait írják le. Így az attribútumok fogalma lényegében megegyezik az E/K diagramok és az ODL esetében.

3. *Kapcsolatok,* amelyek két vagy több egyedhalmazt kapcsolnak össze. Az E/K diagrambeli kapcsolatok meglehetősen hasonlóak az ODL-ben lévő kapcsolatokhoz. Azonban:

   a) Az E/K modellben a kapcsolatok kétirányúak, amíg az ODL-ben egy kapcsolatot és inverzét külön kell specifikálni. Például, a 2.6. ábrán lévő `Film::szereplők` és `Színész::szerepelBenne` inverz kapcsolat egy E/K diagramon egyetlen kapcsolatként jelenik meg.

   b) Egy kapcsolat E/K modellben több egyedhalmazt is összeköthet, míg ODL-ben a kapcsolatok legfeljebb két osztály között lehetnek.

2.7. példa: A 2.8. ábrán lévő E/K diagram ugyanazt modellezi, mint a 2.6. ábrán lévő ODL leírás. Az egyedhalmazok a *Filmek, Színészek* és *Stúdiók.* Az egyedhalmazok



2.8. ábra. *A film adatbázis egyed-kapcsolat diagramja*

nevét többes számban fogjuk használni, amíg az osztályok neve általában egyes számban van. Ez okoz egy kis különbséget a két ábra között.

A *Filmek* egyedhalmaznak ugyanaz a négy attribútuma van, mint a `Film` osztálynak a 2.6. ábrán: *cím, év, hossz* és *szalagFajta.* Hasonlóan a másik két egyedhalmaznak is ugyanazok az attribútumai, mint megfelelő ODL osztályoknak.

A 2.8. ábrán látható az E/K diagram kapcsolatainak és 2.6. ábrán lévő ODL kapcsolatoknak a viszonya is. A *Szereplők* kapcsolat magába foglalja a `szereplők` és `szerepelBenne` ODL inverz kapcsolatok információit. Továbbá a 2.8. ábra E/K diagramjában lévő *Gyártó* reprezentálja a `Film::gyártó` és `Stúdió::gyárt` ODL inverz kapcsolatot. A nyíl, ami a *Stúdiók* egyedhalmazba mutat, a 2.8. ábrán azt jelenti, hogy minden filmet egy stúdió gyártott. Az E/K diagram kapcsolatainak típusait a következőkben fogjuk tárgyalni. □

2.2.1. E/K kapcsolatok típusai

Mint a 2.7. példában már láttuk, 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 pontosan egy *F*-beli egyed kapcsolódik. Azonban egy *F*-beli egyedhez sok *E* egyed kapcsolódhat.

E/K kapcsolatok megjelenítése

Gyakran segít, ha az E/K kapcsolatokat táblázatként vagy relációként fogjuk fel. Minden sor összetartozó párokat reprezentál. Például a *Szereplők* kapcsolat a következőképpen jeleníthető meg:

Filmek	*Színészek*
Elemi ösztön	Sharon Stone
Emlékmás	Arnold Schwarzenegger
Emlékmás	Sharon Stone

Természetesen nincs külön megadva, hogy a kapcsolatot ODL-ben vagy E/K modellben hogyan kell implementálni.

A fenti táblázatot néha a kapcsolat *kapcsolathalmazának* is nevezzük. A kapcsolathalmaz elemei a táblázat sorai. A sorok pedig a kapcsolódó egyedeket reprezentálják úgy, hogy a kapcsolatban részt vevő minden egyedhalmazból tartalmaz egy elemet. Például

(Elemi ösztön, Sharon Stone)

egy sor a *Szereplők* kapcsolat kapcsolathalmazában.

Ha egy-egy kapcsolat van E és F egyedhalmazok között, akkor nyíl mutat mindkét egyedhalmazba. Például a 2.9. ábrán látható két egyedhalmaz a *Stúdiók* és *Elnökök*, illetve a *Vezető* kapcsolat (az attribútumokat elhagytuk). Feltételezhetjük, hogy egy elnök csak egy stúdiót vezet és egy stúdiónak csak egy elnöke van, így ez egy-egy kapcsolat, amit a kétirányú nyíl jelez.

2.9. ábra. *Példa egy-egy kapcsolatra*

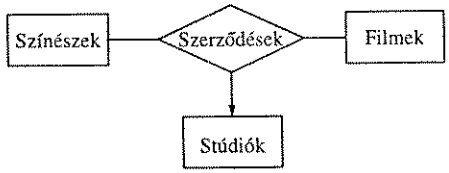
2.2.2. Sokágú kapcsolatok

Az E/K modellben, az ODL leírással ellentétben, lehet olyan kapcsolatokat definiálni, amelyek több egyedhalmazt kapcsolnak össze. A gyakorlatban a hármas és magasabb fokú kapcsolatok ritkák. A sokágú kapcsolatokat E/K diagramban a kapcsolatot reprezentáló rombuszból kiinduló és a kapcsolatban részt vevő egyedekhez vivő vonalakkal jelöljük.

2.8. példa: A 2.10. ábrán a *Szerződések* 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 láttuk az „E/K kapcsolatok megjelenítése" című bekeretezett részben, egy E/K kapcsolat tekinthető egy kapcsolathalmaznak, amelynek az elemei résztvevő egyedhalmazok egyedeiből képzett n-esek. Így a *Szerződések* 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 pontosan egy egyed tartozik az E egyedhalmazból. (Megjegyezzük, hogy ez a definíció egy egyszerű általánosítása a kétágú kapcsolatoknál használtnak.) A 2.10. ábrán a *Stúdiók* egyedhalmazra mutat egy nyíl, ami azt jelenti, hogy csak egy stúdió létezik, amely egy színésszel egy adott filmre szerződést



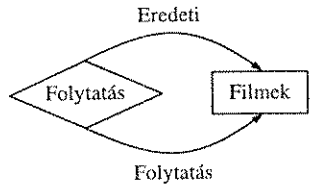
2.10. ábra. *Egy háromágú kapcsolat*

kötött. Azonban nem mutat nyíl a *Színészek* egyedhalmazra vagy a *Filmek* egyedhalmazra sem. Azaz egy stúdió szerződést köthet több színésszel is egy filmre és egy színészszel is köthet szerződést több filmre is. □

2.2.3. 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 egyedhalmaznak egy másik *szerepét* mutatja a kapcsolatban. Ezért meg kell címkézni az éleket, amelyek a kapcsolat és az egyedhalmaz között vannak, és ezeket nevezzük „szerepeknek".

2.9. példa: A 2.11. ábrán a *Folytatás* kapcsolat a *Filmek* egyedhalmaznak önmagával való kapcsolata. Két film között lehet kapcsolat úgy, hogy az egyik folytatása a másiknak.



2.11. ábra. *Egy kapcsolat szerepekkel*

Két filmet különböztet meg a kapcsolat, 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 2.11. ábrán egy sok-egy kapcsolatot. □

Korlátozások a nyíl jelölésre a sokágú kapcsolatokban

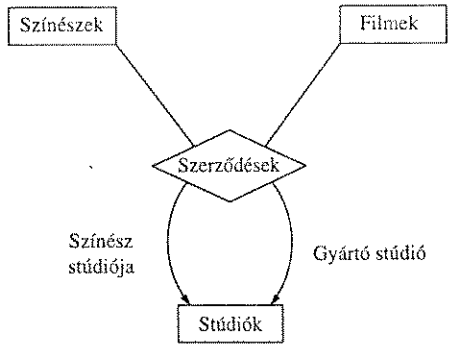
Nincs elegendő lehetőség a nyíl és nem nyíl választásban a három vagy több résztvevős kapcsolatok esetében. Így nem tudjuk leírni az összes lehetőséget nyilakkal. Például, a 2.10. ábrán a stúdió valójában csak a filmnek függvénye és nem a színésznek és a filmnek együtt, mivel csak egy stúdió gyárt egy 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. A 3.5. részben tárgyalunk egy formális jelölést, a funkcionális függőségeket, amely képes leírni az összes alternatívát.

2.10. példa: Záró példánkban egyaránt szerepel sokágú kapcsolat, és egy egyedhalmaz több szerepben. A 2.12. ábrán a 2.8. példában bevezetett *Szerződések* kapcsolat egy bonyolultabb verziója látható. Most a *Szerződések* kapcsolatban kétszer szerepel a stúdió, egyszer a színész és egyszer a film. Ez azt reprezentálja, hogy egy stúdió, amelyiknek van egy színésszel szerződése, 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.

A 2.12. 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. Adott egy színész, egy film és egy stúdió, amely a filmet készíti, akkor csak egy stúdió lehet, aki 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.



2.12. ábra. *Egy négyágú kapcsolat*

Tehát a *Színészek* és a *Filmek* egyedhalmazokra nem mutatnak nyilak. 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. □

2.2.4. Kapcsolatok attribútumai

Néha kényelmes, ha attribútumokat rendelhetünk egy kapcsolathoz és nem kell egy új egyedhalmazt létesíteni az összes kapcsolatával. Például vizsgáljuk meg a 2.10. ábrát, amelyen modelleztük a színészek és a stúdiók szerződéseit valamilyen filmre. 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 ez a stúdióhoz sem rendelhető (különböző színészeknek valószínűleg különböző összeget fizet) és a filmhez sem rendelhető (különböző színészek egy adott filmben is különböző fizetést kaphatnak).

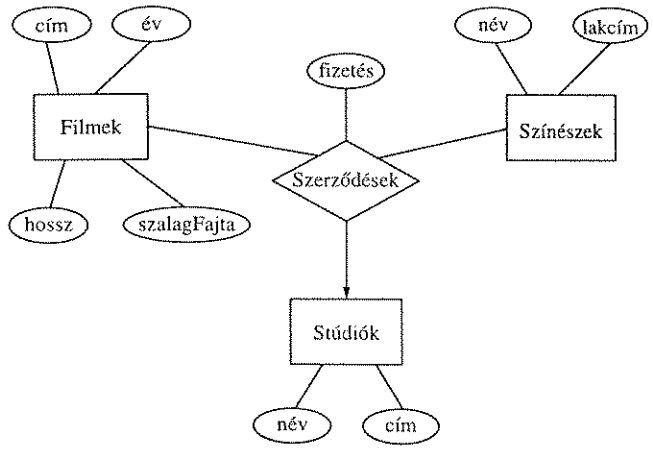
Azonban, ha a fizetést a

(színész, film, stúdió)

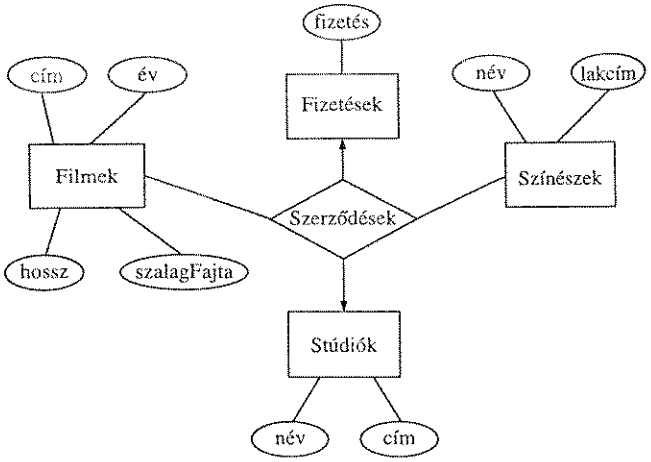
hármashoz rendeljük a *Szerződések* kapcsolat kapcsolathalmazában, akkor az éppen jó lesz. A 2.13. ábra ugyanaz, mint a 2.10. ábra kiegészítve attribútumokkal. A kapcsolatnak van *fizetés* attribútuma és az egyedhalmazoknak pedig ugyanazok az attribútumai, mint a 2.8. ábrán.

Nem szükséges a kapcsolatokhoz attribútumokat rendelni. Helyette felvehetünk egy új egyedhalmazt, amely egyedeinek az attribútumai éppen a kapcsolathoz rendelt attribútumok. Ha ezt az egyedhalmazt is belevesszük a kapcsolatba, akkor elhagyhatjuk a kapcsolat attribútumait.

2.11. példa: Tekintsük a 2.13. ábra E/K diagramját, amelyen a fizetés a *Szerződések* kapcsolat attribútuma. Hozzunk létre egy *Fizetések* egyedhalmazt *fizetés* attribútum-



2.13. ábra. *Egy kapcsolat attribútummal*



2.14. ábra. *Kapcsolat attribútumának megszüntetése egy új egyeddel*

mal. A *Fizetések* lesz a negyedik egyedhalmaz a *Szerződések* kapcsolatban. A teljes E/K diagram a 2.14. ábrán látható. □

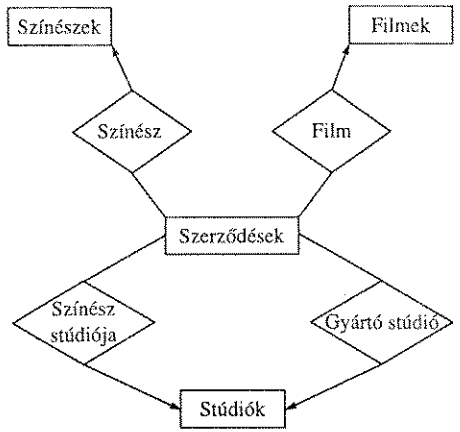
2.2.5. Sokágú kapcsolatok átalakítása binárissá

Az ODL, ellentétben az E/K modellel, csak bináris kapcsolatokat enged meg. Ez a korlátozás nem csökkenti az ODL képességeit, mert bármely sokágú kapcsolat információveszteség nélkül konvertálható sok-egy típusú bináris kapcsolatok egy gyűjteményére. E/K modellben bevezethetünk egy új egyedhalmazt, amelynek az egyedei tulajdonképpen a sokágú kapcsolat kapcsolathalmazának elemei. Ezt az egyedhalmazt *kapcsoló egyedhalmaznak* 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.

2.12. példa: A 2.12. ábrán lévő négyágú *Szerződések* kapcsolatot kicserélhetjük egy ugyanolyan nevű *Szerződések* egyedhalmazra. Mint a 2.15. ábrán látható, négy kapcsolatban vesz részt. Ha a *Szerződések* 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 fenti *film*



2.15. ábra. *Egy sokágú kapcsolat kiváltása egy egyedhalmazzal és bináris kapcsolatokkal*

egyedét. A *Stúdiók* egyedhalmaz *stúdió1* és *stúdió2* egyedét pedig *Színész stúdiója* és a *Gyártó stúdió* kapcsolatokon keresztül azonosítja.

Megjegyezzük, hogy feltettük, hogy a *Szerződések* egyedhalmaznak nincs attribútuma, a többi egyedhalmaz attribútumait pedig a 2.15. ábrán nem tüntettük fel. Természetesen vehetnénk fel attribútumokat a *Szerződések* egyedhalmazhoz mint, az aláírás dátuma stb. □

ODL-ben is tudunk reprezentálni sokágú kapcsolatokat, mint például, ami a 2.12. ábrán látható, a fent leírt E/K modellbeli transzformációhoz hasonlóan. Azonban, mivel ODL-ben nincs sokágú kapcsolat, így a transzformáció nem opcionális, azaz mindig el kell készíteni.

2.13. példa: Tegyük fel, hogy van három osztályunk a 2.12. ábrán lévő három egyedhalmaznak megfelelően, mégpedig `Színész`, `Film` és `Stúdió` osztályok. A négyágú *Szerződések* kapcsolathoz vezessük be a `Szerződés` osztályt. Ennek az osztálynak nincs attribútuma, viszont van négy kapcsolata az E/K kapcsolat négy komponensének megfelelően. Az ODL leírás a 2.16. ábrán látható inverz kapcsolatok nélkül. A *Szerződések* E/K kapcsolat minden négyesének megfelel egy `Szerződés` osztálybeli objektum az ODL-ben. □

```
interface Szerződés {
   relationship Stúdió színészStúdiója;
   relationship Stúdió gyártóStúdió;
   relationship Színész színész;
   relationship Film film;
};
```

2.16. ábra. *A szerződések leírása ODL-ben*

2.2.6. Feladatok

* **2.2.1. feladat:** A 2.1.1. feladatban leírt banki adatbázist írjuk át E/K modellbe. Használjunk nyilakat a kapcsolattípusok jelölésére.

2.2.2. feladat: Módosítsuk a 2.2.1. feladatra adott megoldásunkat a 2.1.2 feladat javaslatai alapján:

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 2.2.1. 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 kollekciótípusa legyen, ellentétben az ODL-lel, ahol bizonyos korlátozással lehet őket használni.

! d) Módosítsuk tovább a diagramunkat úgy, hogy minden ügyfélhez tartozhasson lakcímek egy halmaza és minden egyes lakcímhez telefonszámoknak egy halmaza.

2.2.3. feladat: Adjuk meg a 2.1.3. feladatban specifikált csapatok/játékosok/szurkolók adatbázis E/K diagramját. Vigyázzunk, a színek halmaza nem lehet a csapatok egy attribútumának típusa. Hogyan lehet feloldani ezt a megszorítást?

! **2.2.4. feladat:** Tegyük fel, hogy szeretnénk hozzávenni a 2.2.3. 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.

2.2.5. feladat: Módosítsuk a 2.2.3. feladat E/K diagramját a 2.1.4. feladat alapján a játékosok történetével.

! **2.2.6. feladat:** Adjuk meg a 2.1.5. és 2.1.6. feladatokban specifikált személyek adatbázisának E/K diagramját. Legyenek benne az anya, az apa és a gyerekek kapcsolatok is. Ne felejtsük el, hogy ha egy egyedhalmaz többször vesz részt egy kapcsolatban, akkor különböző szerepeket jelölnünk kell. Tartalmazza az adatbázis minden személy végzettségét is mint, ahogyan a 2.1.6. feladatban specifikáltuk. Minden kapcsolatnak jelöljük a típusát is. Szükséges külön választani az anya, apa és gyerekek kapcsolatokat? Miért?

2.2.7. feladat: Egy megoldása lehet a 2.1.5. feladatban meghatározott információk reprezentálásának a *Család* hármas kapcsolat. Ez a kapcsolat

(személy, anya, apa)

hármasok halmaza, ahol egy személy szüleit tároljuk. Természetesen minden adat a *Személyek* egyedhalmazból kerül ki.

* a) Rajzoljuk meg ezt a diagramot (a végzettségre vonatkozó információk nélkül). Helyezzük el a nyilakat a megfelelő helyekre.

b) Cseréljük ki a *Család* kapcsolatot egy egyedhalmazra és bináris kapcsolatokra. Ismét helyezzük el a nyilakat a kapcsolatok típusának megfelelően.

2.2.8. feladat: Írjuk át a 2.1.7. feladatban specifikált egyetemi adatbázis tervét E/K diagramra.

2.3. Tervezési alapelvek

Még nagyon sok részletet kell megismernünk az ODL és E/K modellekkel kapcsolatban, de már eleget tudunk ahhoz, hogy elkezdjük megvizsgálni a lényeges pontjait annak, hogy mitől jó egy terv és mit kell elkerülni. Ebben a részben megpróbálunk kimondani és kidolgozni néhány hasznos elvet.

2.3.1. Valósághű modellezés

Az első és legfontosabb, hogy a tervnek pontosan meg kell felelni a specifikációnak. Azaz, az osztályoknak vagy egyedhalmazoknak és attribútumaiknak tükrözniük kell a valóságot. Nem lehet a `Színész` osztálynak attribútuma a `hengerekSzáma`, hiszen ez inkább az `Autó` osztályhoz tartozik. Akármilyen összefüggések jönnek létre,

Redundancia és az inverz kapcsolatok

Azt gondolhatjuk, hogy az ODL-ben használt kapcsolat és inverze a redundáns tervezés példái. Azonban nem tételezhetjük fel, hogy a kapcsolatot és az inverzét két különböző adatstruktúra (mint például az egyik, illetve a másik irányba mutató mutatók) által reprezentálja. Idézzük fel a kapcsolatok és inverzeik definícióját, ami azt mondja, hogy egy kapcsolatot elvben mindkét irány együtt határoz meg.

Ha a kapcsolat megvalósításakor mégis két szeparált adatstruktúrát választanánk, akkor vállalnánk a redundancia kockázatát. Mivel a mutatóktól elvárható, hogy konzisztensen karbantartottak legyenek, mint egyéb adatváltozások, így az ODL alapú adatbázis-kezelő rendszer megvalósításánál nagyon óvatosnak kell lenni, hogy hogyan hajtjuk végre az adatbázis-változtatásokat. Azonban ez egy rendszer szintű dolog, és feltételezhetjük, hogy a megvalósítás ezt jól hajtja végre. Így kisebb a redundancia kockázata a megvalósítás szintjén, és a mutatók mindkét irányú létezése nagyobb sebességbeli hatékonyságot eredményezhet.

azoknak értelmeseknek kell lenni az alapján, amit ismerünk a valós világ modellezendő részéről.

2.14. példa: Ha definiálunk egy `szerepelBenne` kapcsolatot a `Színész` és a `Film` 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. □

2.3.2. Redundancia elkerülése

Óvatosnak kell lennünk, hogy minden csak egyszer szerepeljen. Például használtunk egy *Gyárt* 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. Ugyanazon gyártó stúdió esetében a kétszeri előfordulás több helyet igényel, mint az egyszeri.

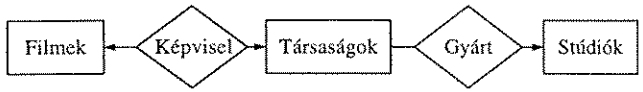
2. Ha a filmnek megváltozik a gyártója, akkor lehet, hogy a gyártó stúdiót kicseréljük a *Gyárt* kapcsolatnál, de elfelejtjük kicserélni a *stúdióNév* attribútumot vagy fordítva. Természetesen lehetne úgy érvelni, hogy nem szabad ilyen meggondolatlanságot elkövetni, de a gyakorlatban a hibázás nem ritka dolog. Ha ugyanazt a dolgot két különböző módon reprezentáljuk, az mindig tartalmazza a hiba lehetőségét.

Ezeket a problémákat a 3.7. részben formálisabban írjuk le, és megismerjük azokat az eszközöket, amelyekkel átalakíthatjuk az adatbázissémát úgy, hogy a redundanciát és a vele járó problémákat elkerüljük.

2.3.3. Egyszerűség

Ne vegyünk fel több elemet az adatbázistervünkbe mint amennyi szükséges.

2.15. 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. Ez egy-egy kapcsolatba 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 2.17. ábra mutatja.



2.17. ábra. *Egy terv szükségtelen egyedhalmazzal*

Tulajdonképpen a 2.17. 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 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. □

2.3.4. A megfelelő 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, hogy attribútumot vagy osztályt/egyedhalmazt használjunk-e. Általában egy attribútumot egyszerűbb implementálni, mint egy osztályt/egyedhalmazt vagy kapcsolatot. Azonban, ha mindent attribútumnak választunk, akkor abból rendszerint problémák származnak.

2.16. példa: Vizsgáljunk meg egy jellegzetes problémát. A 2.6. és 2.8. ábrákon jól tettük-e, hogy a stúdiót egy osztálynak vagy egy egyedhalmaznak választottuk? Lehetnének a stúdiónevek és címek a filmek attribútumai, és így a stúdiókat elhagyhatnánk? Egyik probléma, hogy a stúdiók címét minden filmnél meg kell ismételni. Ez redundanciához vezet, melynek az előnytelenségeit a 2.3.2. részben tárgyaltuk. Sőt az-

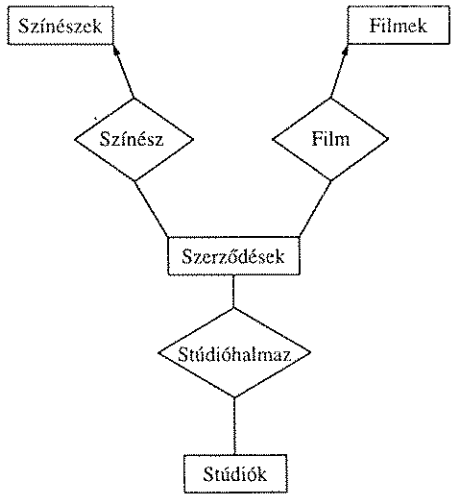
zal a kockázattal is szembe kerülünk, hogy elveszítjük egy stúdiónak a címét azáltal, hogy a stúdió nem tartozik egyetlen filmhez sem.

Ha azonban nem tároljuk a stúdiók címét, akkor már nem káros a stúdiók nevéből filmeknek egy attribútumát készíteni. Ekkor nem lesz redundancia a címek ismétléséből. Az a tény, hogy a stúdiók nevét, mint a `Disney`, meg kell mondani minden filmnél, amit a Disney gyártott, nem valódi redundancia, mivel ezt valahol úgyis meg kell mondani. □

Általában azt javasoljuk, hogy ha egy dologhoz több információ kapcsolódik, mint a neve, akkor valószínűleg szükséges egy egyedhalmaz vagy osztály ehhez a dologhoz. Azonban ha csak a neve is elegendő, akkor valószínűleg jobb, ha attribútumot készítünk belőle. Ez a megközelítés közel áll a sémák „normalizálásához" a relációs modellben, amelyet a 3.7. részben tárgyalunk.

2.17. példa: Vizsgáljuk meg a sokágú kapcsolatok használata és a kapcsoló egyedhalmaz bináris kapcsolatokkal való használata közti különbséget. Láttuk a négyágú *Szerződések* kapcsolatot egy színész, egy film és két stúdió között a 2.12. és 2.15. ábrákon, amelyet mechanikusan átkonvertáltunk egy *Szerződések* egyedhalmazzá. Vajon jól választottunk?

Egy ODL tervben nincs választási lehetőségünk, mert nem használhatunk sokágú kapcsolatokat. E/K modellben viszont mindkét lehetőség megvan. Azonban, ha változtatunk egy kicsit a problémán, akkor már majdnem biztos, hogy a kapcsoló egyed-



2.18. ábra. *Szerződések, mint olyan kapcsoló egyedhalmaz, amely egy színészhez, egy filmhez és a stúdiók egy halmazához kapcsolódik*

halmazt kell választanunk. Tegyük fel, hogy a szerződések egy színészt, egy filmet, de a stúdiók tetszőleges halmazát tartalmazhatja. Ez a szituáció bonyolultabb mint, ami a 2.12. 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 *Szerződések* kapcsolat kapcsolathalmaza a következő alakú hármasokból kell hogy álljon

(színész, film, stúdiók halmaza)

és a szerződések kapcsolat nem csak 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 helyes, azonban a stúdiók halmaza mint alapegyed természetellenesnek tűnik, és így nem javasoljuk.

Jobb megközelítés, ha a szerződéseket mint egyedhalmazt tekintjük. Mint a 2.15. 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 2.18. ábra mutatja az E/K diagramot. Látható, hogy egy szerződéshez tartozik egy színész, egy film és akárhány stúdió.

Ugyanez a tervezési stratégia az ODL-ben is működik. Például a következő interfész-deklaráció megfelelő lesz:

```
interface Szerződés {
   relationship Színész aSzínész;
   relationship Film aFilm;
   relationship Set<Stúdió> stúdiók;
};
```

Itt a szerződés objektumoknak három kapcsolata van, kapcsolatban áll egy színésszel, egy filmmel és a stúdiók egy halmazával. Az inverz kapcsolatokat elhagytuk. □

2.3.5. Feladatok

* **2.3.1. feladat:** A 2.19. ábrán egy banki adatbázis terve látható, amely az ügyfeleket és számláikat tartja nyilván. Tegyük fel, hogy a különböző kapcsolatok és attribútumok jelentése megegyezik azzal, amit a nevük sugall. Bíráljuk meg a tervet. Mely szabályokat sérti? Miért? Milyen módosításokat javasolunk?

!! **2.3.2. feladat:** Ebben és a következő feladatokban a születés E/K modelljének két lehetséges verzióját vizsgáljuk meg. Egy születésnél van egy gyerek (ikrek születését

```
interface Lakcím {
   attribute string cím;
   relationship Set<Ügyfél> lakók
      inverse Ügyfél::lakik;
};

interface Ügyfél {
   attribute string név;
   relationship Lakcím lakik
      inverse Lakcím::lakók;
   relationship Gyűjtőszámla számlái
      inverse Gyűjtőszámla::tulajdonos;
};

interface Számla {
   attribute real egyenleg;
   relationship Set<Gyűjtőszámla> eleme
      inverse Gyűjtőszámla::elemei;
};

interface Gyűjtőszámla {
   attribute string tulajdonosLakcím;
   relationship Ügyfél tulajdonos
      inverse Ügyfél::számlái;
   relationship Set<Számla> elemei
      inverse Számla::eleme;
};
```

2.19. ábra. *Egy banki adatbázis nem megfelelő terve*

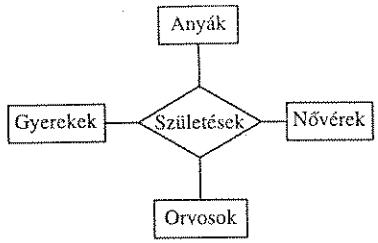
két szülésseI reprezentáljuk), egy anya, valamennyi nővér és valamennyi orvos. Ezért tegyük fel, hogy *Gyerekek, Anyák, Nővérek* és *Orvosok* egyedhalmazaink vannak. Ezenkívül vegyünk egy *Születések* kapcsolatot, amely a négy egyedhalmazt kapcsolja össze a 2.20. ábrán javasolt módon. Megjegyezzük, hogy a *Születések* 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 kapcsolathalmazba, a nővérek és orvosok összes lehetséges kombinációjának megfelelően.

A következő feltevéseket szeretnénk beépíteni a tervünkbe. Mondjuk meg, hogy milyen nyilakat vagy egyéb elemeket kell felvenni az E/K diagramban, hogy teljesüljenek az alábbi feltételek:

a) Minden gyereknek pontosan egy anyja van.



2.20. ábra. *A születések megvalósítása sokágú kapcsolattal*

b) Egy gyerek, nővér, orvos hármashoz pontosan egy anya tartozik.

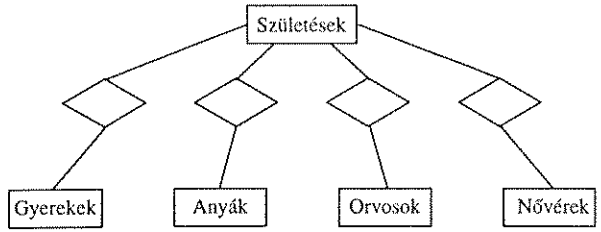
c) Egy adott gyerek és anya kombinációhoz pontosan egy orvos tartozik.

! **2.3.3. feladat:** A 2.3.2. 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 2.21. 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.



2.21. ábra. *A születéseket egy egyedhalmazzal adjuk meg*

!! **2.3.4. feladat:** Tegyük fel, hogy megváltoztatjuk a nézőpontunkat és megengedjük, hogy egy szülésben több gyereket is szülhessen egy anya. Hogyan lehetne ekkor meg-

valósítani azt, hogy minden gyereknek pontosan egy anyja van a 2.3.2. feladatban leírt és a 2.3.3. feladatban megadott megközelítések esetében külön-külön?

! **2.3.5. feladat:** Készítsük el a 2.3.2. feladat és 2.3.3. feladat E/K terveinek ODL leírását. Melyek azok a feltételek, amelyek könnyebben valósíthatók meg ODL-ben? Melyeket nem lehet megvalósítani? Hogyan kell módosítani a tervet, hogy megengedje a 2.3.4. feladatban leírt többes szüléseket?

2.4. Alosztályok

Gyakran lehetnek egy osztályban olyan objektumok, amelyeknek olyan speciális tulajdonságaik vannak, amelyekkel nem rendelkezik az osztály minden objektuma. Így hasznos az osztályt *alosztályokra* bontani, ahol minden alosztálynak vannak speciális attribútumai és/vagy kapcsolatai úgy, hogy az osztályhoz tartozó attribútumok és kapcsolatok rájuk is érvényesek. ODL-ben nagyon egyszerű az alosztályok deklarálása, amelyet a következő részben tárgyalunk. Utána látni fogjuk, hogyan lehet megvalósítani E/K modellben az osztály-alosztály hierarchiát egy speciális kapcsolat segítségével.

2.4.1. Alosztályok az ODL-ben

Azok között a filmek között, amelyeket tárolunk a példaként használt adatbázisunkban, vannak rajzfilmek, bűnügyi filmek, kalandfilmek, vígjátékok és sok egyéb speciális típusú film. Ezen filmtípusok mindegyike definiálható a 2.1. példában bevezetett `Film` osztály alosztályaként. Egy *C* osztályt úgy definiálhatunk egy *D* osztály alosztályaként, hogy a *C* deklarációjában *C* neve után egy kettőspontot teszünk, majd utána a *D* nevét.

2.18. példa. Deklarálhatjuk a `Film` osztály `Rajzfilm` alosztályát és akkor a `Film` osztály *szuperosztálya* lesz a `Rajzfilm` osztálynak.

```
1) interface Rajzfilm: Film {
2)   relationship Set<Színész> hangok;
  };
```

Az első sor deklarálja a `Rajzfilm` osztályt a `Film` osztály alosztályaként. A második sor azt mondja, hogy minden `Rajzfilm` objektumnak van egy `hangok` kapcsolata, amely azokat a színészeket jelenti, akik adják a hangjukat a rajzfilm szereplőinek. Nem adtuk meg a `hangok` kapcsolat inverzét, habár ezt meg kellett volna tennünk. Megjegyezzük, hogy a `hangok` kapcsolat nem kell az összes filmnél, csak a rajzfilmeknél, így nem akartuk felvenni a `Film` osztályba. □

Egy alosztály *örökli* a szuperosztály összes tulajdonságát (ezt úgy is mondjuk, hogy a szuperosztályból *származtatott* az alosztály). Azaz a szuperosztály minden attribútuma és kapcsolata automatikusan attribútuma és kapcsolata az alosztálynak. Így a 2.18. feladatban lévő minden `Rajzfilm` objektum örökölte a `Film` osztály `cím`, `év`, `hossz`, `szalagFajta` attribútumait és a `szereplők` és `gyártó` kapcsolatait, illetve van egy saját `hangok` kapcsolata.

2.4.2. Többszörös öröklődés az ODL-ben

Egy osztálynak lehet több alosztálya is, amelyek öröklik a szuperosztály tulajdonságait a 2.4.1. részben leírt módon. Ezenkívül az alosztályoknak is lehet alosztálya, ami osztályoknak egy hierarchiájához vezet, ahol minden osztály örökli az ősének tulajdonságait. Az is lehet, hogy egy osztálynak több szuperosztálya van. A következő példa a többszörös szuperosztályok lehetőségeit és problémáit illusztrálja.

2.19. példa: Definiáljuk a `Film` osztály egy másik alosztályát, a bűnügyi filmeket, köetkezőképpen:

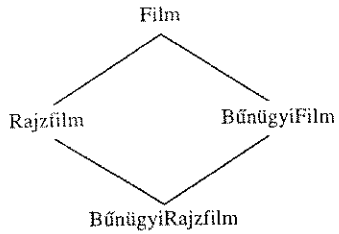
```
1) interface BűnügyiFilm: Film {
2)   attribute string fegyver;
  };
```

Így az összes bűnügyi filmnek van egy attribútuma, amely a gyilkos fegyvert tartalmazza és persze van négy attribútuma és két kapcsolata, amely az összes filmnek egyébként is van.

Vizsgáljunk meg egy olyan filmet mint a *Roger nyúl a pácban*, amely rajzfilm és bűnügyi film is egyben. Ezeknek a filmeknek az esetében szükség van a `hangok` kapcsolatra, a `fegyver` attribútumra és a `Film` osztály tulajdonságaira. Ez leírható azzal, hogy deklarálunk egy új `BűnügyiRajzfilm` alosztályt, amely alosztálya `Rajzfilm` és a `BűnügyiFilm` alosztályoknak. A konkrét deklaráció a következő:

```
interface BűnügyiRajzfilm: Rajzfilm, BűnügyiFilm {};
```

Így egy `BűnügyiRajzfilm` objektumnak minden tulajdonsága megvan, ami a `Rajzfilm` és a `BűnügyiFilm` alosztályoknak is megvan. Nem deklaráltunk külön attribútumokat vagy kapcsolatokat a bűnügyi rajzfilmeknek. A `BűnügyiRajzfilm` osztályobjektumai öröklik a `fegyver` attribútumot a `BűnügyiFilm` osztályból és a `hangok` kapcsolatot a `Rajzfilm` osztályból. Ráadásul, mivel a `BűnügyiFilm` osztály és a `Rajzfilm` osztály öröklik a `Film` osztály négy attribútumát és két kapcsolatát, így ezt a hat tulajdonságot is örökli a `BűnügyiRajzfilm` osztály. Azonban ezt a hat tulajdonságot a `BűnügyiRajzfilm` osztály nem két példányban örökli, hanem csak egy példányban a `Film` osztályból a két közvetlen szuperosztályán keresztül. A 2.22. ábra mutatja a tárgyalt négy osztály alosztály-szuperosztály viszonyát. □



2.22. ábra. *Többszörös öröklődés*

Általában, ha deklarálunk egy *C* osztályt, ami több más osztály alosztálya, akkor ezeket az osztályokat vesszővel elválasztva soroljuk fel a *C* nevét követő kettőspont után. Ezt mutatja a 2.19. példában a `BűnügyiRajzfilm` osztály deklarációja. Amikor egy *C* osztály több más osztályból örököl, akkor *konfliktus* lehet a tulajdonságok nevei között. Azaz *C* két vagy több szuperosztályának van egy attribútuma vagy kapcsolata, amelyeknek ugyanaz a neve, de különböző a típusa.

2.20. példa: Tegyük fel, hogy a `Film` osztálynak van két alosztálya a `Romantikusfilm` és a `Bíróságifilm`, és mindkét alosztálynak van egy `vége` attribútuma. A `Romantikusfilm` osztályban a `vége` attribútum egy felsorolás típusú attribútum, amelynek az értéke a `boldog` vagy a `szomorú` érték. A `Bíróságifilm` osztályban a `vége` attribútum szintén egy felsorolás típusú attribútum, de az értékei a `bűnös` vagy a `nemBűnös` érték. Ha most készítünk egy `RomantikusBíróságifilm` alosztályt, amelynek a `Romantikusfilm` osztály és a `Bíróságifilm` osztály is szuperosztálya, akkor a `vége` attribútum típusa tisztázatlan. □

Habár az ODL maga nem definiál semmi különleges szintaxist erre, az ODL megvalósítások biztosítják az alábbi mechanizmusok közül legalább az egyiket, hogy a felhasználó specifikálni tudja a többszörös öröklődésből fakadó konfliktusok kezelését.

1. Megmondjuk, hogy a tulajdonság két definíciója közül melyik kerüljön az alosztályba. Például a 2.20. példában eldönthetjük, hogy mi a fontosabb a bírósági romantikus filmek esetében, az, hogy boldog vagy szomorú vége van a filmnek, vagy az, hogy az ítélet bűnösség vagy nem bűnösség. Ebben az esetben egyszerűen specifikálhatjuk, hogy `RomantikusBíróságifilm` alosztály a `vége` attribútumot a `Romantikusfilm` szuperosztályából örökölje és ne a `Bíróságifilm` szuperosztályából.

2. Adunk egy új nevet a *C* osztályban a másik tulajdonságnak, amelynek ugyanaz a neve. Például a 2.20. példában, ha a `RomantikusBíróságifilm` osztály örökli a `Romantikusfilm` osztály `vége` attribútumát, akkor mi specifikálhatunk egy `ítélet` attribútumot, amely a `Bíróságifilm` osztály `vége` attribútumának egy átnevezése.

3. Újra definiáljuk a *C* osztály azon tulajdonságait, amelyek két vagy több szuperosztály által is meghatározottak. Például a 2.20. példában dönthetünk úgy, hogy a `vége` attribútumot a `RomantikusBíróságifilm` osztály nem örökli egyik szuperosztályától sem. Helyette újra definiáljuk a `vége` attribútumot egész típusú attribútumként, amely a nézők elégedettségi fokát mutatja a szavazataik alapján.

Megjegyezzük, hogy a 2.19. példában is vannak konfliktusok, ugyanis a `BűnügyiRajzfilm` osztály örököl a közvetlen szuperosztályaiból (`Rajzfilm` és `BűnügyiFilm`) hat tulajdonságot, mint a `cím` vagy a `színészek` tulajdonságokat, amelyeket a két szuperosztály a `Film` osztályból örököl. Mivel azonban a `cím` és a többi tulajdonság definíciója azonos a `Rajzfilm` és a `BűnügyiFilm` szuperosztályokban, így bármelyik definíciót választhatjuk.

2.4.3. Alosztályok az E/K diagramokban

Emlékezzünk rá, hogy az ODL-ben az osztály fogalma és az E/K modellben az egyedhalmaz fogalma megfelelnek egymásnak. Tegyük fel, hogy egy *C* osztály egy *D* osztálynak alosztálya. Ez a fogalom kiterjeszthető az E/K modellben is egy speciális kapcsolat segítségével. Rendeljünk a *C* osztályhoz és a *D* osztályhoz is egy-egy egyedhalmazt. Rajzoljuk meg az egyedhalmazokat reprezentáló téglalapokat és rendeljük *C* egyedhalmazához az attribútumait és kapcsolatait, illetve rendeljük *D* egyedhalmazához a közös attribútumaikat és kapcsolataikat.

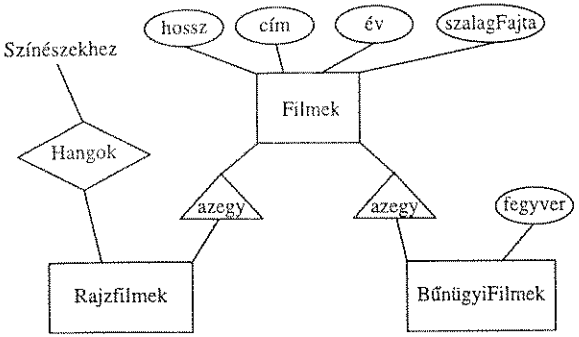
Az alosztály kapcsolatnak megfelelően kössük össze őket egy éllel, amelynek a közepén egy háromszög van. A háromszögnek a csúcsa mutat a szuperosztályra. A háromszögbe opcionálisan az „az egy" szavak írhatók.*

2.21. példa: A *Filmek* egyedhalmazt és a *Rajzfilmek*, illetve *BűnügyiFilmek* alosztályát mutatja a 2.23. ábra. Ez az alosztály struktúra hasonló ahhoz, amit a 2.19. példában ODL-ben készítettünk. A háromszögek a *Rajzfilmek* és a *BűnügyiFilmek* esetében is a *Filmek* szuperosztályra mutatnak. Nem adtuk meg a *Filmek* egyedhalmaz *Szereplők* és *Gyártó* kapcsolatát a *Színészek*, illetve *Stúdiók* egyedhalmazok felé, viszont ábrázoltuk a *Rajzfilmek* egyedhalmaz *Hangok* kapcsolatának egyik felét. □

2.4.4. Öröklődés az E/K modellben

Van egy hajszálnyi különbség az ODL-ben és más objektumorientált modellben lévő öröklődési fogalom és az E/K modellben lévő öröklődési fogalom között. ODL-ben egy objektumnak pontosan egy osztályhoz kell tartoznia. Így a 2.19. példában defini-

* Szerkesztői megjegyzés: Az angol terminológiában az „isa" kapcsolat terjedt el, ami jól olvasható: „a Cartoon isa Movie". Magyarul ezt hasonló tömörséggel az „egy Rajzfilm azegy Film" adja vissza.



2.23. ábra. *Alosztályok az E/K diagramban*

álnunk kellett a `BűnügyiRajzfilm` osztályt, amely azokat az objektumokat tartalmazza, amelyek rajzfilmek és bűnügyi filmek is egyben. Nem lehetne elhelyezni például a *Roger nyúl a pácban* című filmet a `Rajzfilmek` osztályban és a `Bűnügyifilmek` osztályban egyszerre.

E/K modellben úgy tekinthetünk egy egyedet, mint aminek olyan komponensei vannak, amelyek az öröklődési hierarchiában lévő egyedhalmazokhoz tartoznak. A komponensek az egyedhez a speciális (háromszöggel reprezentált) öröklődési kapcsolatokon keresztül tartoznak. Az egyednek attribútuma a komponensének bármely attribútuma és kapcsolata komponensének bármely kapcsolata.

Ebből a szempontból ugyanaz a hatás, mint ODL-ben, mivel a tulajdonságok öröklődése pont ugyanazokat az attribútumokat és kapcsolatokat jelenti egy objektum esetében, mint amelyeket a hozzákapcsolt egyed összegyűjt a komponenseiből. Azonban van egy kis különbség, amit a 2.22. példában tárgyalunk, és a 3.4. részben látni fogunk egy másik különbséget is, amikor ODL és E/K tervet konvertáljuk relációs adatmodellé.

2.22. példa: Figyeljük meg a 2.23. ábrán, hogy nem szükséges felvennünk egy egyedhalmazt a bűnügyi rajzfilmekhez. Az ok az, hogy olyan egyed, mint a *Roger nyúl a pácban,* 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 ad a *Filmek* egyedhalmaz (és két kapcsolatot, amelyek nincsenek a 2.23. ábrán), a *fegyver* attribútumot adja *BűnügyiFilm* egyedhalmaz és a *Hangok* kapcsolatot adja a *Rajzfilm* egyedhalmaz. Ezek pontosan a *Roger nyúl a pácban* objektum tulajdonságai, amelyeket a `BűnügyiRajzfilm` osztály örököl a `Film`, a `Rajzfilm` és a `BűnügyiFilm` szuperosztályokból a 2.19. példában. □

Megjegyezzük azonban, hogy ha vannak olyan tulajdonságai a bűnügyi rajzfilmeknek, amelyek nincsenek sem a rajzfilmeknek, sem a bűnügyi filmeknek, akkor szükségünk lesz egy negyedik *BűnügyiRajzfilm* egyedhalmazra a 2.23. ábrán, amelyhez hozzákapcsoljuk ezeket a tulajdonságokat (attribútumokat, kapcsolatokat). Ekkor a *Roger nyúl a pácban* egyednek négy komponense lesz, és a *BűnügyiRajzfilm* egyedhalmaz biztosítja az új tulajdonságokat a *Roger nyúl a pácban* egyed részére.

2.4.5. Feladatok

* **2.4.1. feladat:** Tekintsük a hadihajók egy adatbázisát ODL-ben. Minden hadihajóról a következő információkat szeretnénk nyilvántartani:

1. Nevét.
2. Vízkiszorítását tonnában.
3. Típusát, azaz, hogy csatahajó vagy romboló.

Ráadásul négyfajta hajót szeretnénk nyilvántartani a következő kiegészítő információkkal:

1. *Ágyúnaszád,* amely nagyméretű fegyvereket szállít, mint amilyeneket a csatahajók vagy a cirkálók. Ezeknél szeretnénk nyilvántartani a fegyverek számát és kaliberét.
2. *Repülőgép-anyahajó,* amely a repülőgépeket szállítja. Itt szeretnénk tárolni a fedélzeti leszálló pálya hosszát és a hozzá tartozó repülőgépek csoportjait.
3. *Tengeralattjáró,* amellyel a víz alatt is lehet közlekedni. Ezeknél szeretnénk nyilvántartani a maximális merülési mélységet. Feltehetjük, hogy nincs olyan ágyúnaszád vagy repülőgép-anyahajó, amelyik tengeralattjáró is egyben.
4. *Csatarepülőgép-anyahajó,* amely egyszerre ágyúnaszád és repülőgép-anyahajó. Természetesen minden információt tárolni szeretnénk róluk, amit az ágyúnaszádokról és repülőgép-anyahajókról tárolunk.[6]

A következőket oldjuk meg:

1. Adjuk meg a fenti osztályok ODL tervét.
2. Mutassuk meg, hogy hogyan reprezentálható az *Ise* csatarepülőgép-anyahajó. A vízkiszorítása 36 000 tonna, 8 darab 14 hüvelyk kaliberű ágyút szereltek rá, 200 láb hosszú a leszállópályája és az 1-es és 2-es repülőcsoportot szállította.

[6] A valóságban tényleg van ilyen. Az *Ise* és *Hyuga* japán csatahajók hátsó részét 1943-ban átépítették, hogy legyen egy leszállópálya és egy fedett hangár.

!! **2.4.2. feladat:** Bizonyos alosztályoknak a 2.4.1. feladatban, mint a `Csatarepülőgép-anyahajó` osztálynak, csak egy típusa lehetséges, viszont másoknak, mint az `Ágyúnaszád` osztálynak, több is lehetséges, például csatahajó vagy cirkáló stb. Ez okoz-e redundanciát? Ha igen, akkor hogyan lehetne megszüntetni?

* **2.4.3. feladat:** Ismételjük meg a 2.4.1. feladatot E/K modellben.

! **2.4.4. feladat:** Módosítsuk a 2.1.5. feladatban lévő „emberek" adatbázistervét a következő speciális embertípusokkal:

1. Nők.

2. Férfiak.

3. Szülők.

Megkülönböztethetnénk bizonyos más embertípusokat is, hogy a kapcsolatok a megfelelő alosztályhoz kapcsolódjanak. Készítsük el újra a tervet

a) ODL-ben.

b) E/K modellben.

2.5. Megszorítások modellezése

Eddig azt láttuk, hogyan modellezhetjük a valós világ egy szeletét ODL osztályok és jellemzőik, attribútumaik és kapcsolataik segítségével, vagy E/K modellben egyedhalmazok és kapcsolataik segítségével. A legtöbb struktúra, ami érdekel bennünket, leírható ezen jelölések valamelyikével. Azonban vannak más fontos nézőpontjai a valós világnak, amelyek nem modellezhetők az eddig látott eszközökkel. Ezek az információk az osztályok, attribútumok és kapcsolatok definiálásakor struktúrákra, típusokra kirótt *megszorításokon* túl gyakran az adatokra vonatkozó megszorítások formájában jelentkeznek.

Az általában használt megszorítások durva osztályozása következik. Azonban nem fogjuk lefedni a megszorítástípusoknak mindegyikét. A 4.5. részben relációs algebrai környezetben és a 6. fejezetben pedig SQL programozási környezetben további megszorításokat találunk.

1. *Kulcsok:* olyan attribútumok vagy attribútumok olyan halmazai, amelyek egyértelműen azonosítanak egy objektumot az osztályán belül, vagy egy egyedet az egyedhalmazán belül. Azaz nincs két olyan objektum egy osztályon belül, amelyek tulajdonságértékei megegyeznek minden attribútumon, ami egy kulcsot alkotó halmazba tartozik.

A megszorítások a séma részei

Ha az adatbázist egy bizonyos időpontban tekinthetjük, akkor hibásan is dönthetünk arról, hogy egy attribútum kulcs, mert abban a pillanatban nincs két objektum, amely megegyezne ezen az attribútumon. Például, ha létrehozzuk a film adatbázist, akkor kezdetben nem lesznek egyforma című filmek benne. Azonban, ha ez alapján az előzetes tény alapján úgy döntünk, hogy a `cím` a kulcs, és megtervezzük, illetve tároljuk az adatbázis struktúráját ezzel a kulccsal, akkor nem fogjuk tudni felvinni másodszor a *King Kong* című filmet az adatbázisunkba.

A kulcs és az egyéb megszorítások általában részei az adatbázissémának. Az adatbázis-tervező adja meg őket a strukturális tervezéssel együtt. Ha egy megszorítás deklarált, akkor minden olyan beszúrás vagy módosítás az adatbázison, ami ellentmond ennek a megszorításnak, tiltott lesz.

Így, habár az adatbázis egy különleges példánya kielégít bizonyos megszorításokat, akkor is csak azokat a valódi megszorításokat kell megadni az adatbázistervezőnek, amelyeknek érvényeseknek kell lenni az adatbázis összes, a valós világot helyesen modellező példányában. Ezek azok a megszorítások, amelyeket a felhasználók feltételezhetnek, és az adatbázis struktúrájában tárolódni fognak.

2. *Egyértékűségi megszorítások:* olyan követelmények, hogy az érték egy bizonyos szerepben egyedi. Ezek speciális esetei a kulcsok, mivel megkövetelik, hogy a hozzá tartozó értékek egyediek legyenek. Azonban, mint látni fogjuk, vannak más lehetőségei is az egyértékűségi megszorításoknak.

3. *Hivatkozásiépség-megszorítások:* megkövetelik, hogy egy objektum által hivatkozott érték létezzen az adatbázisban. Ez analóg azzal, hogy a hagyományos programokban tilosak a sehova se mutató mutatók.

4. *Értelmezésitartomány-megszorítások:* azt jelentik, hogy egy attribútum az értékeit a megadott értékhalmazból vagy értéktartományból veheti fel. A 6.3. részben az SQL szempontjából fogjuk tárgyalni az értelmezési tartomány megszorításokat.

5. *Általános megszorítások:* tetszőleges követelmények, amelyeket be kell tartani az adatbázisban. Például, megkövetelhetjük, hogy legfeljebb 10 színészt sorolunk fel minden filmnél. Látni fogunk általános megszorítást kezelő nyelveket a 4.5. és 6.4. részekben.

Több oka van annak, hogy ezek a megszorítások fontosak. Valami további információt adnak a valós világ modellezni kívánt részéről. Például a kulcsok segítségével

azonosíthatunk egyértelműen egy objektumot vagy egyedet. Ha tudjuk, hogy a `név` attribútum egy kulcs a `Stúdió` osztályban, akkor hivatkozhatunk egy stúdió objektumra a nevével. Ráadásul időt és helyet takarítunk meg egy egyedi érték ismeretével, mivel könnyebb tárolni egy értéket, mint egy halmazt, még akkor is, ha a halmaz egyelemű.[7] A hivatkozások épsége és a kulcsok támogatják az objektumok gyorsabb elérését bizonyos tárolási technikák esetén.

2.5.1. Kulcsok

Egy osztály *kulcsa* ODL-ben egy vagy több attribútum K halmaza úgy, hogy bármely két O_1 és O_2 különböző objektumok esetében a két objektum nem egyezhet meg a K-beli attribútumok mindegyikén. E/K modellben a *kulcs* pontosan ugyanez, csak ki kell cserélni az osztály szót egyedhalmazra és az objektum szót egyedre.

2.23. példa: Tekintsük a `Film` osztályt a 2.1. példából. Először azt gondolhatnánk, hogy a `cím` attribútum maga a kulcs. Azonban van néhány filmcím, amelyet több különböző film is használ, mint például a *King Kong*. Így nem lenne bölcs dolog a `cím` attribútumot kulcsnak deklarálni. Ha ezt tennénk, akkor nem tudnánk tárolni mindkét *King Kong* című filmet az adatbázisunkban.

Egy jobb választás, ha a `cím` és `év` attribútumok halmazát vesszük kulcsnak. Továbbra is megmarad az a kockázat, hogy lesz két olyan film, amelyeket ugyanabban az évben készítettek és ugyanaz a címük (és így mindkettő nem tárolható az adatbázisunkban), azonban ez nem túl valószínű.

A 2.1. részben bevezetett másik két osztály, a `Színész` és `Stúdió` esetében is óvatosan kell megállapítani a kulcsot. A stúdiók esetében feltételezhetjük, hogy nincs két stúdió, amelynek ugyanaz a neve, így a `név` attribútumot vehetjük a `Stúdió` osztály kulcsának. Azonban a színészek esetében a név valószínűleg nem azonosít. Ugyanis a név nem különböztet meg két embert általában. Mivel azonban a színészek hagyományosan „művésznevet" választanak maguknak, így remélhetjük, hogy a `név` attribútum kulcs lesz a `Színész` osztály esetében is. Ha esetleg nem, akkor a `név` és a `lakcím` attribútumpárt még mindig választhatjuk kulcsnak, hacsak nem lesz két olyan színész, akinek a neve megegyezik és ugyanaz a lakcímük is. □

2.24. példa: A 2.23. példában tapasztaltak mutatják, hogy nehéz a kulcsot megtalálni, illetve azt mondani, hogy attribútumok egy halmaza biztosan kulcs. A gyakorlatban azonban ez sokkal egyszerűbb. A valós világ azon részei esetében, amelyet adatbázissal modellezünk, az emberek készítenek tényleges kulcsokat a fontos osztályokhoz. Például a vállalatok minden dolgozóhoz azonosítót rendelnek, és ezeket az azonosítókat gondosan úgy választják, hogy egyediek legyenek. Ezeknek az azonosítóknak az a céljuk, hogy biztosítsák a vállalat adatbázisán belül, hogy minden dolgozó megkülönböztethető legyen a többi dolgozótól, még akkor is, ha néhány dolgozónak ugyanaz a neve. Így a dolgozó azonosítójának megfelelő attribútum kulcs lesz a dolgozók esetében az adatbázisban.

[7] Megjegyezzük, hogy egy C programban is egyszerűbb reprezentálni egy egészet, mint egészeknek egy listáját még akkor is, ha az csak egy egészet tartalmaz.

Az Egyesült Államok vállalataiban teljesen normális, hogy minden dolgozónak van egy Társadalombiztosítási száma. Ha az adatbázisban van egy attribútum a Társadalombiztosítási számnak, akkor ez az attribútum lehet kulcs a dolgozók esetében. Megjegyezzük, hogy nem okoz problémát, ha egy osztálynak több kulcsa is van, mint a dolgozók esetében az azonosító és a Társadalombiztosítási szám.

Ez az ötlet, hogy készítsünk egy attribútumot, amelynek a célja, hogy kulcs legyen, meglehetősen elterjedt. A dolgozó azonosítójának mintájára használhatunk azonosítót az egyetemi hallgatók megkülönböztetésére. Továbbá a vezetői igazolványok számát és az autók rendszámát használhatjuk a sofőrök és az autók azonosítására. Az olvasó további példákat tud találni, ahol egy attribútum létrehozásának az elsődleges célja, hogy kulcs legyen. □

2.5.2. Kulcsok deklarálása az ODL-ben

ODL-ben egy vagy több attribútumot deklarálhatunk kulcsnak egy osztályban. Egy kulcsot a `key` vagy a `keys` kulcsszó (mindegy, hogy melyik) után megadott attribútummal, illetve attribútumokkal deklarálhatjuk. Ha több attribútum alkotja a kulcsot, akkor az attribútumok listáját zárójelbe kell tenni. A kulcs deklarációja közvetlenül az interfész deklarációja után következik az attribútumokat és kapcsolatokat magába foglaló részt megnyitó kapcsos zárójel előtt. A deklarációt magát zárójelek közé kell tenni.

2.25. példa: Deklaráljuk a `Film` osztályt kulcsával, amely a `cím` és `év` attribútumok halmaza. Egyszerűen ki kell cserélni a 2.6. ábra első sorát a következőre:

```
interface Film
   (key (cím, év))
{
```

Használhatnánk a `keys` kulcsszót is, még akkor is, ha csak egy kulcsunk van.

Hasonlóan, ha a `név` attribútum a kulcs a `Színész` osztályban, akkor be kell szúrni a következő sort a 2.6. ábra 8-as sorába a kapcsos zárójel elé:

```
(key név)
```

□

Lehetséges, hogy több attribútumhalmaz is kulcs. Ha így van, akkor a `key(s)` kulcsszó után a kulcsokat vesszővel kell elválasztani. Mint egyébként, ha egy kulcs több attribútumot is tartalmaz, akkor zárójelek között kell megadni az attribútumok listáját, így egyértelmű, hogy egy több attribútumot tartalmazó kulcsról van-e szó vagy pedig több, egyattribútumos kulcsról.

2.26. példa: Tekintsük a `Dolgozó` osztályt, amelynek több kulcsa is van, mint láttuk egy példában. A teljes osztályt, attribútumokat és kapcsolatokat, nem írjuk le most, de tegyük fel, hogy van két attribútuma az osztálynak, a `dolgAzon` és a `tbSzám`, amelyek a dolgozóazonosítót és a Társadalombiztosítási számot reprezentálják. Mindkét attribútumot kulcsnak deklarálhatjuk. Azaz

```
(key dolgAzon, tbSzám)
```

Mivel nincs zárójelek között az attribútumok listája, így ez az ODL-ben azt jelenti, hogy az attribútumok maguk a kulcsok. Ha zárójelek közé tettük volna a listát, azaz `(dolgAzon, tbSzám)`, akkor ez azt jelentené ODL-ben, hogy a két attribútum együtt ad egy kulcsot. Tehát a következő sor azt jelenti, hogy két dolgozónak nem egyezhet meg dolgozóazonosítója és Társadalombiztosítási száma, habár lehet olyan két dolgozó, akik a kettő közül valamelyik attribútumon megegyeznek:

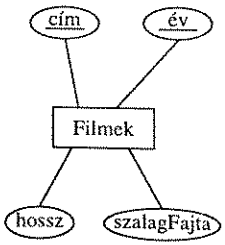
```
(key (dolgAzon, tbSzám)) □
```

2.5.3. Kulcsok jelölése az E/K modellben

Egy egyedhalmaz lényegében egy osztály, így pontosan ugyanolyan értelemben van kulcsa, mint egy ODL osztálynak. Ha attribútumok egy halmaza kulcsot alkot egy egyedhalmazon belül, akkor nincs két olyan egyed, amely megegyezik a kulcs összes attribútumán. E/K diagramban aláhúzással jelöljük a kulcs attribútumait egy egyedhalmazban. Például a 2.24. ábra mutatja a 2.8. ábráról vett *Filmek* egyedhalmazt a *cím* és *év* attribútumokkal, amelyek egy kulcsot alkotnak.

Abban az esetben, ha több kulcs van, az E/K modell nem biztosít jelölést ezeknek a megadására. Szokásos, hogy csak egy kulcsot adunk meg, az *elsődleges kulcsot,* és ezt úgy kezeljük, mint ha több kulcs nem is lenne. Az elsődleges kulcsot jelöljük aláhúzással az E/K modellben és a többit, a *másodlagos kulcsokat,* vagy nem is jelöljük, vagy pedig megjegyzésben felsoroljuk őket meliékelve a diagramhoz.

Van egy szokatlan, de lehetséges eset, amikor magának az egyedhalmaznak nincs kulcsa. Ezeket gyenge egyedhalmazoknak hívjuk és a 2.6. részben tárgyaljuk.



2.24. ábra. *`Filmek` egyedhalmaz kulccsal*

2.5.4. Egyértékűség

Gyakran egy fontos tulajdonsága az adatbázis tervezésének, hogy legfeljebb egy érték tölthet be egy különleges szerepet. Például feltételezzük, hogy a cím, év, hossz és szalagFajta egyértelműen megad egy film objektumot, vagy hogy egy filmet pontosan egy stúdió gyártott. ODL-ben nincs probléma ezeknek a feltételeknek a deklarálásával, mert minden attribútumnak van típusa. Ha a típus nem kollekciótípus (például halmaz), akkor csak egy értéke lehet az adott attribútumnak, vagy csak egy kapcsolt objektum lehet a kapcsolatban. Másrészt, ha egy attribútum vagy kapcsolat egy kollekciótípussal definiált, mint a `Set<Színész>` típus a `szereplők` kapcsolatban a 2.6. ábra 6. sorában, akkor megengedett, hogy több színész is kapcsolódjon egy adott filmhez. Az ilyen kapcsolatot *többértékű* kapcsolatnak nevezzük.

Különbséget kell tenni, hogy legfeljebb egy értéke van egy attribútumnak, illetve egy kapcsolatnak, vagy pedig *pontosan* egy értéke van. Amikor van egy kapcsolatunk, amelyben az egyik osztályból objektumok kapcsolódnak egy másik osztály egyetlen objektumához, és ez utóbbi objektum kötelezően létezik is, akkor kapunk egy „hivatkozási épség" megszorítást, amelyet a 2.5.5. részben tárgyalunk. Amikor egy attribútumnak egy értéke van, akkor két lehetőségünk van:

1. Megköveteljük, hogy ez az érték létezzen.

2. Megengedjük, hogy ez az érték ne legyen megadva.

Ha egy attribútum része az osztály kulcsának, akkor általában megköveteljük, hogy létezzen minden objektumban. Más attribútumok esetében bevezethetünk egy *nullértéket,* ami azt jelenti, hogy nincs értéke az attribútumnak. Nem lehet nullérték megszorítás ennek az attribútumnak az esetében.

2.27. példa: A 2.23. példában eldöntöttük, hogy a `Film` osztály kulcsa a `cím` és az `év` attribútumok. Nyilván megköveteljük, hogy ez a két attribútum minden film objektumban létezzen. Viszont a `hossz` attribútum értéke lehet kitöltetlen. Használhatnánk a –1-et a `hossz` attribútum nullértékének, mivel nem lehet negatív hosszúságú egy film. Ha egy filmnek nem ismerjük a hosszát, akkor a `hossz` attribútum értékét –1-re állíthatjuk. Hasonlóan bevezethetnénk egy harmadik értéket is a felsorolásban, amikor definiáljuk a `szalagFajta` attribútum lehetséges értékeit. A `színes` és `feketeFehér` értékeken túl, választhatnánk egy `NULL` vagy `ismeretlen` értéket, annak jelölésére, hogy a szalag fajtájáról nincs információnk. □

Az E/K modell is felkínálja az egyértékűségeket. Egy egyedhalmaz minden attribútuma egyértékű. Általában feltesszük, hogy ez az érték lehet a nullérték is. Ha egy attribútum nem lehet nullérték, akkor ezt megjegyzésben mellékelni kell.

A nyilak, amelyek mutatják a sok-egy és egy-egy kapcsolatokat is, kifejezik az egyértékűségeket. Azaz, ha egy kapcsolatban nyíl van az egyik *E* egyedhalmazhoz, akkor *E*-nek legfeljebb egy egyede kapcsolódhat a másik egyedhalmaz minden egyedéhez.

2.5.5. Hivatkozások épsége

Amíg az egyértékűségi megszorítás ügyel arra, hogy legfeljebb egy érték szerepeljen egy adott szerepben, addig a *hivatkozásiépség-megszorítás* arra ügyel, hogy pontosan egy érték legyen az adott szerepben. A hivatkozásiépség-megszorítás egy fajtája lehetne az, hogy egy attribútumnak nem lehet nullértéke, de a hivatkozások épségét általánosabban az osztályok közötti kapcsolatok hivatkozásaira használjuk.

Vizsgáljuk meg a `gyártó` kapcsolatot, amely a `Film` osztályt köti össze a `Stúdió` osztállyal a 2.6. ábrán lévő ODL terv 7. sorában. Megkérdezhetnénk, hogyan lehetséges az, hogy a `gyártó` kapcsolatnak az értéke egy stúdió objektum kell hogy legyen, de stúdió objektum még nem is létezik. A válasz az, hogy egy ODL megvalósításban a `gyártó` kapcsolatot egy mutató vagy egy hivatkozás reprezentálja, ami egy stúdió objektumra mutat. Lehetséges, hogy a stúdió objektumot kitörlik a `Stúdió` osztályból, ekkor azonban a mutató lógni fog, azaz nem mutat többé valós objektumra.

A hivatkozások épsége azt követeli meg a `gyártó` kapcsolat esetében, hogy a hivatkozott stúdió objektum létezzen. Van néhány módszer, aminek segítségével ezt a megszorítást állandóan betarthatjuk.

1. Egy hivatkozott objektumot nem törlünk (példánkban lévő stúdió objektumot nem szabad törölni).

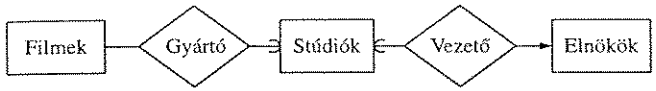
2. Megköveteljük, hogy ha egy hivatkozott objektumot törlünk, akkor töröljük az összes objektumot, ami hivatkozik rá. A példánkban ez azt jelenti, hogy ha törlünk egy stúdió objektumot, akkor törölnünk kell az összes filmet, amit ez a stúdió gyártott.

Ráadásul ezen törlési stratégiák egyike mellé még megköveteljük, hogy amikor egy film objektumot létrehozunk, akkor egy létező stúdió objektumot kell megadni a `gyártó` kapcsolatában. Továbbá, ha egy kapcsolatnak megváltozik az értéke, akkor az új értéknek is egy létező objektumnak kell lenni. Egy kapcsolat hivatkozási épségének a fenti elvek alapján történő betartása az adatbázis megvalósításának része, amivel itt nem foglalkozunk.

2.5.6. Hivatkozási épség az E/K diagramokban

Kiegészíthetjük a nyíl jelölést úgy, hogy mutassa 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 nem csak sok-egy vagy egy-egy kapcsolat, hanem megköveteli az egy oldalon a hivatkozás épségét, azaz az E minden egyedéhez kell hogy létezzen egy vele kapcsolatban álló F egyedhalmazbeli egyed. Ugyanezen az ötlet használható, amikor R több egyedhalmaz közötti kapcsolat.

2.28. példa: A 2.25. ábra mutatja a hivatkozásiépség-megszorításokat a *Filmek, Stúdiók* és *Elnökök* egyedhalmazok között. Ezeket az egyedhalmazokat és kapcsolatokat a 2.8. és 2.9. ábrákon vezettük be. Láthatjuk, hogy a *Stúdiók* egyedhalmazhoz megy egy kerek nyíl a *Gyártó* kapcsolatban. Ez azt jelenti, hogy annak a stúdiónak, amely gyártott egy filmet, mindig benne kell lenni a *Stúdiók* egyedhalmazban.



2.25. ábra. *Hivatkozásiépség-megszorításokat tartalmazó E/K diagram*

Hasonlóan kerek nyilat találunk a *Vezető* kapcsolatban szintén a *Stúdiók* oldalán. Ez azt jelenti, hogy ha egy elnök vezetője egy stúdiónak, akkor annak a stúdiónak létezni kell a *Stúdiók* egyedhalmazban.

Megjegyezzük, hogy a *Vezető* 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. □

2.5.7. Egyéb megszorítások

Mint a fejezet elején említettük, vannak másfajta megszorítások is, amelyeket szeretnénk érvényesíteni az adatbázisban. Itt ezeket csak röviden érintjük, részletesen a 6. fejezetben foglalkozunk velük.

Az *értelmezésitartomány-megszorítás* korlátozza az attribútumok felvehető értékeit. Az ODL megkövetel egy típust minden attribútum esetében, ami egy kezdetleges formája az értelmezésitartomány-megszorításnak. Például, ha a `hossz` attribútum típusa `integer`, akkor nem lehet értéke a 101,5 vagy egyéb nem egész érték. Azonban az ODL nem támogatja a további szűkítéseket, például azt, hogy a hossz 60 és 240 között kell hogy legyen. A 6.3. részben látni fogjuk, hogy az SQL viszont támogatja az ilyen megszorításokat.

Vannak általános megszorítások, amelyek a fejezetben említett egyik kategóriába sem esnek bele. Ilyen egy kapcsolat foka. Például, egy film objektumhoz vagy egyedhez nem kapcsolódhat 10-nél több színész objektum vagy egyed a `szereplők` kapcsolatban. E/K modellben a kapcsolat éleire írhatjuk rá a korlátot, ami azt jelenti, hogy a kapcsolatban részt vevő egyedek száma korlátozott az egyedhalmazra nézve. ODL-ben ugyanezt típusmódosítással érhetjük el. Például megváltoztatjuk a `szereplők` típusát úgy, hogy egy 10 elemű tömb legyen, mivel nincs mód arra, hogy megadjuk egy halmaz elemeinek maximális számosságát.

2.29. példa: A 2.26. ábra mutatja, hogy E/K modellben hogyan lehet megadni azt a megszorítást, amely nem engedi, hogy 10-nél több szereplője legyen egy filmnek. Mint egy korábbi példában megjegyeztük, a nyíl annak a megszorításnak a szinonimája, hogy „≤ 1" és a kerek nyíl, amit a 2.25. ábrán láttunk, pedig az „= 1" szinonimája. □

2.26. ábra. *A filmenkénti maximális szereplők számát szabályozó megszorítás*

2.5.8. Feladatok

2.5.1. feladat: Adjuk meg a következő feladatokhoz készült ODL tervekben a kulcsokat:

* a) 2.1.1. feladat

b) 2.1.3. feladat

* c) 2.1.5. feladat

d) 2.4.1. feladat

2.5.2. feladat: Adjuk meg a következő feladatokhoz készült E/K diagramokban a kulcsokat és a hivatkozásiépség-megszorításokat:

* a) 2.2.1. feladat

b) 2.2.3. feladat

c) 2.2.6. feladat

d) 2.4.3. feladat

! **2.5.3. 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 az 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 egy kulcsa az E_i egyedhalmaznak. Adjuk meg R legkisebb lehetséges kulcsát E_1 és E_2 segítségével az alábbi feltevések mellett:

a) R egy sok-sok kapcsolat.

* b) R egy sok-egy kapcsolat úgy, hogy E_2 az egy oldal.

c) R egy sok-egy kapcsolat úgy, hogy E_1 az egy oldal.

d) R egy egy-egy kapcsolat.

!! **2.5.4. feladat:** Vizsgáljuk meg ismét a 2.5.3. feladatban lévő problémát, de n bármilyen szám lehessen, ne csak 2. Használva azt az információt, hogy R milyen nyilakkal kapcsolódik az egyedhalmazokhoz, adjuk meg, hogy hogyan lehetne megtalálni az R legkisebb K lehetséges kulcsát K_i-k segítségével.

! **2.5.5. feladat:** Adjunk további példákat (2.24. példában felsoroltakon kívül) a valós életből arra, hogy egy attribútum létrehozásának elsődleges célja az, hogy kulcs legyen.

2.6. Gyenge egyedhalmazok

Van egy szokatlan, de lehetséges eset, amikor egy egyedhalmaz kulcsában szereplő attribútumok közül néhány, vagy esetleg az összes, más egyedhalmaznak attribútuma. Az ilyen egyedhalmazokat *gyenge egyedhalmazoknak* nevezzük.

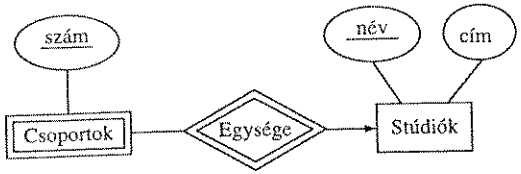
2.6.1. A gyenge egyedhalmazok bevezetésének okai

Két elvi oka lehet a gyenge egyedhalmazok bevezetésének. Egyik, amikor egy egyedhalmaz része egy másiknak, azaz, ha az E egyedhalmaz része az F egyedhalmaznak, akkor lehetséges, hogy E-beli egyedek nevei nem egyértelműek, amíg ha az E-beli egyedhez tartozó F-beli egyed nevét hozzávesszük, akkor igen.

2.30. példa: Nézzünk néhány példát egyedhalmazok hierarchiájára, ami gyenge egyedhalmazhoz vezet:

1. 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 2.27. ábra. A *Csoportok* gyenge egyedhalmaz kulcsa a saját *szám* attribútuma és a *Stúdiók* egyedhalmaz *név* attribútuma, amelyhez a *Csoportok* egyedhalmaz és az *Egysége* sok-egy kapcsolaton keresztül kapcsolódik.

2. A fajokat nemzetségükkel és fajukkal jelölik. Például az emberek a *Homo sapiens* fajhoz tartoznak, ahol a *Homo* a nemzetség neve és a *sapiens* a faj neve. Általában



2.27. ábra. *Egy gyenge egyedhalmaz és kapcsolatai*[8]

egy nemzetség fajokat tartalmaz, amelyeknek a neve a nemzetség nevével kezdődik és a faj nevével fejeződik be. Sajnos 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. Tehát a *Fajok* egy gyenge egyedhalmaz, amelynek a kulcsa a nemzetséggel egészül ki. □

A gyenge egyedhalmazok másik forrása a 2.2.5. részben bemutatott eljárás, amelynek segítségével a sokágú kapcsolatokat átalakítjuk binárissá[9]. Ezeknek az egyedhalmazoknak gyakran nincs is attribútuma. A kulcsuk olyan attribútumokból áll, amelyek a hozzákapcsolódó egyedhalmazok kulcsattribútumai.

2.31. példa: A 2.28. ábrán láthatjuk a *Szerződések* kapcsoló egyedhalmazt, amely helyettesíti a 2.8. példában lévő *Szerződések* háromágú kapcsolatot. 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. □

2.6.2. Gyenge egyedhalmazokra vonatkozó követelmények

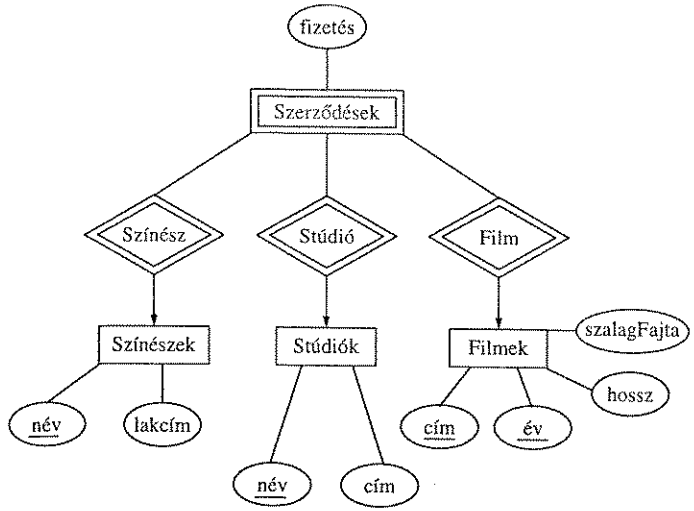
Egy gyenge egyedhalmaz kulcsa nem származhat akárhonnan. Ha E egy gyenge egyedhalmaz, akkor minden F egyedhalmaz esetén, amely egy vagy több attribútumával hozzájárul E kulcsához, E-nek kapcsolódnia kell F-hez egy R kapcsolaton keresztül. Továbbá a következő feltételeknek kell eleget tenni:

1. R-nek egy sok-egy kapcsolatnak[10] kell lennie, amelynek az egy oldala az F egyedhalmaznál van.

[8] A dupla vonalakat a 2.6.3. részben magyarázzuk el.

[9] Megjegyezzük, hogy az E/K modellben nem kötelező megszüntetni a sokágú kapcsolatokat, habár ODL-ben és az olyan régi modellekben, mint a 2.7. részben tárgyalt hálós és hierarchikus modellek, a sokágú kapcsolatokat át kell alakítani binárissá.

[10] 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.



2.28. ábra. *A kapcsoló egyedhalmazok gyengék*

2. F azon attribútumai, amelyek benne vannak E kulcsában, benne vannak F kulcsában is.

3. Ha F maga is gyenge egyedhalmaz, akkor F azon kulcsattribútumai, amelyek benne vannak E kulcsában, lehet, hogy egy F-hez sok-egy kapcsolattal kapcsolódó másik egyedhalmaz attribútumai.

4. Ha több sok-egy kapcsolat is vezet E-ből F-be, akkor minden kapcsolaton keresztül segíthetik F kulcs attribútumai E kulcsának kialakítását. Megjegyezzük, hogy egy E-beli e egyed más-más F-beli egyeddel kapcsolódhat a különböző 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 2.30. 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 nem elég. Az egyetlen mód, hogy további információt kapcsoljunk a csoportokhoz, ha van egy determinisztikus eljárás, ami generálja a további értéket, amelynek segítségével a csoport egyértelmű lesz. Csak a következő módon találhatunk egy egyértelmű értéket egy csoportegyedhez, ha:

1. az érték a *Csoportok* egyedhalmaz egy attribútumának értéke, vagy ha

Miért nincsenek „gyenge osztályok" ODL-ben?

Az, hogy hogyan kell megkeresni egy kulcsot, sosem merül fel ODL-ben vagy más objektumorientált modellben. A 2.5.2. részben láttuk, hogy tudunk deklarálni egy vagy több attribútum felhasználásával kulcsot, de ezt nem kötelező megtenni. Az objektumoknak van objektumazonosítójuk, ami egy cím, ahol megtalálhatók. Az objektumazonosító egyértelműen megkülönbözteti egymástól az objektumokat, még akkor is, ha az attribútumok értékei és a kapcsolatok nem különböztetik meg őket. Ezzel szemben az E/K modell „értékorientált" és az egyedeket csak az attribútumaikhoz rendelt értékek különböztetik meg. Így az E/K diagramban gondosan ügyelni kell arra, hogy bármely egyedhalmaz egyedei megkülönböztethetők legyenek egymástól az értékek alapján anélkül, hogy lenne bármilyen „objektumazonosító".

2. az érték a csoport egyed egy kapcsolatán keresztül egy másik egyed attribútumának egy értéke, és ez az egyed egyértelmű. Azaz a kapcsolat sok-egy típusú (vagy speciális esetben egy-egy típusú). Ezenkívül a vett értéknek kulcsnak kell lenni a másik egyedhalmazban.

2.6.3. Gyenge egyedhalmazok jelölése

A következő jelöléseket használjuk gyenge egyedhalmazokkal kapcsolatban.

1. Ha egy egyedhalmaz gyenge, akkor dupla kerettel jelezzük. Ez a jelölés látható a 2.27. és 2.28. ábrákon.

2. Ha egy egyedhalmaz gyenge, akkor dupla kerettel jelöljük azt a sok-egy kapcsolatát, amelyen keresztül a kulcsa kiegészül. Ez látható az *Egysége* kapcsolat esetében a 2.27. ábrán és a 2.28. ábra összes kapcsolata esetében.

3. Ha egy egyedhalmaz bármely attribútuma része a kulcsnak, akkor aláhúzással jelöljük. Például a 2.27. ábrán a csoportok száma részt vesz a kulcsban, bár nem adja a teljes kulcsot.

   Összegezhetjük a fenti jelöléseket a következő szabályban:

- A dupla kerettel jelölt egyedhalmazok gyengék. Egy ilyen egyedhalmaz kulcsa előáll a saját aláhúzott attribútumaiból és azon egyedhalmazok kulcs attribútumaiból, amelyek dupla kerettel megjelölt sok-egy kapcsolattal kapcsolódnak a gyenge egyedhalmazhoz.

2.6.4. Feladatok

* **2.6.1. 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 nem csak azt reprezentálja, hogy egy hallgató mely kurzusokat vette fel, hanem a kurzusra kapott érdemjegyet is. Rajzoljunk egy E/K diagramot a fenti esethez, jelöljük be a gyenge egyedhalmazokat és a kulcsokat. Az érdemjegy része-e a kurzusfelvételt reprezentáló egyedhalmaz kulcsának?

2.6.2. feladat: Módosítsuk a 2.6.1. feladatot úgy, hogy a hallgató összes részteljesítési érdemjegyét tartsuk nyilván, ami egy adott kurzus elvégzéséhez kapcsolódik. Ismét jelöljük a gyenge egyedhalmazokat és a kulcsokat.

2.6.3. feladat: A 2.3.3. feladathoz készült E/K diagramban jelöljük be a gyenge egyedhalmazokat és a kulcsokat.

2.6.4. 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 tanszéknek a 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.

2.7. Történeti fontosságú korábbi adatmodellek

Ebben a fejezetben bemutatunk két további modellt és azok terminológiáit. A „hálós" és „hierarchikus" modellek korai kísérletek voltak az adatbázisrendszerek alapjainak megteremtésére. Az első kereskedelmi adatbázisrendszerekben használták őket a 60-as évek végén és a 70-es években. A 3. fejezetben tárgyalt relációs modellen alapuló rendszerek azóta kiszorították őket. Azonban sok olyan megoldást, ötletet tartalmaznak, amelyek tovább élnek az újabb objektumorientált adatbázis-tervezésben.

2.7.1. Hálós modell

A *hálós modell* tekinthető az E/K modell bináris sok-egy kapcsolatokra való szűkítésének. A hálós modellnek két alapeleme van:

1. *Logikai rekordtípusok.* Ezek az egyedhalmazokhoz hasonlóak, a típus nevéből és attribútumaik listájából állnak. A logikai rekordtípus elemeit *rekordoknak* nevezzük, amelyek az E/K modell egyedeinek felelnek meg.

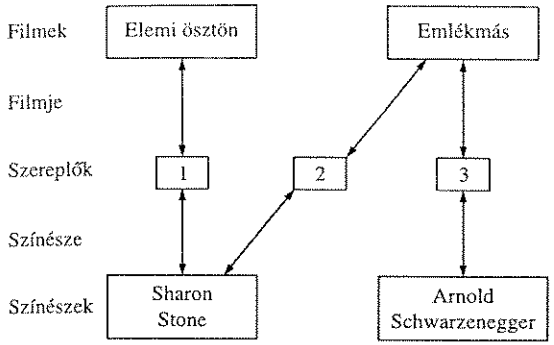
2. *Kapcsok.* Ezek sok-egy bináris kapcsolatok. Két egyedhalmazt kapcsolnak össze, ahol az egyik egyedhalmaz *tulajdonos típusú*, a másik pedig a *tag típusú*. A kapcsolat egy oldala a tulajdonos, és a sok oldala pedig a tag. Azaz a tag minden rekordjához pontosan egy tulajdonos típusú rekord kapcsolódik és minden tulajdonos típusú rekordhoz nulla, egy vagy több tag típusú rekord kapcsolódhat.

2.32. példa: Tekintsük a filmek, színészek és a filmek szereplőit tartalmazó példánkat hálós modellben. A színészek és filmek nyilván logikai rekordtípusok lesznek. Azonban a „szereplők" kapcsolat egy sok-sok kapcsolat, így nem tudjuk megadni egy kapocsként a hálós modellben. Készítünk egy új logikai rekordtípust `Szereplők` néven, ami egy kapcsoló logikai rekordtípus lesz, hasonlóan a 2.2.5. részben tárgyalt kapcsoló egyedhalmazhoz. Minden szereplő rekord egy színész-film pár, ami azt jelenti, hogy a színész szerepelt a filmben. Tehát három logikai rekordtípus lesz a hálós tervünkben:

```
Színészek(név, lakcím)
Filmek(cím, év, hossz, szalagFajta)
Szereplők()
```

A `Színészek` logikai rekordtípusnak két attribútuma van, a név és a színész lakcíme, a `Filmek` típus attribútumai a film címe, gyártási éve, hossza és hogy milyen szalagra rögzítették. Nem ábrázoljuk a filmekhez kapcsolódó stúdiót, habár a teljes terv tartalmazhatna egy kapcsot ehhez a kapcsolathoz. A kapcsoló `Szereplők` típusnak nincs attribútuma tervünkben, habár lehetne. Például, ha tárolni szeretnénk a színésznek fizetett díjat, akkor ez az összeg a színész-film pártól függne, és így a `Szereplők` típus attribútuma lehetne. Ebben a példában azonban a `Szereplők` típus rekordjainak a szerepe a kapocs megteremtése a résztvevők között.

Két kapocs van a tervünkben. Az egyik kapocs a `Színészek` és `Szereplők` között van. Itt a `Színészek` a tulajdonos és a `Szereplők` a tag. Ezt a kapcsot nevezzük el `Színésze` kapocsnak. Ez kapcsol minden egyes színész-film párhoz egy színészt. A második kapocs a `Filmje`, amelyben a tulajdonos a `Filmek` típus és a tag pedig a `Szereplők` típus. Minden film rekordhoz tartoznak színész-film párok. Megjegyezzük, hogy mindkét kapocs sok-egy típusú. Egy (s, f) `Szereplők` típusú párhoz az s-nek megfelelő `Színészek` típusú rekord és az f-nek megfelelő `Filmek` típusú rekord kapcsolódik.



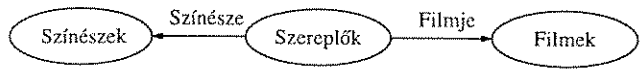
2.29. ábra. *Rekordok és kapcsok*

A 2.29. ábra illusztrálja, hogy a három logikai rekordtípus rekordjai hogyan kapcsolódnak egymáshoz a kapcsokon keresztül. Ez a diagram nem séma, inkább a rekordokat magukat mutatja és a módot, ahogyan egymáshoz kapcsolódnak a kapcsokon keresztül. Láthatunk három `Szereplők` rekordot. Az 1 jelenti, hogy Sharon Stone az Elemi ösztön című filmben szerepelt. A 2 jelenti a Sharon Stone–Emlékmás, és a 3 jelenti az Arnold Schwarzenegger–Emlékmás párokat. Ezek a számok nem részei a rekordoknak, csak arra használjuk őket, hogy hivatkozhassunk a `Szereplők` rekordjaira. Megjegyezzük, hogy a `Szereplők` rekordjai két kapocsban is tagok, míg a többi rekord tulajdonos. □

2.7.2. Hálós sémák ábrázolása

A logikai rekordtípusokat és kapcsokat diagramban is ábrázolhatjuk. Általában a logikai rekordtípusokat egy kör, a kapcsokat pedig egy nyíl jelöli. A logikai rekordtípusok nevét a körbe írjuk, kapcsok nevét pedig a nyílra írjuk rá. A nyíl mindig a tagtól a tulajdonos felé mutat.

2.33. példa: A 2.32. példa három logikai rekordtípusa és köztük lévő két kapocs a 2.30. ábrán látható. □



2.30. ábra. *A filmes példa hálós sémája*

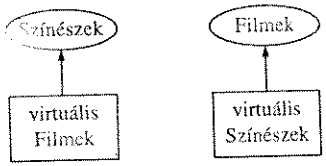
2.7.3. Hierarchikus modell

A hierarchikus modell tekinthető a hálós modell szűkítésének, ahol a logikai rekordtípusok és kapcsok egy erdőt alkotnak (fák egy halmazát). Azaz, ha minden kapcsot úgy tekintünk, hogy a tulajdonos őse a tagnak, akkor a logikai rekordtípusok egy erdőt alkotnak. A probléma ezzel az, hogy ez nem minden hálóra készíthető el. Például a 2.30. ábrán lévő `Szereplők` logikai rekordtípus esetében két ős kell a hierarchiában, egyik a `Színészek`, másik a `Filmek`. Így a 2.30. ábra nem egy erdő.

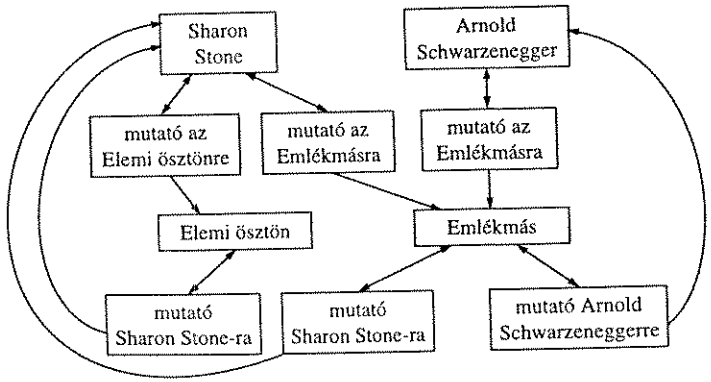
Idézzük fel, hogy a `Szereplők` logikai rekordtípus valójában egy kapcsoló típus a színészek és filmek között, egy sok-sok kapcsolatot helyettesít. A hierarchikus modellben a sok-sok kapcsolat reprezentálható a kapcsolódó típusok *virtuális* másolata segítségével. A virtuális típus tekinthető úgy, mint egy mutató a valós típus egy rekordjára.

2.34. példa: A 2.31. ábra mutatja a filmek-színészek példa hierarchikus sémáját. Két egyszerű fa alkotja az erdőt. Az első fa gyökere a `Színészek` és ennek a leszármazottja virtuális `Filmek`. A második fa gyökere a `Filmek` és a leszármazottja pedig a virtuális `Színészek`.

A 2.31. ábra sémáját bemutató adatok láthatók a 2.32. ábrán. A sémában a `Szereplők` leszármazottja a virtuális `Filmek`. Így minden `Színészek` típusú



2.31. ábra. *A filmes példa hierarchikus sémája*



2.32. ábra. *A filmes példa hierarchikus reprezentációja*

Miért hierarchiák?

Az ember csodálkozhat azon, hogy mi támaszthatta alá az adatbázisiparban a nem túl természetesnek tűnő hierarchikus modell használatát. A legegyszerűbb érv, hogy az adatok hierarchiába szervezése által, virtuális rekordokat csak akkor használva, amikor az feltétlenül szükséges, az adatok egy szekvenciális állományban tárolhatók úgy, hogy egybegyűjtjük a rekordokat és őseiket. Így tárolhatjuk a 2.32. ábra adatait egy rekordban a következő módon. Sharon Stone-t követi az összes „tulajdona", azaz ebben az esetben virtuális `Filmek` mutatók. Az a feltételezés, hogy az információhoz úgy jutunk el, hogy lefelé haladunk a fában. Az ehhez szükséges információt mindig közvetlenül mellette találjuk meg az állományban úgy, mintha az őstől haladnánk a leszármazott felé a fában, amivel időt takarítunk meg a kívánt információ előkeresésében.

Ebben a példában csak kis előnye van a hierarchikus szervezésnek, mivel a virtuális típusokat reprezentáló mutatókat követve tetszőleges helyre juthatunk el azon lemezen, ahol az információt tárolják. Azonban, a sok-sok kapcsolatot kivéve, gyakran lényegesen hatékonyabb lekérdezést eredményez a hierarchikus szervezés.

rekord leszármazottja virtuális `Filmek` típusú. A virtuális rekordokat a „mutató a/az ....-ra/re" kifejezéssel jelöltük az ábrán.

Például látható, hogy a `Színészek` Sharon Stone rekordjának két leszármazottja van. Mindkettő egy mutató a `Filmek` egy-egy rekordjára, ebben az esetben az *Elemi ösztön* és az *Emlékmás* rekordokra. Követve a színészek és filmek közötti sok-sok kapcsolatot, kiindulhatunk egy `Színészek` típusú rekordból, mint Sharon Stone, és eljutunk a leszármazott virtuális `Filmek` típusú rekordokhoz, majd onnan a valódi `Filmek` rekordokhoz. □

2.7.4. Feladatok

2.7.1. feladat: Írjuk át a következő feladatokban leírt terveket hálós modellbe:

* a) 2.1.1. feladat

b) 2.1.3. feladat

c) 2.1.5. feladat

d) 2.3.2. feladat

2.7.2. feladat: Ismételjük meg a 2.7.1. feladatot hierarchikus modellre.

*! **2.7.3. feladat:** Tegyük fel, hogy van egy egyed-kapcsolat diagramunk n egyedhalmazzal és m bináris kapcsolattal. Ha ezt konvertáljuk hálós modellre, akkor mennyi lesz a szükséges kapcsok minimális és maximális száma? Minden kapcsolat lehet sok-sok, sok-egy vagy egy-egy típusú.

!! **2.7.4. feladat:** Tegyük fel, hogy van egy egyed-kapcsolat diagramunk n egyedhalmazzal és m bináris kapcsolattal. Adjuk meg a szükséges virtuális rekordtípusok minimális és maximális számát, ha átírjuk ezt a diagramot hierarchikus modellbe.

! **2.7.5. feladat:** Hogyan válaszolnánk a 2.7.3. és 2.7.4. feladatok kérdéseire, ha a kapcsolatok k ágú kapcsolatok lennének, ahol k>2?

2.8. Összefoglalás

- *Tervezési jelölésrendszerek:* Adatbázis-tervezésre használhatunk egyed/kapcsolat modellt vagy objektumorientált modellt, mint az ODL. Az E/K modellt áttranszformálják egy valós adatbázisrendszer modelljébe, ami gyakran a relációs modell. Az ODL tervet vagy ugyanúgy áttranszformálják, vagy majdnem direkt inputja lehet egy objektumorientált adatbázisrendszernek.

- *ODL leírás:* Ebben a nyelvben objektumok osztályait adjuk meg attribútumaikkal, kapcsolataikkal és metódusaikkal. Attribútumoknak meg kell adni a típusát is. Az ODL típusrendszere magába foglalja a hagyományos típusokat, mint az egésztípusok stb., és tartalmaz típuskonstruáló lehetőségeket, amelyekkel struktúrákat, halmazokat, multihalmazokat, listákat, tömböket adhatunk meg. A kapcsolatokat a kapcsolódó osztállyal írjuk le. Megengedett, hogy egy kapcsolat többértékű legyen.

- *Egyed/kapcsolat diagram:* E/K modellben egyedhalmazokat adunk meg attribútumaikkal és az egyedhalmazok közti kapcsolatokat. Egyedhalmazok elemeit egyedeknek nevezzük. Négyszögeket, rombuszokat és köröket használunk az egyedhalmazok, kapcsolatok és attribútumaik ábrázolásához.

- *Kapcsolatok típusa:* Mind ODL-ben, mind E/K modellben hasznos megkülönböztetni a kapcsolatokat típusuk alapján. Bináris kapcsolat lehet sok-sok, sok-egy vagy egy-egy típusú. Több egyedhalmaz közötti kapcsolat E/K modellben megengedett, míg ODL-ben nem.

- *Gyenge egyedhalmazok:* Esetenként problémát okozhatnak az E/K modellben a gyenge egyedhalmazok. Ezek egyedei kapcsolódó egyedhalmazok attribútumainak segítségével azonosíthatók. A gyenge egyedhalmazok megkülönböztetésére a dupla keret speciális jelölést használjuk.

- *A jó terv:* Az adatbázis-tervezés megköveteli, hogy a választott jelölésrendszerrel (ODL vagy E/K például) 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 ODL és az E/K modell is támogatja osztályok vagy egyedhalmazok speciális eseteinek leírását. ODL-ben van alosztály és öröklés fogalom, míg az E/K modellben ezt egy speciális kapcsolaton keresztül oldhatjuk meg.

- *Kulcsok:* Az ODL és az E/K modell is megengedi, hogy attribútumok egy halmazát kulcsnak deklaráljuk, ami azt jelenti, hogy ezen attribútumok értéke egyértelműen definiál egy objektumot vagy egyedet. ODL-ben van objektumazonosító fogalom is. Ez olyan érték, amely egyértelműen azonosítja az objektumokat, de nem elérhető a felhasználók számára.

- *Hálós modell:* Ezt a modellt ma már ritkán használják. Hasonló az E/K diagramhoz, azzal a megszorítással, hogy a kapcsolatok binárisak és sok-egy típusúak.

- *Hierarchikus modell:* Ez a modell is nagyon ritka ma már. Ez a modell is hasonlít az E/K modellhez. Itt az egyedhalmazok egy erdőbe vannak rendezve sok-egy kapcsolatokkal, amelyeknek a sok oldala a leszármazott, és az egy oldala az ős.

2.9. Irodalomjegyzék

Az E/K modellt leíró eredeti cikk a [3]. A [4] és [1] referenciák az egyed-kapcsolat tervezést alaposan tárgyalják, de más modelleket is leírnak.

A [2] egy ODL kézikönyv. Ez egy ODMG (Object Data Management Group) folyamatban lévő munkája. Ez a szervezet elektronikus elérést is biztosít, ahol az ODL újabb verziói is találhatók. E-mail címük: info@odmg.org és a web oldaluk címe: http://www.odmg.org.

1. Batini, C., S. Ceri, S. B. Navathe, *Conceptual Database Design,* Benjamin/Cummings, Redwood City, CA, 1992.

2. Cattell, R. G. G. (szerkesztő), *The Object Database Standard: ODMG–93 Release 1.2,* Morgan-Kaufmann, San Francisco, 1996.

3. Chen, P. P. „The entity-relationship model: toward a unified view of data", *ACM Trans. on Database Systems* **1**:1, pp. 9–36, 1996.

4. El Masri, R., S. B. Navathe, *Fundamentals of Database Systems,* Benjamin Cummings, Menlo Park, 1994.

3. fejezet

A relációs adatmodell

Az adatmodellezés második fejezetben tárgyalt objektumorientált és egyed-kapcsolat szemléletmódjai jól leírják az adatszerkezeteket, napjainkban azonban az adatbázis-implementációk majdnem mindig egy másik szemléleten alapulnak, az ún. *relációs modellen*. A relációs modell az egyetlen adatmodellezési fogalom használata miatt bizonyult különösen hasznosnak: a „reláció" az adatok kétdimenziós táblázatban való elrendezését jelenti. Az 5. fejezetben látjuk, hogy a relációs modell hogyan támogat egy nagyon magas szintű programozási nyelvet, az SQL-t (Structured Query Language, Strukturált Lekérdezőnyelv). Az SQL segítségével egyszerű programokkal hatékonyan kezelhetjük a relációkban tárolt adatokat.

Másrészt az adatbázisok tervezése a 2. fejezetben tanult modellek segítségével gyakran könnyebb, emiatt az első célunk, hogy bemutassuk hogyan tudjuk az ODL, illetve E/K terveket átírni relációkká. Ezután ismertetjük a relációs modell saját adatbázis-tervezési elméletét. Ezt az elméletet általában a relációk „normalizálásának" nevezzük. Az elmélet elsősorban a „funkcionális függőségek"-re támaszkodik, amelyek magukban foglalják és kiterjesztik a „kulcs" fogalmát, amit korábban nem formálisan vezettünk be a 2.5.1. részben. A normalizálás elméletével gyakran javítani tudjuk a konkrét adatbázistervet reprezentáló relációink megválasztását.

3.1. A relációs modell alapjai

A relációs modellben az adatok egyszerűen reprezentálhatók: kétdimenziós táblákban, ún. *relációkban*. A 3.1. ábrán egy példát mutatunk relációra. A reláció neve `Film`, és ugyanazt az információt tartalmazza, mint a 2.1. példában a 2.4. ábrán szereplő ODL

cím	*év*	*hossz*	*szalagFajta*
Csillagok háborúja	1977	124	színes
Rút kiskacsa	1991	104	színes
Wayne világa	1992	95	színes

3.1. ábra. *A `Film` reláció*

`Film` osztály, amelyet itt újból megismétlünk a 3.2. ábrán. Megjegyezzük, hogy ez a definíció a `Film` osztálynak nem a végleges definíciója, hanem csak a `cím`, `év`, `hossz` és `szalagFajta` attribútumokat tartalmazza.

3.1.1. Attribútumok

A reláció fejrészében találhatók az *attribútumok*. A 3.1. ábrán az attribútumok: `cím`, `év`, `hossz` és `szalagFajta`. A reláció-attribútumok a reláció oszlopnevei. Általában az attribútumok megadják az abban az oszlopban szereplő adatok jelentését. Például a `hossz` attribútummal ellátott oszlop tartalmazza a különböző filmek hosszát percekben.

Megjegyezzük, hogy a 3.1. ábrán szereplő `Film` reláció-attribútumai a 2.4. ábra ODL definíciójában szereplő attribútum nevű struktúraelemeknek felelnek meg. Ez teljesen természetes megközelítése annak, hogyan válasszuk meg a reláció-attribútumokat. Bár általában nem követelmény, hogy egy reláció attribútumainak meg kellene felelniük az adatok ODL vagy E/K leírása valamely komponenseinek.

```
1) interface Film {
2)    attribute string cím;
3)    attribute integer év;
4)    attribute integer hossz;
5)    attribute enumeration(színes, feketeFehér)
      szalagFajta;
  };
```

3.2. ábra. *A `Film` osztály ODL leírása*

3.1.2. Sémák

A reláció nevét és a reláció-attribútumok halmazát együtt nevezzük *relációsémának*. A relációsémát a reláció nevével és az attribútumainak zárójelek közötti felsorolásával adjuk meg. Tehát a 3.1. ábra `Film` relációsémája a következő:

```
Film(cím, év, hossz, szalagFajta)
```

Figyeljük meg, hogy a relációséma attribútumai halmazt alkotnak, nem pedig listát, mégis amikor a relációkról van szó általában, meg kell határoznunk az attribútumoknak valamilyen „standard" sorrendjét. Tehát amikor attribútumok listájával vezetjük be a relációsémát, mint az előbb, akkor ugyanezt a standard sorrendet vesszük akkor is, amikor megjelenítjük a relációt vagy annak a sorait.

A relációs modellben a terv egy vagy több relációsémát tartalmaz. A relációsémákból álló halmazt a tervben *relációs adatbázissémának*, vagy csak röviden *adatbázissémának* nevezzük.

3.1.3. Sorok

A reláció azon sorait, amelyek különböznek az attribútumokból álló fejléc sorától *soroknak (tuple)* nevezzük. A reláció minden egyes attribútumához tartozik a sorban egy *komponens*. Például a 3.1. ábra három sora közül az első sornak négy komponense: `Csillagok háborúja`, `1977`, `124`, `színes`, amelyek ebben a sorrendben a `cím`, `év`, `hossz` és `szalagFajta` attribútumokhoz tartoznak. Amikor külön írjuk le a sorokat, nem pedig valamely reláció részeként, akkor vesszőkkel választjuk el a komponenseket, és a sort pedig bezárójelezzük. Például

```
(Csillagok háborúja, 1977, 124, színes)
```

a 3.1. ábra 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, mint ahogyan az attribútumokat felsoroltuk a relációsémában.

Gyakran azt gondolhatjuk, hogy a sorok objektumokat reprezentálnak, azok a relációk pedig, amelyekhez tartoznak, az osztályukat reprezentálják. A példánkban szereplő relációnál valóban ez a helyzet, mindegyik sor egy-egy film objektumot reprezentál. A sorainkban szereplő komponensek és a 3.2. ábrán megadott film objektumok tulajdonságai azonosak. Csak arra kell ügyelnünk, hogy az objektumoknak van azonosítójuk, a soroknak pedig nincs. Azaz elvileg a filmek objektumorientált reprezentációjában lehet két olyan különböző film objektum, amelyeknek az összes attribútumon ugyanazok az értékei, bár mint azt korábban a 2.23. példában tárgyaltuk, a filmek esetén azt várjuk, hogy ez nem történhet meg.

A relációk sorokból álló halmazok, és így egy sor nem fordulhat elő többször is az adott relációban. Tehát, ha a reláció objektumokból álló osztályt reprezentál, akkor meg kell arról bizonyosodunk, hogy a relációnak megfelelő-e az attribútumhalmaza, azaz nincs két objektum, amelyek ugyanazokat az értékeket vennék fel a reláció minden attribútumán. A filmek esetében a 2.23. példában már kijelentettük, hogy nem várunk két olyan filmet, amelynek ugyanaz a címe és ugyanabban az évben készült. Legrosszabb esetben készítenünk kell egy olyan attribútumot, amely magának az objektumazonosítónak egy mesterséges reprezentációja. Például minden filmhez hozzárendelhetünk egy egyértelmű egész „filmazonosítót", és ezt a `filmazonosítót` hozzávesszük a példában szereplő relációnk attribútumhalmazához.

3.1.4. 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 vagy bármely más olyan típus, amelynek az értékei kisebb komponensekre felbonthatók. Ez az egyik olyan követelmény, ami miatt nem tudjuk az ODL-ben előforduló attribútumokat közvetlenül a reláció egyszerű attribútumaira lefordítani. Például, ha a `név` ODL-attribútum típusa

```
Struct Név { string vezetéknév, string utónév}
```

akkor a megfelelő relációban kell hogy legyen két attribútum, a `vezetéknév` és az `utónév`. Ezt a 3.2.2. részben fogjuk teljesebben kifejteni.

Feltesszük továbbá, hogy a reláció minden attribútumához tartozik egy *értéktartomány*, azaz egy elemi típus. A reláció bármely sorában szereplő komponensek csak olyan értékeket vehetnek fel, amelyekre minden komponens értékének a megfelelő oszlop értéktartományából kell származnia. Például a 3.1. ábrán szereplő `Film` reláció sorainak az első komponense karaktersorozat, a második és harmadik komponense egész, és a negyedik komponens értéke pedig a `színes` és `feketeFehér` konstansok közül az egyik.

3.1.5. Relációk egyenértékű ábrázolási módjai

Tanultuk, hogy a relációnak mind a sémája, mind a sorai halmazok, nem pedig listák. Emiatt lényegtelen, hogy milyen sorrendben jelenítjük meg őket. Például a 3.1. ábra három sorát a hat lehetséges sorrend bármelyikében is írjuk fel, a reláció „ugyanaz", mint ami a 3.1. ábrán található.

Továbbá a reláció-attribútumoknak 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 3.3. ábrán látható reláció a 3.1. 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. Megjegyezzük, hogy az attribútumok és oszlopok permutációja minden sorra külön-külön történik.

3.1. példa: A 3.1. ábra

```
(Csillagok háborúja, 1977, 124, színes)
```

sora és a 3.3. ábra

```
(1977, Csillagok háborúja, színes, 124)
```

sora ugyanazt az objektumot reprezentálja. Mégis csak akkor bizonyosodhatunk meg

A sor formális fogalma

Ebben a könyvben a sorokat listaként adjuk meg, azaz minden relációra az attribútumok valamilyen sorrendjét értelmezzük. Van azonban a soroknak olyan fogalma is, amellyel elkerülhetjük az attribútumok sorrendjének a rögzítését. A sort ekkor függvénynek tekintjük, azaz a relációséma attribútumairól az értékekre történő leképezésnek – azaz, amely ezekhez az attribútumokhoz az adott sor megfelelő komponenseit rendeli. Például a 3.1. példában két sorrendben adjuk meg ugyanazt a sort, amelyet a következő függvénynek tekinthetünk:

```
cím → Csillagok háborúja
év → 1977
hossz → 124
szalagFajta → színes
```

az ekvivalenciáról, ha tudjuk a megfelelő reláció-attribútumok sorrendjét. Azaz általában azt javasoljuk, hogy válasszuk meg a reláció-attribútumok sorrendjét, és maradjunk ennél a sorrendnél mindaddig, amíg a relációt használjuk. □

év	*cím*	*szalagFajta*	*hossz*
1991	Rút kiskacsa	színes	104
1992	Wayne világa	színes	95
1977	Csillagok háborúja	színes	124

3.3. ábra. *A* `Film` *reláció másik megjelenítése*

3.1.6. 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 *relációelőfordulásnak* nevezzük. Például a 3.1. ábrán található három sor a `Film` reláció egy előfordulása. Feltehetően a `Film` reláció változott már a múltban és változni fog a jövőben. Például, 1980-ban a `Film` nem tartalmazta a `Rút kiskacsa` és `Wayne világa` sorokat. A hagyományos adatbázisrendszerek bármely relációnak csak egyetlen változatát kezelik: csak azokat a sorokat, amelyek „most" vannak a relációban. Ezt a relációelőfordulást *aktuális előfordulásnak* nevezzük.

Sémák és előfordulások

Ne felejtsük el a relációséma és a relációelőfordulás közötti lényeges különbséget. A séma a reláció nevét és attribútumait tartalmazza és viszonylag változatlan. Az előfordulás pedig a reláció sorainak a halmaza és az előfordulás gyakran változhat.

A séma/előfordulás megkülönböztetése általános az adatmodellezésben. Például a 2. fejezetben megkülönböztettük az ODL interfész definícióit, amellyel az objektumosztályok szerkezetét adtuk meg, és az adott osztályban előforduló objektumok halmazát. Az interfész definíciója a sémáéval analóg, a definiált osztály objektumainak halmaza pedig az előfordulással. Hasonlóan az egyed és kapcsolat leírások a sémaleírás E/K-modellbeli formái, és az egyedhalmazok és a kapcsolathalmazok az E/K-séma előfordulását adják. Megjegyezzük, hogy az adatbázis-tervezésnél az adatbázis előfordulása nem része a tervnek. Amikor elkészítjük a tervünket csak azt képzeljük el, hogyan nézhetnek ki a tipikus előfordulások.

3.1.7. Feladatok

3.1.1. feladat: Legyen a 3.4. ábrán található két relációelőfordulás 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.

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

* a) Három attribútum és három sor, mint a 3.4. ábra `Számlák` relációjában?

b) Négy attribútum és öt sor?

c) n attribútum és m sor?

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

3.4. ábra. *Egy banki adatbázis két relációja*

3.2. ODL sémák átírása relációsémákká

Nézzük meg azt a folyamatot, ahogy létrehozunk egy új adatbázist, például a film adatbázist. A tervezési fázissal kezdjük, amikor olyan kérdésekre válaszolunk, hogy milyen információt fogunk tárolni, hogyan kapcsolódnak egymáshoz az információ egyes elemei, milyen megszorításokat teszünk fel, például milyen kulcsmegszorításokat vagy hivatkozási épség típusú megszorításokat tételezünk fel stb. Ez a fázis hosszú ideig elhúzódhat amíg a lehetőségeket kiértékeljük és a véleményeket összehangoljuk.

A tervezési fázis után következik az implementációs fázis egy valódi adatbázisrendszer felhasználásával. Minthogy a kereskedelmi forgalomban levő adatbázisrendszerek nagy többsége a relációs modellen alapul, feltehetjük, hogy a tervezési fázisban is inkább ezt a modellt használtuk, mint a második fejezetben tárgyalt objektumorientált ODL modellt vagy az E/K modellt.

A gyakorlatban azonban sokkal egyszerűbb, ha először a második fejezetben tárgyalt modellek közül az egyikkel elkészítjük a tervünket, és azután ezt alakítjuk át relációs modellé. Ennek az az elsődleges oka, hogy a relációs modellben csak egyetlen fogalom van, a reláció, és nincsenek különféle kiegészítő fogalmak (mint például az E/K modellben az egyedhalmazok és kapcsolatok), és a modell rugalmatlanságait könnyebb azután kezelni, miután már kiválasztottuk a tervet.

Ebben a részben megnézzük, hogyan tudjuk átírni az ODL terveinket relációs tervekké. A 3.3. részben áttekintjük az E/K modellek relációs modellekké történő átírását. Ezután a 3.4. részben foglalkozunk az alosztályokkal. Minthogy az ODL és az E/K modellek különbözőképpen kezelik az alosztályokat (az öröklődési jellegű specializáló kapcsolatokat), így a relációkká történő átírásuk is egy kicsit eltér egymástól.

A megszorításokat sokszor az adatbázisséma részeként adjuk meg. Az ODL-ben vagy az E/K modellben szereplő megszorítások, például a kulcsmegszorítások és a hivatkozási épség típusú megszorítások a relációs modellben is kifejezhetők. A megszorítások egyik lényeges osztályát, a relációs modellben „funkcionális függőség"-ként ismert megszorításokat a 3.5. részben kezdjük el tárgyalni. A relációkon értelmezett egyéb megszorításokat a 4.5. résztől kezdődően tanulmányozzuk.

3.2.1. Attribútumok átírása

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 tegyük fel a lehető legegyszerűbb esetet, amikor az osztályok valóban átírhatók relációkká és a tulajdonságok pedig attribútumokká. Az alábbi megkötéseket tesszük fel:

1. Az osztályok összes tulajdonságai attribútumok (nem kapcsolatok vagy metódusok).

2. Az attribútumok atomi típusúak (nem összetett típusok vagy halmazok).

3.2. példa: A 3.2. ábra egy ilyen osztályra mutat egy példát. Négy attribútuma van és nincs más tulajdonsága. Mindegyik attribútuma atomi típusú, a `cím` karaktersorozat, az `év` és a `hossz` egészek, és a `szalagFajta` pedig két értékből álló felsorolt típusú.

Létrehozunk egy ugyanolyan nevű relációt, mint az osztály, ami jelen esetben a `Film` volt. A reláció négy attribútumból áll, az osztály minden attribútumához tartozik egy attribútum a relációban. A relációattribútumok nevei ugyanazok lehetnek, mint a megfelelő osztály attribútumnevei. Így ennek a relációnak a sémája

```
Film(cím, év, hossz, szalagFajta)
```

ahogyan a 3.1.1. részben ezt előre jeleztük.

Az osztály egy objektumának mind a négy osztályattribútumon egy értéke van. Egy ilyen objektumhoz az összes attribútumon felvett értékeiből, mint a sor komponenseiből készíthetünk sort. Ennek az átírásnak az eredményét már láttuk korábban. A 3.1. ábrán található példa néhány `Film` objektum sorrá való átírására. □

3.2.2. Összetett attribútumok

Sajnos még akkor is, ha az osztály tulajdonságai mind attribútumok, előfordulhatnak nehézségek az osztály relációra történő átírása közben. Ennek az oka az, hogy az ODL-ben az attribútumok összetett típusúak is lehetnek, mint például rekordszerkezetek, 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. Csak az lehet egyedül a probléma, hogy két struktúrának ugyanolyan nevű mezői lehetnek, ebben az esetben egy új attribútumnevet kell találnunk, hogy meg tudjuk különböztetni ezeket a relációban.

```
interface Színész {
   attribute string név;
   attribute Struct Cím
      { string város, string utca} lakcím;
};
```

3.5. ábra. *Egy összetett attribútummal rendelkező osztály*

3.3. példa: Tekintsük a `Színész` osztály előzetes definícióját a 2.3. példából, amelyet itt a 3.5. ábrán megismételtünk. A `név` attribútum atomi, a `lakcím` attribútum két, a `város` és `utca` mezőkből álló rekordszerkezet. Így ezt az osztályt egy háromattribútumos relációval tudjuk reprezentálni. Az első attribútum, a `név` megfelel az ugyanilyen elnevezésű ODL-attribútumnak. A második és a harmadik attribútumok

Felsorolások és dátumok reprezentálása

Az ODL-ben vannak olyan atomi típusok, speciálisan a felsorolások és a dátumok, amelyeket nem tudunk közvetlenül reprezentálni a szabványos relációs modellben. Ezek a típusok azonban nem jelentenek alapvető problémát. Például a felsorolás valójában az első néhány egészhez rendelt nevek listája. Így a hét napjaira vett ODL felsorolási típust egy egész típusú attribútummal reprezentálhatjuk, ahol a számok csak 0 és 6 közötti értékekre korlátozódnak. Másik megoldásként karaktersorozat típusú attribútum is használható, ahol a napokat, „Hétfő", „Kedd" stb. sztringek jelentik. Hasonlóan az ODL dátumait a relációs modellben karaktersorozat típusú attribútumokkal reprezentálhatjuk. Az 5. fejezetben tárgyaljuk az SQL lekérdezőnyelvet, amely éppúgy támogatja a felsorolt vagy dátum attribútumtípusokat, mint az ODL.

név	*város*	*utca*
`Carrie Fisher`	`Hollywood`	`Maple St. 123`
`Mark Hamill`	`Brentwood`	`Oak Rd. 456`
`Harrison Ford`	`Beverly Hills`	`Palm Dr. 789`

3.6. ábra. *Színészeket ábrázoló reláció*

`város` és `utca` nevet kapják, és megfelelnek a `lakcím` rekordszerkezet két mezőjének, a kettő együtt jelenti a lakcímet. Így a relációsémánk

```
Színész(név, város, utca)
```

Ennek a relációnak egy előfordulására a 3.6. ábrán láthatunk néhány lehetséges sorból álló példát. □

A rekordszerkezet persze nem a legbonyolultabb attribútumféle, amit az ODL osztálydefiníciókban használhatunk. A `Halmaz`, `Multihalmaz`, `Tömb` és `Lista` típuskonstruktorokkal is felépíthetünk értékeket. Mindegyik sajátos problémát jelent a relációs modellé transzformálás során. Részletesen csak a legáltalánosabbal, a `Halmaz` konstruktorral foglalkozunk.

Az egyik megközelítés, ahogy reprezentálhatjuk egy A attribútum halmazértékeit, hogy a halmaz minden értékéhez készítünk egy külön sort ezzel az A értékkel, és a sor többi attribútumához az objektum megfelelő értékeit vesszük fel. Először nézzük meg egy példán, hogyan működik helyesen ez a megközelítés, azután megnézzük a buktatóit.

3.4. példa: Tegyük fel, hogy a `Színész` osztályt úgy definiáltuk, hogy minden színészhez egy lakcímhalmazt tudunk bejegyezni. Az ODL osztálydefiníciót a 3.7. ábra tartalmazza. Tegyük fel azt, hogy Carrie Fishernek van egy tengerparti otthona is, de a 3.6. ábrán említett másik két színésznek csak egyetlen otthona van. Ekkor létrehozunk két olyan sort, amely a `név` attribútumon megegyezik a „`Carrie Fisher`" értékkel, amint azt a 3.8. ábra mutatja. A többi sor ugyanaz marad, mint a 3.6. ábrán volt. □

Az Olvasónak vigyáznia kell, hogy ezzel a technikával, amikor egy halmazt több sorra bont szét, nehogy rosszul tervezett relációt hozzon létre. A 3.7. részben fogjuk

Megjegyzés az adatok minőségéről :–)

Arra törekedtünk, hogy a példaadataink olyan pontosak legyenek, amennyire lehetséges, de a színészekre vonatkozó lakcímeknek és egyéb személyes adatoknak hamis értékeket vettünk fel, hogy megvédjük a színészszakmával foglalkozók magánéletét, mivel sokan közülük visszahúzódnak és elkerülik a nyilvánosságot.

```
interface Színész {
   attribute string név;
   attribute Set<
         Struct Cím {string város, string utca}
      > lakcím;
};
```

3.7. ábra. *Lakcímhalmazú színészek*

név	*város*	*utca*
Carrie Fisher	Hollywood	Maple St. 123
Carrie Fisher	Malibu	Locust Ln. 5
Mark Hamill	Brentwood	Oak Rd. 456
Harrison Ford	Beverly Hills	Palm Dr. 789

3.8. ábra. *Megengedünk lakcímhalmazokat*

áttekinteni a felmerülő problémákat, és később megtanuljuk, hogyan lehet újratervezni az adatbázisséмát. Most csak egy példán keresztül nézzük meg milyen problémák jelentkezhetnek.

3.5. példa: Tegyük fel, hogy a `születésnapot`, mint egy attribútumot hozzávesszük a `Színész` osztály definíciójához, azaz a 3.9. ábrán található definíciót használjuk. A 3.7. ábra definícióját kiegészítettük a `dátum (Date)` típusú `születésnap` attribútummal, amely egy ODL atomi típus. A `születésnap` attribútum a `Színész` reláció egy attribútuma lehet, amelynek a sémája így az alábbi:

```
Színész(név, város, utca, születésnap)
```

Az atomi értékek: előny vagy hátrány?

Úgy látszik, hogy a relációs modellben akadályokkal találkozunk utunkon, ugyanakkor az ODL-ben rugalmasabban lehetséges a strukturált értékeket tulajdonságokként tekinteni. Hajlamosak lehetünk teljes egészében elutasítani a relációs modellt, vagy csak egyszerű fogalomként tekinteni, amelyet már túllépett az elegánsabb „objektumorientált" szemlélet, mint amilyen például az ODL. Ezzel szemben az az igazság, hogy a relációs modellen alapuló adatbázisrendszerek a legelterjedtebbek a piacon. Ennek az egyik oka, hogy a modell egyszerűsége miatt elegáns és hatékony programozási nyelvekkel lehetséges az adatbázisok lekérdezése. A 4.1. és 4.2. részekben bevezetünk ilyen absztrakt programozási nyelveket, a relációs algebrát és a Datalogot. Ennél is fontosabb ezek beépülése az SQL-be, amely napjainkban az adatbázisrendszerek szabványnyelveként használatos.


```
interface Színész {
   attribute string név;
   attribute Set<
         Struct Cím {string város, string utca}
      > lakcím;
   attribute Date születésnap;
};
```

3.9. ábra. *Színészek lakcímhalmazzal és születésnappal*

név	*város*	*utca*	*születésnap*
Carrie Fisher	Hollywood	Maple St. 123	9/9/99
Carrie Fisher	Malibu	Locust Ln. 5	9/9/99
Mark Hamill	Brentwood	Oak Rd. 456	8/8/88

3.10. ábra. *A születésnapokkal kibővítve*

Változtassuk meg a 3.8. ábrán szereplő adatokat. Mivel a lakcímek halmaza üres is lehet, tegyük fel, hogy Harrison Fordnak nincsen lakcíme az adatbázisban. A módosított relációt a 3.10. ábrán láthatjuk.

Két helytelen dolog történt a 3.10. ábrán. Az első, hogy Carrie Fisher születésnapját megismételtük minden ilyen sorban, amely ezáltal a relációban tárolt információ redundanciáját eredményezte. Megjegyezzük, hogy a színésznő nevét is megismételtük, de ez az ismétlés nem valódi redundancia, mivel ha nem szerepelne a név, akkor nem tudnánk, hogy mindkét lakcím Carrie Fisherhez tartozik. A különbség, hogy a színész neve annak az objektumnak a kulcsa, amelyet a relációval reprezentálunk, és emiatt minden sorban szerepelnie kell, hogy azonosítsa a színészt. Ellentétben a születésnap olyan adat a színészről, amely nem része az objektumot reprezentáló kulcsnak, így a születésnap ismétlése redundáns.

A másik probléma, hogy mivel Harrison Fordnak üres a lakcímhalmaza, ezért az összes információt elveszítettük róla. Speciálisan a reláció nem tartalmazza a születésnapját, pedig ez benne van Ford esetén a `Színész` objektumban. Megjegyezzük újból, hogy ezek közül a problémák közül egyikkel sem követtünk el végzetes hibát az ODL sémáknak relációsémákká történő átírási módszerünkkel. Csak legyünk tudatában az ilyen jellegű problémáknak. Később a 3.7. részben következő „normalizáció" módszerével fogjuk rendbe tenni a relációsémát. □

Ha több kollekció típusú osztályattribútum van (amelyeket többértékű attribútumoknak fogunk hívni), akkor egyetlen objektum reprezentálásához szükséges sorok száma megsokszorozódik. Ekkor a többértékű attribútumok értékeinek minden kombinációjához létre kell hoznunk egy sort. A 3.2.5. részben visszatérünk erre a problémára a kollekció típusú kapcsolatokkal kapcsolatban.

3.2.3. Egyéb típuskonstruktorok reprezentálása

A rekordszerkezeteken és halmazokon kívül az ODL osztálydefiníció használhat `Multihalmaz`, `Tömb` vagy `Lista` 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, hanem a relációsémát bővítjük egy újabb attribútummal, amellyel azt reprezentáljuk az egyes elemek hányszor fordulnak elő a multihalmazban.[1] Például tegyük fel, hogy a 3.7. ábrában a `lakcím` halmaz helyett multihalmaz szerepelne. Legyen például Carrie Fisher `Hollywood, Maple St. 123` lakcíme kétszer, `Malibu, Locust Ln. 5` lakcíme pedig háromszor. Ekkor ezt az alábbi reláció reprezentálja:

név	*város*	*utca*	*szám*
Carrie Fisher	Hollywood	Maple St. 123	2
Carrie Fisher	Malibu	Locust Ln. 5	3

A lakcímek listája egy új attribútummal, a `pozícióval` reprezentálható, amely megmutatja, hogy az adott lakcím hanyadikként 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	Maple St. 123	1
Carrie Fisher	Malibu	Locust Ln. 5	2

Végül, hà a lakcímek fixhosszúságú 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	Maple St. 123	Malibu	Locust Ln. 5

[1] Az absztrakt relációs modellnek az a leírása, amit ebben a fejezetben adtunk meg, nem engedi meg, hogy egy relációba teljesen megegyező sorok kerüljenek be. Ugyanakkor nem árt tudni, hogy 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 a 4.6. és 5.4. részekben olvashatunk bővebben. Ha a sorok száma lényeges szempont, akkor azt ajánljuk, hogy a fentiekben megadott sémát használjuk még akkor is, ha az adatbázis-kezelőnk meg is engedné az azonos sorokat.

3.2.4. Egyértékű kapcsolatok reprezentálása

Általában egy ODL osztálydefiníció tartalmazza a többi ODL osztállyal való kapcsolatokat is. Például nézzük meg a 2.6. ábrán szereplő `Film` osztály teljes definícióját, amelyet a 3.11. ábrán újból megismételtünk.

```
interface Film {
   attribute string cím;
   attribute integer év;
   attribute integer hossz;
   attribute enumeration Szalag
     {színes, feketeFehér} szalagFajta;
   relationship Set<Színész> szereplők
     inverse Színész::szerepelBenne;
   relationship Stúdió gyártó
     inverse Stúdió::gyárt;
};
```

3.11. ábra: *A `Film` osztály teljes definíciója*

Először nézzük meg a `gyártó` kapcsolatot, amely minden filmhez hozzákapcsolja azt a stúdiót, ahol gyártották a filmet. Az első gondolatunk az lehet, hogy egy kapcsolat olyan, mint egy attribútum. Készíthetünk egy reláció-attribútumot vagy attribútumokat, amivel a kapcsolt osztály objektumait reprezentálhatjuk, hasonlóan ahhoz, ahogyan egy struktúra vagy struktúrahalmaz értékű ODL-attribútumot reprezentáltunk. A `gyártó` kapcsolat esetén így a `Film` relációsémához hozzáveszünk egy-egy attribútumot a `Stúdió` osztály minden tulajdonságához.

Ebben a megközelítésben az egyetlen probléma a `Stúdió` objektumok `gyárt` tulajdonsága, amely visszafelé ad kapcsolatot a `Film` osztályhoz. Ahhoz, hogy reprezentáljuk ezt a kapcsolatot, minden filmsornak tartalmaznia kellene információt az összes többi filmről, amelyet az adott stúdió gyárt. A filmekkel kapcsolatos információba így bele kell értenünk a gyártó stúdiót is, ami arra késztet bennünket, hogy újból belevegyük ennek a stúdiónak az összes filmjeiről szóló információt. Világos, hogy ez a megoldás nagy nehézséget jelent, ha egyáltalán végrehajtható.[2]

Egy jobb megközelítést találunk, ha arra gondolunk, hogyan tárolódnak valójában az objektumok a számítógép memóriájában. Ha az O_1 objektum tartalmaz egy másik O_2 objektumra történő hivatkozást, akkor nem másoljuk bele az O_2-t az O_1-be, hanem inkább az O_2-re mutató „mutatót" használunk O_1-en belül.

[2] A filmek és stúdiók esetében kialakuló lánc valójában még nem is jelentene akkora problémát, mivel nem vezetne ki egy adott stúdió által gyártott filmek köréből. Ezzel szemben, ha ugyanezt a `szereplők` és a `szerepelBenne` inverz kapcsolatokra alkalmaznánk, akkor egy filmből kiindulva először eljutnánk a film szereplőihez, majd azokhoz a filmekhez, amelyekben ezek a színészek játszanak, aztán ezeknek a filmeknek a szereplőihez és így tovább. Végül rossz esetben majdnem minden film és színész bekerülne a kialakuló láncba.

A relációs modell viszont nem tartalmaz mutatót vagy bármilyen mutatóra emlékeztető fogalmat. Ehelyett nekünk kell szimulálnunk a mutatók hatását olyan értékekkel, amelyek a kapcsolatban álló objektumokat reprezentálják. A kapcsolódó osztály olyan attribútumhalmazára van szükségünk, amely kulcsot alkot. Ha találunk egy ilyen halmazt, akkor a kapcsolatot úgy kezeljük, mint a kapcsolódó osztály kulcsattribútumát vagy attribútumait. Egy példán keresztül mutatjuk be ezt a technikát.

3.6. példa: Tegyük fel, hogy a `név` kulcs a `Stúdió` osztályban, amelynek az ODL definícióját a 2.6. ábráról átmásoltuk:

```
interface Stúdió {
   attribute string név;
   attribute String cím;
   relationship Set<Film> gyárt
      inverse Film::gyártó;
};
```

Módosíthatjuk a 3.1. ábrán levő `Film` relációsémánkat úgy, hogy tartalmazzon egy olyan attribútumot, amellyel a gyártó stúdiót reprezentáljuk. Legyen például a `stúdióNév` ez az attribútum. A 3.12. ábrán látjuk a hozzávett `stúdióNév` attribútumot és néhány mintasort. □

cím	*év*	*hossz*	*szalagFajta*	*stúdióNév*
Csillagok háborúja	1977	124	színes	Fox
Rút kiskacsa	1991	104	színes	Disney
Wayne világa	1992	95	színes	Paramount

3.12. ábra. *A `Film` reláció a gyártó stúdiót jelentő új attribútummal*

3.2.5. Többértékű kapcsolatok reprezentálása

A 3.11. ábrán a `szereplők` kapcsolat olyan problémát jelent, amilyennel korábban nem találkoztunk a `gyártó` kapcsolatnál. Ha a kapcsolat típusa egy osztály, akkor azt mondjuk, hogy a kapcsolat *egyértékű*. A `gyártó` kapcsolat példa egy egyértékű kapcsolatra, aminek a típusa a `Stúdió`. Ha pedig a kapcsolat típusa egy osztályra alkalmazott valamely kollekciótípus, akkor azt mondjuk, hogy a kapcsolat *többértékű*. Például a `szereplők` többértékű kapcsolat, mivel a típusa `Set<Színész>`. Más szavakkal, az A osztálytól a B osztályra való bármely egy-sok vagy sok-sok kapcsolat többértékű kapcsolat A-ról B-re.

A többértékű kapcsolat reprezentálásához az alábbi két technikát kell kombinálnunk:

1. Ugyanúgy, mint az egyértékű kapcsolatoknál, minden kapcsolt objektumhoz keresnünk kell egy kulcsot.

2. Ugyanúgy, mint a halmazértékű attribútumoknál, a kapcsolt objektumokat úgy kell reprezentálnunk, hogy minden értékhez létrehozunk egy sort. Megint ugyanúgy, mint a halmazértékű attribútumoknál, ez a megközelítés megengedi a redundanciát, mivel a reláció többi attribútumainak az értékeit a halmaz minden elemére egyszer megismételjük. Ezt a problémát a 3.7. részben oldjuk meg, de ebben a pillanatban elfogadjuk ezt a hibát.

3.7. példa: Tegyük fel, hogy a `Színész` osztályban a `név` kulcs. Ekkor kibővíthetjük a `Film` osztályhoz tervezett relációt egy új attribútummal, amelyet nevezzünk `színészNévnek`, és amely minden filmhez tartalmazza a szereplők neveit. Egy filmet emiatt annyi sor reprezentál, amennyi szereplőt felsoroltunk hozzá az adatbázisban. Néhány példasort láthatunk a 3.13. ábrán. Megjegyezzük, hogy ekkor redundancia lép fel, mivel minden filmhez a film minden színészére az összes többi információt megismételjük egyszer. □

cím	*év*	*hossz*	*szalagFajta*	*stúdióNév*	*színészNév*
Csillagok háborúja	1977	124	színes	Fox	Carrie Fisher
Csillagok háborúja	1977	124	színes	Fox	Mark Hamill
Csillagok háborúja	1977	124	színes	Fox	Harrison Ford
Rút kiskacsa	1991	104	színes	Disney	Emilio Estevez
Wayne világa	1992	95	színes	Paramount	Dana Carvez
Wayne világa	1992	95	színes	Paramount	Mike Meyers

3.13. ábra. *A `Film` reláció a szereplőkre vonatkozó információval kiegészítve*

Egy osztálynak esetenként egynél több többértékű kapcsolata van. Ebben az esetben a feltárt osztály egyetlen objektumának reprezentálásához számos sor szükséges. Tegyük fel, hogy egy C osztály többértékű kapcsolatai R_1, R_2, ..., R_m. Ekkor a C osztályhoz rendelt reláció attribútumai: C minden attribútumához tartozó attribútumok, C minden egyértékű kapcsolatának a kulcsait reprezentáló attribútumok, valamint R_1, R_2, ..., R_m kapcsolatok célosztályainak a kulcsait reprezentáló attribútumok.

Tegyük fel, hogy a C osztály egy speciális o objektuma az R_1 kapcsolaton keresztül n_1 számú objektumhoz kapcsolódik, az R_2 kapcsolaton keresztül n_2 számú objektumhoz kapcsolódik stb. Ekkor minden egyes olyan választáshoz, melynek során az R_1, R_2 stb. kapcsolatokon keresztül választunk egy-egy o objektummal kapcsolatban álló objektumot, létre kell hoznunk egy o-nak megfelelő sort. Így ehhez az objektumhoz $n_1 \times n_2 \times \ldots \times n_k$ darab sort kell készítenünk a C osztályhoz konstruált relációban.

3.8. példa: Legyen egy C osztály egyértékű attribútumainak halmaza X, és két többértékű kapcsolata az R_1 és az R_2. Tegyük fel, hogy ezek a kapcsolatok a C osztályhoz olyan osztályokat kapcsolnak, amelyeknek a kulcsattribútumai Y, illetve Z halmazok. Most vegyük a C osztály egy c objektumát, amely az R_1 kapcsolaton keresztül az y_1 és y_2 kulcsú objektumokhoz kapcsolódik, az R_2 kapcsolaton keresztül pedig a z_1, z_2 és z_3 kulcsú objektumokhoz kapcsolódik. Az x reprezentálja a c objektum értékeit az X attribútumhalmazon.

Ekkor a C osztályból létrehozott relációban a c objektumot hat sor reprezentálja. Ezek a sorok a következők:

(x, y_1, z_1) (x, y_1, z_2) (x, y_1, z_3)
(x, y_2, z_1) (x, y_2, z_2) (x, y_2, z_3)

Azaz Y kulcsait párosítjuk Z kulcsaival az összes lehetséges módon. □

3.2.6. Mit tegyünk, ha nincs kulcs?

Az objektumorientált modellekben, mint az ODL, megengedett, hogy egy osztályban két objektum minden tulajdonságon pontosan megegyezzen. Így fel kell készülnünk, hogy megbirkózzunk azzal a problémával, amikor két színésznek ugyanaz a neve. Ha így van, akkor a `név` nem kulcs a `Színész` osztályban, és nem használhatjuk a `Film` reláció soraiban a szereplők azonosítására. Hozzávehetünk talán a színészek további attribútumaiból párat, hogy kulcsot kapjunk, de elvileg semmi sem biztosítaná, hogy ne lehetne két ugyanolyan nevű színész, akik ugyanazon a napon születtek, ugyanaz a lakcímük stb., szóval a színészek az adatbázisban tárolt más tulajdonságokon is megegyezhetnek.

Az egyetlen megoldás, ami biztosan jól működik, hogy felveszünk egy új attribútumot a megfelelő relációban, amely az osztály objektumainak „objektumazonosítóját" reprezentálja. Például, ha nem biztos, hogy a név a színészekhez kulcs, akkor minden színészhez felveszünk egy „azonosítószámot", amely lehet a színésznek a Színészek Egyesületében kapott tagsági száma. Az azonosítószámok egyértelműek, mivel egyetlen központi hivatal felelős ezek kezeléséért, amely nyomon követi, milyen számokat használtak eddig.

3.9. példa: Ha elfogadjuk ezt a megközelítést, hogy egyértelmű azonosítószámot adunk minden színésznek, és ezt az azonosítószámot használjuk kulcsként a színész reprezentálására a kapcsolatokban, akkor a `Film` reláció a következő formájú:

cím	*év*	*hossz*	*szalagFajta*	*stúdióNév*	*azon#*
`Csillagok háborúja`	`1977`	`124`	`színes`	`Fox`	`12345`

Itt csak egyetlen sort adtunk meg, feltettük, hogy Carrie Fisher azonosítószáma 12345. A `Színész` relációnak is tartalmaznia kell az azonosítószámot, a színészek többi olyan információjával együtt, amelyet a `Színész` ODL osztálydefiníciója alapján kaptunk. Például

azon#	*név*	*város*	*utca*	*szerepelt*
`12345`	`Carrie Fisher`	`Hollywood`	`Maple St. 123`	`Csillagok háborúja`

javaslat a relációsémára és a sorok közül Carrie Fisher adatait megadó sorra. □

3.2.7. Kapcsolat és inverzének reprezentálása

Elméletben, ha közvetlenül alakítjuk át az ODL terveket relációs modellre, akkor egy kapcsolatot kétszer reprezentálunk, mindkét irányban egyszer. Így a 3.7. példában egy film minden szereplőjét tároltuk egy-egy olyan sorban, amelyben ugyanennek a filmnek a címe szerepelt. Ha a `Színész` osztályhoz tervezünk egy relációt, akkor a `szerepelBenne` kapcsolatot reprezentálnánk úgy, hogy minden színészhez létrehozunk annyi sort, ahány filmben szerepelt, és ezekbe a sorokba mindegyik filmnek bejegyezzük a címét és az évet (ugyanis a filmekhez a `cím` és az `év` együtt jelentette a kulcsot).

Azonban redundanciát jelent, ha mindkettőt reprezentáljuk, azaz a `szereplők` kapcsolatot és ennek az inverzét is. Az egyik már megadja az összes olyan információt, mint amit a másik nyújt. Így törölhetjük a `szerepelBenne` információt a `Színész` relációból. Vagy meghagyhatjuk ezt az információt a `Színész` relációban és töröljük a `színészNév` attribútumot a `Film` relációból. A 3.3.2. részben látunk egy harmadik megoldást is, amikor a kapcsolatot és az inverzét külön relációban reprezentáljuk. Gyakran, persze nem minden esetben, ez a megoldás a legelőnyösebb. Amikor az a legjobb választás, hogy a kapcsolat külön relációban legyen, akkor a 3.7. részben később tárgyalt „normalizáció" folyamattal is arra kényszerülünk, hogy különválasszuk a kapcsolatot egy saját relációba, még ha ezt nem is tettük volna a kezdeti relációs tervezésben.

Megjegyezzük, hogy az ODL modellben mind a kapcsolatra, mind pedig az inverzére szükségünk van, mivel ebből következnek azok a mutatók, amelyek a filmekről a szereplőkre és a színészekről a filmjeikre mutatnak. Nem lehet a mutatókon „visszafelé" haladni, emiatt mindkét irányban szükségesek a mutatók.[3] A relációs modellben

Kapcsolatok egyirányú reprezentálása

Ha bináris kapcsolat van két egyedhalmaz között, megválaszthatjuk, hogy melyik relációba írjuk be a kapcsolatot. Lényeges, hogy melyiket választjuk? Ha a kapcsolat sok-sok vagy egy-egy, akkor valószínűleg nem. Ha pedig a kapcsolat sok-egy, akkor azt javasoljuk, hogy a „több"-höz írjuk a kapcsolatot, azaz ahhoz az egyedhalmazhoz, amelyikből több kapcsolódhat a másik egyedhalmaz egy egyedével. Ennek az a célja, hogy elkerüljük a redundanciát.

Például a *Filmek* és *Stúdiók* közötti *Gyártók* kapcsolatot legjobb, ha a *Filmek*hez írjuk be. Azaz a *Filmek* relációban megadunk egy attribútumot a gyártó stúdió nevéhez. Ellenkezőleg, ha a *Stúdiók* relációba írnánk be a kapcsolatot, akkor minden egyes stúdiósorból több sort kellene készítenünk, minden egyes filmhez, amelyet a stúdió gyárt, egyet-egyet. Ami azt eredményezné, hogy minden stúdió esetén minden egyes filmhez megismételnénk az összes többi információt.

[3] Természetesen nem lehetünk biztosak abban, hogy az adott ODL megvalósítás ezeket a mutatókat úgy fogja reprezentálni, ahogy várnánk. Az is lehet, hogy ténylegesen egyetlen mutatóval reprezentálja az inverz kapcsolatokból álló párokat.

Mutatók: előnyösek vagy hátrányosak?

Az ODL kapcsolatokat feltételezhetően minden objektumtól a kapcsolt objektumra vagy objektumokra történő mutatókkal vagy hivatkozásokkal implementálják. Ez az implementáció nagyon kényelmes, mert lehetővé teszi, hogy egy objektumtól gyorsan eljussunk a kapcsolt objektumokig. Összehasonlítva, a relációs modellben, az „objektumokat" a kulcsértékeikkel reprezentáljuk mutatók helyett, ami úgy tűnhet, hogy lassabb navigáció egy objektumtól a kapcsolt objektumokig.

Például a `Film` objektumot úgy reprezentáltuk a 3.7. példában, hogy egy filmben szereplő minden színészre készítettünk egy sort, amely a színész nevét tartalmazta csak, nem az összes információt a színészről. Ha meg akarjuk keresni *Wayne világa* színészeinek lakcímét, akkor vennünk kell minden színésznevet és megkeressük a `Színész` relációban a színész sorát vagy sorait.

Azt gondolhatnánk emiatt, hogy a mutatók hiánya a relációs modellben a modell „hibája". A gyakorlatban olyan indextáblákat készíthetünk a relációkhoz, amelyekben sokkal hatékonyabban kereshetjük meg az adott komponensen adott értékű sorokat (lásd 1.2.1. és 5.7.7. részeket). Emiatt gyakorlatilag csak keveset veszítünk, hogy nem használunk mutatókat. Továbbá a mutatók inkább a központi memóriában futó programokban hasznosak, amelyek csak a végrehajtásuk alatt nagyon rövid ideig léteznek, de az adatbázisok nagyban különböznek az ilyen programoktól. Az objektumok évekig léteznek és több másodlagos tárolási eszközön vannak elosztva, amelyek szélesen osztott számítógépes rendszerekhez kapcsolódnak, az ilyen objektumok közötti mutatók implementálása rendkívül bonyolult. Így a relációs modell mutatómentes megközelítése nagyon előnyös.

pedig, mint az E/K modellben a kapcsolatokat összetartozó értékek (a kulcsok) fejezik ki. A sorok tartalmazzák az összekapcsolódó kulcspárokat, például a film címét és évét és a színész nevét, amit bármelyik irányban felhasználhatunk a kapcsolat követésében.

3.2.8. Feladatok

3.2.1. feladat: Alakítsuk át az alábbi feladatok ODL terveit relációs adatbázissémákká.

* a) 2.1.1. feladat.

b) 2.1.2. feladat (mind a négy változatára, amit ebben a feladatban megadtunk).

c) 2.1.3. feladat.


```
interface Vevők {
   attribute Struct Név
      { string vezetéknév, string utónév} név;
   attribute Set<
         Struct Telefon { string típus, int szám}
      >telefonszámok;
   relationship Vevők hivatkozik inverse hivatkozások;
   relationship Set<Vevők> hivatkozások
         inverse hivatkozik;
};
```

3.14. ábra. *A Vevők rekordok*

d) 2.1.4. feladat.

* e) 2.1.5. feladat.

f) 2.1.6. feladat.

3.2.2. feladat: Alakítsuk át a 2.7. ábra ODL leírását egy relációs adatbázissémává. Hogyan befolyásolja a 2.1.8. feladat háromféle módosítása a relációsémát?

! **3.2.3. feladat:** A 3.14. ábrán találjuk a vevők osztály ODL definícióját. Ennek az osztálynak az objektumaiban tartjuk a különféle típusú (pl. otthoni, munkahelyi, fax) telefonszámokat és a „hivatkozások" halmazát, amelyben a vevő kreditjei vannak, hogy más vevőket is beszervezett az üzletbe. Alakítsuk át ezt az ODL tervet egy relációs adatbázissémává.

3.3. E/K diagram átírása relációs modellé

Az egyed/kapcsolat diagram átírása relációs adatbázissémára hasonló, mint az ODL tervek átírása volt adatbázissémára. Bizonyos értelemben az E/K modell egy közbülső formát jelent az objetumorientált tervezés és a relációs tervezés között. Így egy E/K diagramból kiindulva a munka nagy része már készen van. Három lényeges különbség van:

1. Az E/K modellben a kapcsolatokat külön fogalmakként leválasztjuk az osztályokról ahelyett, hogy belevennénk az osztályokba tulajdonságként. Ebből a különbségből adódik, hogy elkerüljük a 3.2.2. részben talált redundanciát, amikor egy többértékű kapcsolatot, mint a `szerepelBenne`, akartunk a `Film` objektum soraiban reprezentálni.

2. Az ODL-ben az attribútumok bármilyen kollekciótípusúak lehettek, mint például `<Set>`. Az E/K modell valamennyire bizonytalan abban, milyen típusokat engedünk meg, általában úgy tekintjük, hogy lehetségesek strukturált értékek, de nem hal-

mazok vagy másféle kollekciótípus konstruktorok.[4] Ez eredményezi, hogy a halmazértékű attribútumokat, mint a 3.4. példában látott színész lakcímeinek halmazát az E/K modellben egyedhalmazokként kezeljük, és definiálunk egy `Lakik` kapcsolatot, amivel hozzákapcsoljuk a színészekhez a lakcímeiket.

3. Az E/K modellben a kapcsolatoknak is lehetnek attribútumaik. Az ODL-ben nincs ennek megfelelő fogalom.

3.3.1. Egyedhalmazok átírása relációkká

Először vegyünk egy olyan egyedhalmazt, amely nem gyenge. A gyenge egyedhalmazok kezelésére szükséges módosításokat később a 3.3.3. részben fogjuk megnézni. Minden nem gyenge egyedhalmazhoz létrehozunk egy relációt ugyanezzel a névvel és ugyanezzel az attribútumhalmazzal.* Ebben a relációban nincsenek jelölve, hogy mely kapcsolatokban vesz részt az egyedhalmaz, a kapcsolatokat külön relációkban fogjuk kezelni, ahogyan ezt a 3.3.2. részben tárgyaljuk.

3.10. példa: Vegyük a 2.8. ábra `Filmek`, `Színészek` és `Stúdiók` egyedhalmazait, amelyet itt újból megismétlünk a 3.15. ábrán. A `Filmek` egyedhalmaz attribútumai a `cím`, `év`, `hossz`, `szalagFajta`. A relációra átírt eredmény, a `Filmek` reláció olyan, mint a 3.1. ábra `Film` relációja, amivel a 3.1. szakasz kezdődött. □

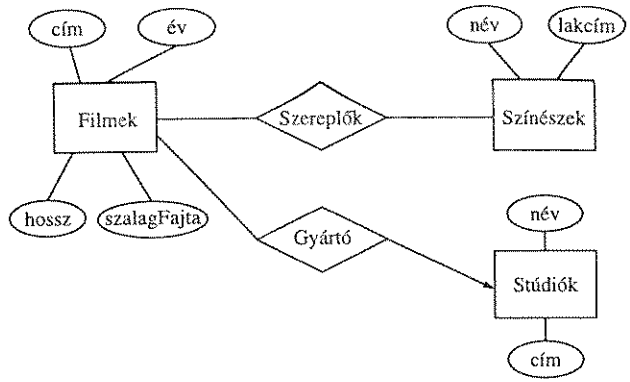
3.11. példa: Most vegyük a 3.15. ábra `Színészek` egyedhalmazát. Két attribútuma van, a `név` és a `lakcím`. Így a megfelelő `Színészek` relációt az alábbi formában kapjuk

név	*lakcím*
`Carrie Fisher`	`Hollywood, Maple St. 123`
`Mark Hamill`	`Brentwood, Oak Rd. 456`
`Harrison Ford`	`Beverly Hills, Palm Dr. 789`

Ez a reláció a 3.6. ábra `Színész` relációjára emlékeztet, amit a 3.3. példában hoztunk létre. Abban három attribútum volt, amiből kettő, a `város, utca` a strukturált lakcímet jelentette. Csak kicsi a különbség. Ha a 2.8. ábrát úgy terveztük volna, hogy a `Színészek` egyedhalmaznak legyenek `város` és `utca` az attribútumai, akkor az ennek megfelelő `Színészek` reláció pontosan úgy nézne ki, mint a 3.6. ábra `Színész` relációja. □

[4] Megjegyezzük, hogy az E/K modell formális leírásában az attribútumok típusára pontosan ugyanazok a korlátozások érvényesek, mint az ODL-ben, azaz lehetnek struktúrák kollekciói vagy ennél egyszerűbb típusok.

* Szerkesztői megjegyzés: az egyedeknek megfeleltetjük az attribútumaihoz rendelt értékekből álló sort. Az így keletkező relációelőfordulás az (erős) egyedhalmaz kulcsa miatt nem tartalmazhat két azonos sort, tehát korrekt.



3.15. ábra. *A film adatbázis E/K diagramja*

3.3.2. E/K kapcsolatok átírása relációkká

Az E/K modellben a kapcsolatokat szintén relációkkal reprezentáljuk. Egy adott kapcsolathoz rendelt *R* reláció a következő attribútumokat tartalmazza:

1. Az *R* kapcsolatban részt vevő minden egyedhalmazra belevesszük az *R* reláció sémájába a kulcs attribútumot vagy attribútumokat.

2. Ha a kapcsolatnak vannak attribútumai, akkor ezek az *R* reláció attribútumai lesznek.

Ha valamelyik egyedhalmaz többszörösen benne van a kapcsolatban, akkor át kell neveznünk az attribútumait, hogy elkerüljük a nevek ismétlődését egy sémán belül. Hasonlóan, ha ugyanaz az attribútumnév kétszer vagy többször szerepel magának az *R*-nek és az *R* kapcsolatban részt vevő egyedhalmazoknak az attribútumai között, akkor át kell neveznünk, hogy elkerüljük az ismétlődést.

3.12. példa: Most vegyük a 3.15. ábra `Gyártó` kapcsolatát. Ez a kapcsolat a `Filmek` és a `Stúdiók` egyedhalmazokat kapcsolja össze. Így a `Gyártó` relációsémájában használjuk a `Filmek` kulcsát, azaz `cím` és `év` attribútumokat, és a `Stúdiók` kulcsát, a `név` attribútumot. Erre a relációra egy példa a következő:

cím	*év*	*stúdióNév*
`Csillagok háborúja`	`1977`	`Fox`
`Rút kiskacsa`	`1991`	`Disney`
`Wayne világa`	`1992`	`Paramount`

A világosság kedvéért itt a `stúdióNév` attribútumot választottuk, ami megfelel a `Stúdiók` reláció `név` attribútumának.

Vegyük észre, hogy a fenti reláció meg a 3.1. példában a `Filmek` egyedhalmazból előállított reláció pontosan ugyanazt az információt tartalmazza, mint a 3.6. példában a `Film` osztályból a `szereplők` tulajdonság kizárásával előállított 3.12. ábrán szereplő reláció. □

3.13. példa: Hasonlóan írjuk át a 3.15. ábra `Szereplők` kapcsolatát relációvá, amelynek attribútumai `cím` és `év` (`Filmek` kulcsa) és a `színészNév`, amely a `Színészek` egyedhalmaz kulcsa. A 3.16. ábrán a `Szereplők` relációra találunk egy példát. Megjegyezzük, hogy ez a reláció és a 3.1. ábra együttesen tartalmaz minden olyan információt, mint amit a 3.13. ábra, de nem ismétli meg a `Színész` osztály nem kulcsbeli attribútumait (`hossz` és `szalagFajta` attribútumokat), ami a 3.13. ábra hibája volt.

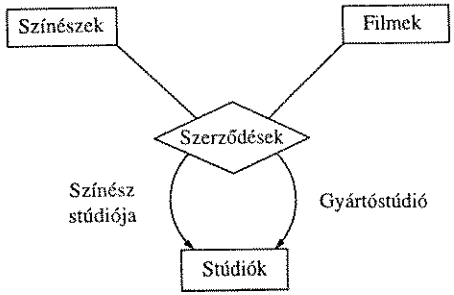
Úgy tűnik, hogy az év redundáns a 3.16. ábrán, de ez csak amiatt van, mert ezek a filmcímek egyediek. Ha lett volna több „`King Kong`" című film, akkor látnánk, hogy az év lényeges, hogy kiválassza melyik színész melyik filmváltozatban szerepelt. □

cím	*év*	*színészNév*
Csillagok háborúja	1977	Carrie Fisher
Csillagok háborúja	1977	Mark Hamill
Csillagok háborúja	1977	Harrison Ford
Rút kiskacsa	1991	Emilio Estevez
Wayne világa	1992	Dana Carvezy
Wayne világa	1992	Mike Meyers

3.16. ábra. *A `Szereplők` kapcsolat relációja*

Felhívjuk a figyelmet az adatbázissémáknak azokra az előnyeire, amit E/K diagramból kiindulva kaphatunk meg, összehasonlítva az ODL tervekből való átírással.

- A relációk legtöbbször „normalizáltak" abban az értelemben, hogy elkerülik azoknak a redundanciáknak nagy részét, amelyek az ODL leírásokból közvetlenül átírt tervekben jelen vannak.



3.17. ábra. *A `Szerződés` kapcsolat*

Visszatérve az ODL-relációátírásra

Láttuk, hogy az E/K modell kapcsolatok relációkká való átírásával jobb relációs adatbázissémát kaptunk, mint amikor az ODL kapcsolatokat írtuk át relációkká. Természetesen ekkor is választhatjuk azt az E/K technikát, hogy leválasztjuk a sok-egy és sok-sok kapcsolatokat külön relációkban. Ezzel elkerülhetjük a redundanciát és a sorok számának a gyors gyarapodását, ami akkor fordulhatna elő, amikor egy osztályt egy többértékű kapcsolatával együtt próbálunk implementálni. A 3.7. részben tárgyaljuk azt a technikát, hogyan tudjuk a közvetlenül az ODL-ből átírt relációsémákat mechanikus úton kijavítani.

- A kétirányú ODL kapcsolatokat egyetlen relációval helyettesítjük, amely mindkét irányban reprezentálja a kapcsolatot.

3.14. példa: A többértékű kapcsolatok szintén könnyen átírhatók relációkká. Tekintsük a 2.12. ábra négyirányú *Szerződés* kapcsolatát, amelyet itt a 3.17. 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 ehhez a filmhez kötött szerződést. Ezt a kapcsolatot a `Szerződés` relációval reprezentáljuk, amelynek a sémáját az alábbi négy egyedhalmaz kulcsaiban szereplő attribútumok alkotják:

1. `színészNév`, a színész kulcsa.
2. `cím` és `év`, a film két attribútumából álló kulcsa.
3. `színészStúdió` kulcs jelzi az első stúdió nevét, korábban feltettük, hogy a `Stúdió` egyedhalmaz kulcsa a stúdió neve.
4. `gyártóStúdió` kulcs jelzi annak a stúdiónak a nevét, amelyik elkészíti a filmet a szóban forgó színésznek a közreműködésével.

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 színésznévre vagy a stúdiónévre, és az utóbbi esetben pedig melyik stúdió nevére vonatkozik. □

3.3.3. 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 a *W* attribútumait, hanem más egyedhalmazok kulcsattribútumait is, ame-

lyek segítik a *W* kulcsának kialakítását. Ezeket a segítő egyedhalmazokat könnyen felismerhetjük arról, hogy a *W*-hez dupla keretes rombusszal kapcsolódnak.

2. Bármely olyan kapcsolatban, amelyben a *W* gyenge egyedhalmaz részt vesz, a *W* kulcsának az összes kulcsattribútumát használjuk, azokat az egyéb egyedhalmazokbeli attribútumokat is beleértve, amelyek hozzájárultak a *W* kulcsához.

3. Látni fogjuk, hogy a dupla keretes rombuszkapcsolatokat a *W* gyenge egyedhalmazról valamely más egyedhalmazra, vagyis amelyek segítek biztosítani a *W* kulcsát, egyáltalán nem kell relációkká átírnunk. Ez amiatt van így, mert az ilyen kapcsolat attribútumai mindig magának a *W* gyenge egyedhalmaz attribútumainak részhalmazát képezik, és így ezek a kapcsolatok nem nyújtanak további információt azon túl, hogy segítenek a *W* kulcsát megtalálni.

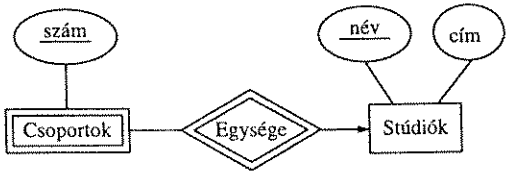
Természetesen, amikor ezeket a további attribútumokat felvesszük egy gyenge egyedhalmaz kulcsába, akkor vigyáznunk kell, hogy ne használjuk ugyanazt a nevet kétszer. Ha szükséges, nevezzük át az összes ilyen attribútumot.

3.15. példa: Tekintsük a `Csoportok` gyenge egyedhalmazt a 2.27. ábrán, amelyet a 3.18. ábrán megismételünk. Ebből a diagramból három relációt kapunk, amelyeknek a sémái:

```
Stúdiók(név, cím)
Csoportok(szám, stúdióNév)
Egység(szám, stúdióNév, név)
```

Az első relációt, a `Stúdiókat` természetes módon írtuk át az azonos nevű egyedhalmazból. A második reláció a `Csoportok` gyenge egyedhalmazból származik. Ennek a relációnak az attribútumai a *Csoportok* kulcsattribútumai, ha pedig lenne a *Csoportok*nak valamilyen nem kulcsattribútuma is, akkor ezeket is belevennénk a relációsémába. A `Csoportok` relációban a `stúdióNevet` választottuk attribútumként, amely megfelel a *Stúdiók* egyedhalmaz `név` attribútumának.

A harmadik reláció, az `Egysége`, az azonos nevű kapcsolatból származik. Mint mindig, egy E/K kapcsolatot a relációs modellben egy olyan relációval reprezentálunk, amelynek sémája a kapcsolt egyedhalmazok kulcsaiból áll. Ebben az esetben az `Egysége` attribútumai a `szám` és `stúdióNév`, a *Csoportok* gyenge egyedhalmaz kulcsa, és a `név` attribútum, a *Stúdiók* kulcsa. Megjegyezzük, hogy mivel az *Egysége* sok-egy kapcsolat, a stúdió `stúdióNév` biztos, hogy megegyezik a stúdió `névvel`.



3.18. ábra: *A Csoportok, avagy példa gyenge egyedhalmazra*

Például tegyük fel, hogy a hármas számú csoport (#3) a Disney-csoportok közül az egyik. Ekkor az *Egysége* kapcsolathalmaz tartalmazza a következő párt

(Disney-csoport #3, Disney)

Ez a pár az alábbi `Egysége` sort adja

(3, Disney, Disney)

Ennek a következménye, hogy „egybeolvashatjuk" az `Egysége, stúdióNév` és `név` attribútumokat, megadva a következő sémát:

```
Egysége(szám, név)
```

Tulajdonképpen el is hagyhatjuk az `Egysége` relációt, mivel az összes attribútuma szerepel a `Csoportok` reláció attribútumai között. □

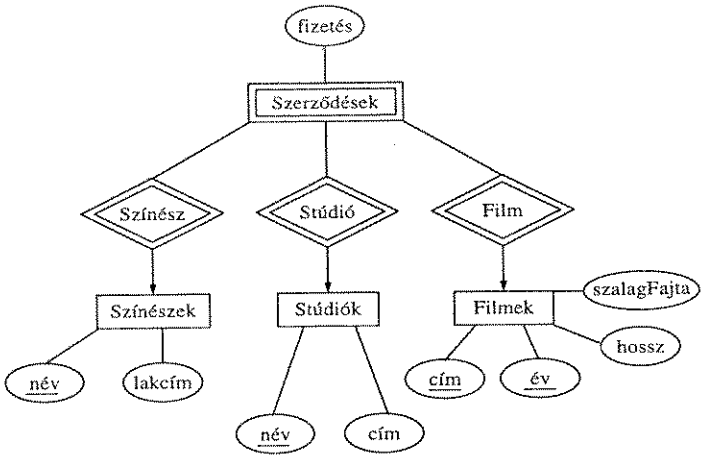
3.16. példa: Tekintsük most a 2.6.1. részben, a 2.31. példában és a 2.28. ábrán szereplő *Szerződések* nevű gyenge egyedhalmazt. A 3.19. ábrán megismételjük a szóban forgó diagramot. A `Szerződések` relációsémája következő:

```
Szerződések(színészNév, stúdióNév, cím, év, fizetés)
```

Az első attribútum a *Színészek* kulcsa, megfelelően átnevezve, a második attribútum a *Stúdiók* kulcsa, szintén megfelelőképpen átnevezve, ezután következik a *Filmek* egyedhalmaz kétattribútumos kulcsa, és végül szerepel a *Szerződések* egyedhalmaz egyetlen saját attribútuma, a *fizetés*. A *Színész*, *Stúdió* és *Film* kapcsolatokhoz nem készítettünk relációkat, mivel mindegyiknek a sémája a fenti `Szerződések` sémájának valódi részhalmaza lenne.

Mellékesen vegyük észre, hogy ugyanazt a relációt kaptuk, mint amit akkor kaptunk volna, ha a 2.13. ábra E/K diagramjából indultunk volna ki. Azon az ábrán a *Szerződések*et a színészek, stúdiók, filmek közötti olyan háromágú kapcsolattal adtuk meg, amelynek van egy saját *fizetés* attribútuma is. □

A 3.15. és a 3.16. példákban megfigyelt jelenség, vagyis hogy a dupla keretes rombuszkapcsolathoz nem szükséges relációt megadni, általában is igaz a gyenge egyedhalmazokra. Az *E* gyenge egyedhalmazhoz tartozó reláció sémája mindig tartalmazza azoknak a relációknak a sémáját, amelyeket a dupla rombuszos *R* kapcsolatokhoz készítenénk, ugyanis ezek olyan sok-egy kapcsolatok az *E*-ről egy vagy más egyedhalmazokra, amelyek az *E* kulcsának a definíciójához szükségesek. Mivel az *E*-hez tarto-



3.19. ábra. *A* `Szerződések` *nevű gyenge egyedhalmaz*

zó reláció tartalmazza az *E* kulcsattribútumait, ezek között szerepelni fog az *R* kapcsolattal összekötött két egyedhalmaz kulcsainak összes attribútuma. Ezek alapján a gyenge egyedhalmazokra vonatkozóan a következő módosított szabályokat állíthatjuk fel.

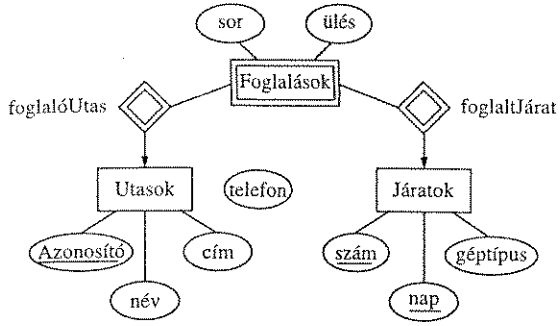
- Ha *E* egy gyenge egyedhalmaz, akkor készítsünk az *E* számára egy olyan relációt, amelynek sémája tartalmazza az *E* összes kulcsattribútumát, beleértve azokat az attribútumokat is, amelyek a sok-egy kapcsolaton keresztül „segítségül" használt egyedhalmazok kulcsai.

- *Ne* készítsünk relációt ahhoz a kapcsolathoz, amely sok-egy kapcsolat egy gyenge egyedhalmazból egy másik egyedhalmazba, és ráadásul ez a kapcsolat egy olyan dupla keretes rombuszos kapcsolat, amely a gyenge egyedhalmaz kulcsának definiálásához volt szükséges.

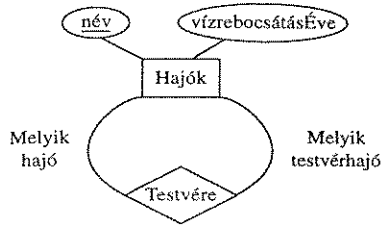
3.3.4. Feladatok

* **3.3.1. feladat:** Alakítsuk át a 3.20. ábra E/K diagramját relációs adatbázissémává.

* **3.3.2. feladat:** A 3.21. ábra E/K diagramja hajókat reprezentál. Testvéreknek hívjuk azokat a hajókat, amelyeket ugyanannak a tervnek az alapján építettek. Alakítsuk át ezt a diagramot is relációs adatbázissémává.



3.20. ábra. *A repülőgépes helyfoglalás E/K diagramja*



3.21. ábra. *A testvérhajók egy lehetséges E/K diagramja*

3.3.3. feladat: Alakítsuk át a következő E/K diagramokat relációs adatbázissémákra.

a) 2.28. ábra

b) 2.6.1. feladat válasza

c) 2.6.4.a) feladat válasza

d) 2.6.4.b) feladat válasza

3.4. Osztályhierarchia reprezentálása relációs modellben

Az objektumorientált és az E/K modellek az alosztályok fogalmát egy kissé eltérően kezelik. Ez a különbség két különböző lehetőséget nyújt számunkra ahhoz, hogyan adhatunk meg relációkat az osztályok hierarchiájának reprezentálására. Idézzük fel, hogy mi ez a különbség:

- Az ODL-ben bármely objektum pontosan egy osztályhoz tartozik. Az objektum örökli az összes szuperosztályának jellemzőit, de ugyanakkor nem eleme egyik szuperosztálynak sem.

- Az E/K modellben előfordulhat, hogy egy objektumot olyan egyedek segítségével reprezentálunk, amelyek egymással specializáló kapcsolatban álló egyedhalmazok elemei.

Nézzük meg, hogy a fentiek alapján hogyan tudunk különböző stratégiákat megadni az adatbázissémák tervezéséhez. Azonban ne feledjük, hogy sem az ODL sem az E/K modell nem határozza meg, hogy melyik megközelítést kell használnunk, azaz ha akarjuk bármelyik megközelítést alkalmazhatjuk a másik modellre is.

3.4.1. ODL alosztályok relációs reprezentálása

Először nézzünk egy technikát arra, hogyan alakíthatjuk át ODL alosztályok hierarchiáját relációsémákká. A következő alapelveket követjük:

- Minden alosztálynak megfeleltetünk egy relációt.

- Ebben a relációban az alosztály összes jellemzőjét reprezentáljuk, beleértve az összes örökölt jellemzőit is.

3.17. példa: Vizsgáljuk meg a 2.22. ábrán szereplő négy osztály hierarchiáját. Ezek az osztályok a következők voltak:

1. `Film`, amely a legáltalánosabb osztály. Ez az osztály ebben a fejezetben még számos példában elő fog fordulni.

2. `Rajzfilm`, amely a `Filmek` egy alosztálya, és amelynek van egy további jellemzője a `hangok` kapcsolat. Ez a kapcsolat a színészeknek egy halmazát adja meg.

3. `BűnügyiFilm`, amely a `Film` osztálynak szintén egy alosztálya és ennek is van egy további attribútuma: a `fegyver`.

4. `BűnügyiRajzfilm`, amely egyaránt alosztálya a `Rajzfilm` és a `BűnügyiFilm` alosztályoknak és ennek már nincsen további alosztálya. Ez az osztály természetesen örökli mindhárom szuperosztályának összes jellemzőjét.

A `Film` sémája változatlanul a következő:

```
Film(cím, év, hossz, szalagFajta, stúdióNév, színészNév)
```


A 3.13. ábrán megadtuk ennek a relációnak néhány jellemző sorát. A `Rajzfilm` reláció sémája hét attribútumot fog tartalmazni, nevezetesen a `Film` séma attribútumait és a `hang` attribútumot.

```
Rajzfilm(cím, év, hossz, szalagFajta, stúdióNév,
  színészNév, hang)
```

A `BűnügyiFilmek` számára egy másik relációt veszünk fel, melynek sémája tartalmazza a `Filmek` attribútumát és a `fegyver` attribútumot, vagyis a `BűnügyiFilmek` relációsémája a következő:

```
BűnügyiFilm(cím, év, hossz, szalagFajta, stúdióNév,
  színészNév, fegyver)
```

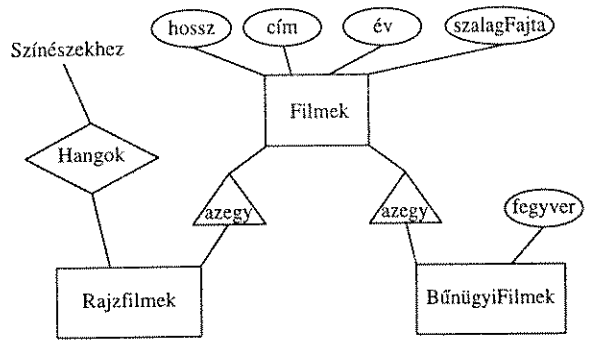
Végül a `BűnügyiRajzfilmek` relációsémájába nemcsak a `Film` hat attribútumát vesszük bele, hanem a másik két szuperosztályának a `hang` és `fegyver` attribútumait is. Így ennek a relációnak a sémája nyolc attribútumból áll:

```
BűnügyiRajzfilm(cím, év, hossz, szalagFajta,
  stúdióNév, színészNév, hang, fegyver)
```

□

3.4.2. Specializáló E/K kapcsolatok reprezentálása a relációs modellben

Az E/K modellben a specializáló („isa", illetve magyarul „azegy") hierarchiák mögött az a filozófia áll, hogy a hierarchiát olyan egyedek népesítik be, amelyek specializáló kapcsolatokon keresztül kapcsolódnak egymáshoz. Emiatt teljesen természetes az, hogy minden egyes egyedhalmazhoz egy olyan relációt készítünk, amelynek a sémája egyedül ennek az egyedhalmaznak az attribútumait tartalmazza. Azonban ahhoz, hogy



3.22. ábra. *A filmek hierarchiája*

az egyes sorokhoz tartozó egyedeket azonosítani tudjuk, mindenképpen szükségünk van minden egyes egyedhalmaz kulcsattribútumaira. Ennek eredményeképpen bármely alosztály elemére vonatkozó információt több reláció között fogunk szétosztani, de valószínűleg amúgy is ez az eset következne be, mivel az E/K diagramot relációsémává átalakító eljárás az E/K diagram attribútumaira és kapcsolataira vonatkozó információt külön relációkban tárolja.

A specializáló kapcsolatokhoz nem készítünk relációkat. Ehelyett a specializáló kapcsolatokat burkolt formában az a tény fejezi ki, hogy a kapcsolódó egyedeknek ugyanazok a kulcsértékeik.

3.18. példa: Reprezentáljuk a 2.22. ábra hierarchiáját az E/K modellben. Ennek az E/K diagramnak a lényeges részét már bemutattuk a 2.23. ábrán, és ezt most megismételjük a 3.22. ábrán. Ennek a diagramrészletnek a reprezentálásához az alábbi relációsémákra van szükség:

1. A `Filmek(cím, év, hossz, szalagFajta)` reláció. Ez a reláció már szerepelt a 3.10. példában.

2. A `BűnügyiFilmek(cím, év, fegyver)` reláció. Az első két attribútuma a *Filmek* egyedhalmaz kulcsa, az utolsó attribútum pedig *BűnügyiFilm* egyedhalmaz egyetlen saját attribútuma.

3. A `Rajzfilmek(cím, év)` reláció. Ez a reláció a rajzfilmeknek a halmaza. Ennek a relációnak nincs más attribútuma a filmek kulcsán kívül, mivel a rajzfilmekre vonatkozó többletinformációt a *Hangok* kapcsolat tartalmazza.

4. A `Hangok(cím, év, név)` reláció, amely a *Színészek* és a *Rajzfilmek* közötti *Hangok* kapcsolatnak felel meg. Az utolsó attribútum a *Színészek* kulcsa az első kettő pedig a *Rajzfilmek* kulcsa.

Vegyük észre, hogy a 3.22. ábrán nem szerepel olyan egyedhalmaz, amely a `BűnügyiRajzfilmek` osztálynak felelne meg. Emiatt ellentétben a 3.17. példában megadott relációs tervvel, nem adtunk meg külön relációt az olyan filmekre, amelyek egyaránt bűnügyi filmek és rajzfilmek is. Ugyanis az ilyen filmek esetében a hozzájuk tartozó hangokat megkapjuk a `Hangok` relációból a fegyvert a `BűnügyiFilmek` relációból, és az összes többi információt vagy a `Filmek` relációból vagy egy olyan kapcsolatnak megfelelő relációból, amelyben vagy a `Filmek` vagy a `Rajzfilmek` vagy a `BűnügyiFilmek` egyedhalmazok szerepelnek.

Továbbá vegyük azt is észre, hogy a `Rajzfilmek`nek megfelelő reláció sémája a `Hangok` relációsémájának egy részhalmaza. Emiatt sok esetben elhagyhatnánk a `Rajzfilmek` relációt, mivel nem ad ahhoz képest többlet információt, mint amennyi a `Hangok` relációban található. Azonban gondoljunk arra is, hogy lehetnek a rajzfilmek között némafilmek is az adatbázisunkban. Ezekhez a rajzfilmekhez nem tartoznának hangok, és emiatt elveszítenénk azt a tényt, hogy ezek a filmek rajzfilmek. Tulajdonképpen ugyanez a probléma előfordul a 3.17. példában szereplő `Rajzfilmek` reláció esetén is, ugyanis ha nincsen hang megadva, akkor semmilyen információt nem tárolhatunk az adott rajzfilmről. Ezt a problémát majd a normalizációval tudjuk megoldani, amint látni fogjuk a 3.7. fejezetben. □

3.4.3. A két kiindulási eset összehasonlítása

A 3.4.1. és 3.4.2. részekben tárgyalt megközelítések mindegyikének megvan a saját problémája. Az ODL átírása során egy objektum összes jellemzője együttesen kerül be egy relációba. Emiatt azonban kénytelenek vagyunk több relációt is végignézni, ha meg akarunk találni egy objektumot. Például a 3.17. példa adatbázissémáját használva ahhoz, hogy megtaláljuk egy filmnek a hosszát, négy különböző relációt kell végignéznünk, amíg meg nem találjuk azt az osztálynak megfelelő relációt, amelyben a kívánt film szerepel.

Másrészt az E/K diagram átírása során egy objektum kulcsát megismételjük minden egyes olyan egyedhalmaz vagy kapcsolat esetén, amelyhez az objektum (egyed) hozzátartozik. Ez az ismétlés sok helyet foglalhat el feleslegesen. Ezenkívül itt is előfordulhat, hogy több relációt kell végignéznünk ahhoz, hogy megtaláljuk a kívánt információt egy objektumról. Ez fordulna elő például ha a 3.18. példa adatbázissémájában egy bűnügyi film hosszát és fegyverét szeretnénk megkeresni.

3.4.4. Relációk összevonása nullértékek használatával

Megadható még egy megközelítés azzal kapcsolatban, hogyan reprezentáljuk az osztályok hierarchiáját. Használhatunk ugyanis speciális *nullértéket,* amit `NULL`-lal fogunk jelölni. Amikor a `NULL` szerepel egy sor valamely attribútumhoz tartozó komponensében, az lényegében azt jelenti, hogy a sornak ehhez az attribútumához nincsen megfelelő érték megadva. Ugyan a nullértékek nem képezik részét a hagyományos relációs modellnek, de azért nagyon hasznos és kiemelkedő szerepet játszanak az SQL lekérdezőnyelvben, ahogy ezt majd látni fogjuk az 5.9. részben.

Abban az esetben, ha a sorokban megengedett a nullértékek használata, akkor az osztályok hierarchiáját egyetlen relációval is megadhatjuk. Ennek a relációnak attribútuma lesz a hierarchiában szereplő bármely osztálynak az összes attribútuma. Így egy objektumot egyetlen sorral tudunk reprezentálni, és ebben a sorban `NULL` szerepel minden egyes olyan attribútum esetében, amely nem tartozik az adott objektum osztályához.

3.19. példa: Ha ezt a megközelítést használnánk a 3.17. példa problémájánál, akkor csak egyetlen relációt kellene készítenünk, melynek sémája a következő lenne:

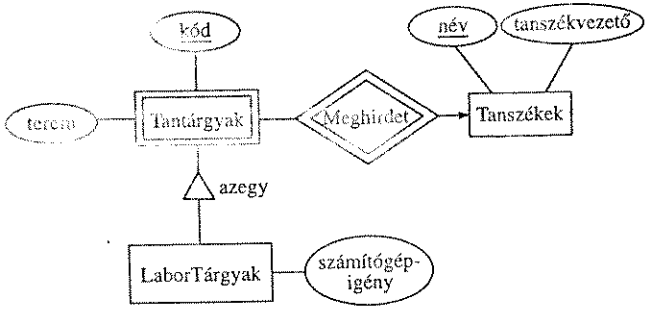
```
Film(cím, év, hossz, szalagFajta, stúdióNév,
  színészNév, hang, fegyver)
```

Azokat a filmeket, amelyek egyszerre rajzfilmek és bűnügyi filmek is (például a *Roger nyúl a pácban*), több nullértéket nem tartalmazó sorral reprezentálhatnánk, és ebben az esetben minden egyes hanghoz külön sor tartozna.[5] Másrészt az olyan filmek esetében, amelyek rajzfilmek, de nem bűnügyi filmek (például a *Kis hableány*), a `fegyver` komponensbe nullérték kerülne. A *Gyilkosság az Orient expresszen* c. filmnek a `hang` attribútum értéke nullérték lenne, míg az *Elfújta a szél*nek mind a `hang` mind a `fegyver` attribútumának értéke lenne nullérték. □

Vegyük észre, hogy hasonlóan a 3.4.2. rész megközelítéséhez ez a megközelítés is lehetővé teszi számunkra, hogy elég egy relációt végignézni ahhoz, hogy a hierarchiában szereplő osztályok sorait megtaláljuk. Másrészt a 3.4.1. rész megközelítéséhez hasonlóan itt is teljesül, hogy bármely objektumhoz tartozó összes információ megtalálható egy relációban.*

3.4.5. Feladatok

3.4.1. feladat: Alakítsuk át a 3.23. ábra E/K diagramját relációs adatbázissémává.



3.23. ábra. *A 3.4.1. feladathoz tartozó E/K diagram*

3.4.2. feladat: A 3.24. ábrán egy olyan sémának az ODL leírását találjuk, amely a 3.4.1. feladat E/K diagramjához hasonlít. Alakítsuk át ezt is relációs adatbázissémává.

Ne feledjük, hogy a `Tantárgy` objektumoknak valamilyen objektumazonosítóval kell rendelkezniük. E célból bevezethetünk egy olyan attribútumot, például a `TantárgyAzon` attribútumot, amely az objektumazonosítót reprezentálja. Ebben a feladatban nem kell feltétlen ugyanazt a stratégiát utánozni, amelyet a 3.4.1 feladatban használtunk a gyenge egyedhalmaz átalakításánál, de persze használhatjuk azt is, ha úgy látjuk jónak.

[5] A *Roger nyúl a pácban* c. filmhez hasonló filmek esetében, vagyis amikor a filmben színészek is szerepelnek és a rajzoknak szinkronhangja is van, tulajdonképpen minden egyes színész-hang párhoz hozzátartozna egy sor. Egy tiszta rajzfilm esetében viszont a `színészNév` attribútumhoz `NULL` tartozna, és csak a hangot, illetve a többi információt kellene tárolni.

* Szerkesztői megjegyzés: ezzel a megközelítéssel könnyen veszíthetünk információt, nevezetesen, nem biztos, hogy megállapítható egy filmről, hogy rajzfilm.

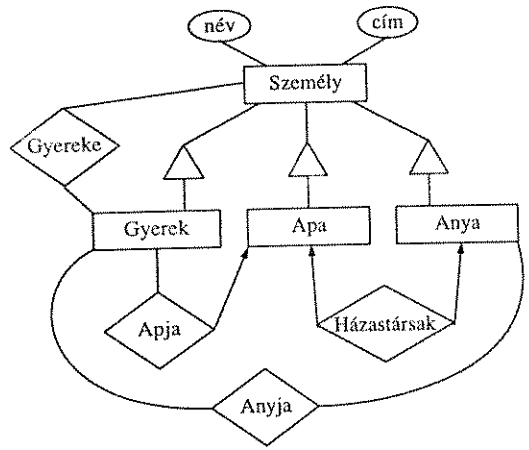

```
interface Tantárgy {
      attribute int kód;
      attribute string terem;
   relationship Tanszék meghirdetőTanszék
      inverse Tanszék::meghirdet;
};
interface Labor : Tantárgy {
   attribute int számítógépigény;
};
interface Tanszék {
   unique attribute string név;
   attribute string tanszékvezető;
   relationship Set<Tantárgy> meghirdet
      inverse Tantárgy::meghirdetőTanszék;
};
```

3.24. ábra. *A tantárgyak és laboratóriumi foglalkozások egy lehetséges ODL leírása*

3.4.3. feladat: Alakítsuk át a következő feladatok ODL terveit relációs adatbázissémákká.

* a) 2.4.1. feladat

b) 2.4.4. feladat



3.25. ábra. *A 3.4.5. feladathoz tartozó E/K diagram*

3.4.4. feladat: Alakítsuk át a következő feladatok E/K terveit relációs adatbázissémákká.

* a) 2.4.3. feladat

b) 2.4.4. feladat

! **3.4.5. feladat:** Alakítsuk át a 3.25. ábrán látható E/K diagramot relációs adatbázissémává.

! **3.4.6. feladat:** Mennyire kapnánk a 3.4.5. feladat relációs adatbázissémájától eltérő megoldást, ha a megfelelő ODL definícióból indultunk volna ki?

3.5. Funkcionális függőségek

A relációs modellben a legfontosabb megszorítás amivel foglalkozunk az egyetlenérték megszorítás, az úgynevezett „funkcionális függőség". Ennek a megszorítástípusnak az ismerete az adatbázissémák tervezésénél válik majd jelentőssé a redundancia megszüntetésében, amit később a 3.7. részben fogunk tárgyalni. Más megszorítások is segítik a helyes adatbázisséma tervezését: a többértékű függőség, amelyet a 3.8. részben tárgyalunk, a létezési megszorítások és a függetlenségi megszorítások, amelyeket a 4.5. részben értelmezünk.

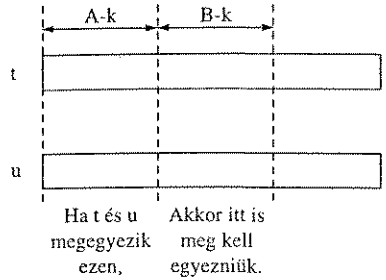
3.5.1. A funkcionális függőség definíciója

A *funkcionális függőség* 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 egy másik attribútumon, a B-n is."* Ezt a függőséget formálisan $A_1A_2\ldots A_n \rightarrow B$-vel jelöljük, és azt mondjuk, hogy „$A_1, A_2, \ldots, A_n$ funkcionálisan meghatározza B-t."

Ha $A_1, A_2, \ldots, A_n$ attribútumhalmaz több attribútumot is funkcionálisan meghatároz, azaz például ha

$$A_1A_2\ldots A_n \rightarrow B_1$$
$$A_1A_2\ldots A_n \rightarrow B_2$$
$$\ldots$$
$$A_1A_2\ldots A_n \rightarrow B_m,$$

* Szerkesztői megjegyzés: fontos az olvasó figyelmét felhívni arra, hogy egy relációelőfordulás akkor elégíti ki ezt a funkcionális függőséget, ha teljesül rá az állítás. Ez úgy is lehetséges, hogy nincs két olyan sora, amelyek megegyeznek $A_1, A_2, \ldots, A_n$ mindegyikén.



3.26. ábra. *A funkcionális függőség két soron vett hatása*

akkor a függőségeknek ezt a halmazát röviden az alábbi formában írhatjuk

$$A_1A_2\ldots A_n \rightarrow B_1B_2\ldots B_m$$

A 3.26. ábrán láthatjuk, hogy mit jelent a funkcionális függőség az R reláció bármely két t és u sorára.

3.20. példa: Tekintsük a 3.13. ábrán a `Film` relációt, és ennek a 3.27. ábrán található előfordulását. A `Film` relációval kapcsolatosan több funkcionális függőséget tudunk ésszerűen állítani. Például megkövetelhetjük az alábbi három függőséget

```
cím év → hossz
cím év → szalagFajta
cím év → stúdióNév
```

Mivel mindhárom függőségnek ugyanaz a bal oldala, `cím` és `év`, ezért röviden ezeket egy sorban összegezhetjük

```
cím év → hossz szalagFajta stúdióNév
```

Informálisan a funkcionális függőségek halmaza azt jelenti, hogy ha két sornak ugyanaz az értéke a `cím` komponensen és ugyanaz az értéke az `év` komponensen, akkor ennek a két sornak ugyanazt az értéket kell felvennie a `hossz` komponensen, ugyanazt az értéket a `szalagFajta` komponensen, és ugyanazt az értéket a `stúdióNév` komponensen. Ez a követelmény akkor érdekes, ha visszaemlékezünk arra, hogy eredetileg hogyan alakítottuk ki a `Film` relációsémát. A `cím` és `év` attribútumok a film objektum egy kulcsát alkották. Tehát azt várjuk, hogy ha megadjuk a film címét és azt, hogy melyik évben készült, akkor egyértelműen tudjuk a film hosszát, a film típusát és, hogy melyik stúdió készítette.

Másrészt vegyük észre, hogy a

```
cím év → színészNév
```

A funkcionális függőség sémaszintű fogalom

Ne felejtsük el, hogy a funkcionális függőség, mint minden más függőség, a relációsémára jelent feltételt, nem pedig egy bizonyos előfordulásra.* Ha csupán egy pillanatnyi előfordulást nézünk, nem mondhatjuk meg, hogy egy bizonyos funkcionális függőség biztosan érvényes-e vagy sem. Például vegyük a 3.27. ábrán található előfordulást, esetleg feltehetnénk azt, hogy fennáll a következő függőség: `cím` → `szalagFajta`, mivel a `Film` reláció szóban forgó előfordulásának minden sorára igaz, hogy ha bármely két sor megegyezik a `címen`, akkor megegyezik a `szalagFajtán` is.

Azonban nem állíthatjuk ezt a funkcionális függőséget általában a `Film` relációra, ugyanis lehetne például olyan előfordulásunk, amelyben két sor van a `King Kong` két olyan filmváltozatáról, hogy az egyik színes a másik pedig fekete-fehér, ekkor a fent javasolt funkcionális függőség már nem lenne érvényes.

állítás hamis. Ez nem érvényes funkcionális függőség. Pedig feltehetnénk, hogy a függőség fennáll, hiszen tudjuk, hogy a `cím` és `év` a filmben kulcs. De amikor definiáltuk a `Film` osztályt, az teljesült, hogy mindegyik filmhez a színészek *halmaza* volt csak egyértelműen meghatározva. Amikor ODL-ről térünk át relációs modellre, akkor minden filmhez több sort kell elkészítenünk, és minden sorba más színészt kell bejegyeznünk. Tehát, még ha az összes ilyen sor meg is egyezik a `Film` osztály többi tulajdonságain, ezek nem egyeznek meg a színészek nevein. □

cím	*év*	*hossz*	*szalagFajta*	*stúdióNév*	*színészNév*
Csillagok háborúja	1977	124	színes	Fox	Carrie Fisher
Csillagok háborúja	1977	124	színes	Fox	Mark Hamill
Csillagok háborúja	1977	124	színes	Fox	Harrison Ford
Rút kiskacsa	1991	104	színes	Disney	Emilio Estevez
Wayne világa	1992	95	színes	Paramount	Dana Carvez
Wayne világa	1992	95	színes	Paramount	Mike Meyers

3.27. ábra. *A `Film` reláció*

* Szerkesztői megjegyzés: még pontosabban ez azt írja elő, hogy csak olyan előfordulások lehetségesek, amelyek kielégítik a függőséget.

3.5.2. Relációk kulcsai

Azt mondjuk, hogy az egy vagy több attribútumból álló $\{A_1, A_2, ..., A_n\}$ halmaz a reláció *kulcsa*, ha:

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

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

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

3.21. példa: A `{cím, év, színészNév}` attribútumok a 3.27. ábrán szereplő `Film` 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: `cím, év` és `színészNév`. Mivel ezek megegyeznek a `címen` és éven, ezért meg kell egyezniük a `hossz, szalagFajta` és `stúdióNév` attribútumokon, mint korábban, a 3.20. példában már beláttuk. Tehát nincs két olyan különböző sor, amely mindhárom `cím, év` és `színészNév` attribútumon megegyezne, hiszen ez tulajdonképpen ugyanazt a sort jelentené.

Most azt kell megmutatnunk, hogy `{cím, év, színészNév}` attribútumhalmaznak nincs olyan valódi részhalmaza, amely funkcionálisan meghatározná az összes többi attribútumot. Ahhoz, hogy ezt belássuk, először vegyük észre, hogy a `cím` és `év` nem határozza meg a `színészNevet`, mivel több filmben is szerepel egynél több színész. Tehát a `{cím, év}` nem kulcs.

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

`év színészNév` → `cím`

nem teljesülő funkcionális függőség. Végül azt állítjuk, hogy `{cím, színészNév}` sem kulcs, hiszen lehet két olyan film, amelyeknek ugyanaz a címük, mégis más években készültek. Lehetséges, hogy a két filmben ugyanazok a színészek szerepelnek, mégsem ez a jellemző.[6] □

[6] Ne feledjük, hogy a funkcionális függőségek az adatokra vonatkozó feltevések, állítások. Nem várhatjuk, hogy valaki abszolút biztonsággal eldönti helyettünk, hogy mely függőségek teljesülnek, melyek nem, aztán megosztja velünk ezt az információt. Éppen ellenkezőleg, nekünk kell szabadon dönteni arról, hogy melyek a legtermészetesebbnek tűnő elvárások az adatok összefüggéseivel kapcsolatban.

Mi „funkcionális" a funkcionális függőségben?

$A_1A_2...A_n \rightarrow B$-t „funkcionális" függőségnek nevezzük, mivel elméletben van egy olyan függvény (funkció), ami olyan értéklistán van értelmezve, amely az A_1, A_2, ..., A_n minden egyes attribútumához hozzárendel egy-egy értéket, és előállítja a B egyértelmű értékét (vagy egyáltalán semmilyen értéket). Például a `Film` 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 `Film` 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 argumentum értékekből. Azaz nincs olyan művelet, amit 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 `címhez` és `évhez` tartozó sort és megnézzük, hogy milyen érték található a `hosszra` abban a sorban. (A függvényt táblázattal adjuk meg, és nem kiszámítási eljárással.)

Előfordulhat, hogy a relációnak egynél több kulcsa van. Ha így van, akkor általában kijelöljük az egyik kulcsot, mint *elsődleges kulcsot*. A kereskedelmi adatbázisrendszereknél az elsődleges kulcs megválasztása az implementációt is befolyásolhatja, például azt, hogyan tároljuk a relációt a lemezen.

3.5.3. Szuperkulcsok

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

3.22. példa: A 3.21. példa relációjában több szuperkulcs van. Nemcsak a

```
{cím, év, színészNév}
```

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

```
{cím, év, színészNév, hossz}
```

is szuperkulcs. □

3.5.4. Relációk kulcsainak megtalálása

Ha egy relációsémát ODL vagy E/K tervek relációkká történő átírásával alakítunk ki, akkor gyakran előre feltételezzük a reláció kulcsát. Ebben a részben az E/K diagramból származó relációkat vizsgáljuk. A 3.5.5. részben fogjuk venni az ODL tervekből származó relációkat.

Ha egy ODL vagy E/K tervből írtuk át a relációt, nagyon gyakori (de nem biztos), hogy minden relációnak csak egy kulcsa van. Ha a relációnak egy kulcsa van, az a megállapodás, hogy húzzuk alá a relációsémában a kulcsattribútumokat.

Az első szabály a származtatott kulcsokra vonatkozik:

- Ha egy egyedhalmazból írtuk át a relációt, akkor a reláció kulcsát ennek az egyedhalmaznak vagy osztálynak a kulcsattribútumai alkotják.

3.23. példa: A 3.10. és 3.11. példákban megadtuk, hogyan írjuk át a `Filmek` és `Színészek` egyedhalmazokat relációkká. Ezeknek az egyedhalmazoknak a kulcsai `{cím, év}`, illetve `{név}` voltak. Így ezek lesznek a kulcsok is a megfeleltetett relációkban, azaz az alábbi relációsémákat kapjuk

```
Filmek(cím, év, hossz, szalagFajta)
Színészek(név, lakcím)
```

amelyekben aláhúzással jelöltük meg a kulcsokat. □

A második szabály a bináris kapcsolatokra vonatkozik. Ha egy R reláció egy kapcsolatból származik, akkor a kapcsolat foka befolyásolja az R kulcsát. Három esetet különböztetünk meg:

- Ha a kapcsolat sok-sok, akkor a két összekapcsolt egyedhalmaz kulcsainak összes attribútuma adja meg az R kulcsattribútumainak halmazát.

- Ha a kapcsolat sok-egy kapcsolat az E_1 egyedhalmazról az E_2 egyedhalmazra, akkor az E_1 kulcsattribútumai az R kulcsattribútumai lesznek, de az E_2 egyedhalmaz kulcsattribútumai nem lesznek azok.

- Ha a kapcsolat egy-egy, akkor bármelyik összekapcsolt egyedhalmaz kulcsattribútumai lehetnek az R kulcsattribútumai. Ekkor R-nek nem egyedi a kulcsa, hanem több kulcsa van.

3.24. példa: A 3.12. példában vizsgáltuk a *Gyártó* kapcsolatot, amely sok-egy kapcsolat a *Filmek* egyedhalmazról a *Stúdiók* egyedhalmazra. Így a `Gyártó` reláció kulcsa tartalmazza a *Filmek* kulcsát, azaz a `cím` és `év` attribútumokat. A `Gyártó` sémája, aláhúzva a kulcsattribútumokat a következő

```
Gyártó(cím, év, stúdióNév)
```

Ezzel szemben a 3.13. példában vizsgált, *Filmek* és *Színészek* közötti sok-sok *Szereplők* nevű kapcsolatból eredményül kapott reláció összes attribútuma kulcsattribútum.

```
Szereplők(cím, év, színészNév)
```

Valójában az egyetlen lehetőség, hogy a sok-sok kapcsolatból átírt reláció ne csupán kulcsattribútumokat tartalmazzon az, hogy magának a kapcsolatnak legyenek attribútumai. Ekkor ezeket az attribútumokat nem kell belevennünk a kulcsba. □

Végül nézzük meg a többágú kapcsolatokat. Minthogy nem tudjuk leírni az összes lehetséges függőséget a kapcsolatból kivezető nyilakkal, előfordulnak olyan helyzetek, amikor nem nyilvánvaló, hogy mi lesz a kulcs vagy kulcsok, anélkül hogy át ne gondolnánk pontosan, hogy mely egyedhalmazok mely egyedhalmazokat határoznak meg funkcionálisan. Amit biztosan állíthatunk, hogy

- Ha egy R többágú kapcsolatból vezet egy nyíl egy E egyedhalmazhoz, akkor a megfeleltetett relációban van legalább egy olyan kulcs, amelyben nem szerepel az E kulcsa.

3.5.5. ODL-ből származtatott relációk kulcsai

A feladat valamivel bonyolultabb, ha ODL tervet írunk át relációkká. Először is, az ODL osztálynak egy vagy több deklarált kulcsa lehet, de előfordulhat az is, hogy egyáltalán nincs kulcs az attribútumok között. Ebben az esetben bele kell vennünk a relációba egy olyan attribútumot, amely helyettesíti az osztály objektumainak objektumazonosítóját, amint ezt korábban a 3.2.6. részben tárgyaltuk.

Akár az ODL osztálynak a saját attribútumaiból van kulcsa, akár nekünk kell helyettesítenünk az objektumazonosítót kulcsként, mégis vannak olyan körülmények, amikor az osztály kulcsattribútumai a relációnak *nem* képezzük kulcsát. Ennek az az oka, hogy az ODL relációkká átírásánál időnként túl sok minden kerül egyetlen relációba. A probléma akkor merül fel, ha az osztály definíciója tartalmaz kapcsolatokat.

Először legyen egy C osztálynak az R egy egyértékű kapcsolata valamely D osztályra. Ekkor ahogyan a 3.2.4. részben ajánlottuk, vegyük bele D kulcsát C relációjába. A C kulcsa a megfeleltetett relációnak is a kulcsa.

A problémás eset, ha a C osztálynak az R egy többértékű kapcsolata valamely D osztályra. Ha az R inverze az ellentétes irányban egyértékű (azaz R egy-sok kapcsolat), akkor ahogyan azt a 3.2.7. szakaszban a „Kapcsolatok egyirányú reprezentálása" bekeretezett részben ajánlottuk, csak a D relációban reprezentáljuk az R-et (azaz R inverzét). R inverzének a reprezentálása nem okoz problémát a D-ben, ugyanis D-ben ez egyértékű.

Tegyük fel, hogy R sok-sok, azaz R is és az inverze is többértékű mind a C-ben és mind a D-ben. Ekkor a C-hez létrehozott relációban a C osztály egy objektumának a reprezentálása több sorral lehetséges. Ennek eredménye, hogy C kulcsa a megfeleltetett relációnak nem lesz kulcsa, hanem hozzá kell vennünk D kulcsát a C kulcsához, hogy megkapjuk a reláció kulcsát.

3.25. példa: A 3.7. példában átírtuk a `Film` ODL osztályt relációvá oly módon, hogy a `Film` reláció attribútumaihoz hozzávettük:

1. A `Stúdió` osztály `stúdióNév` kulcsát (amelyhez a `Film` a `gyártó` egyértékű kapcsolattal kapcsolódik) és
2. A `Színész` osztály `színészNév` kulcsát (amelyhez a `Film` reláció a `szereplők` többértékű kapcsolattal kapcsolódik).

Ezek közül az első egyértékű kapcsolat lévén nem változtatja meg a `Film` reláció kulcsát. Azonban a második többértékű kapcsolat, ezért ki kell egészítenünk ezzel a `Film` reláció kulcsát, ami így a következő lesz:

```
{cím, év, színészNév}
```

A 3.13. ábrán található példán a `Film` relációra láthatjuk, hogy a `cím` és év önmagukban nem adnak kulcsot, de a javasolt `színészNév` hozzávételével már kulcsot alkot. □

Általában, ha a C-nek megfeleltetett reláció több C-ről való többértékű kapcsolatot is reprezentál, akkor a C-hez kapcsolódó összes osztálynak a kulcsát hozzá kell vennünk a C kulcsához, és így az eredmény a C kapcsolatát is reprezentáló reláció kulcsa lesz. Természetesen, ha több kapcsolat kapcsolja a C-t ugyanahhoz a D osztályhoz, akkor a C-hez tartozó relációban különböző attribútumokkal reprezentáljuk minden egyes C-ről D-re kapcsolathoz a D kulcsát.

Emiatt egy ODL osztály kulcsa nem elegendő ahhoz, hogy a megfeleltetett relációnak a kulcsát alkossa. Ha a 3.2. részben megadott relációkká átírást követjük, akkor gyakran ki kell javítanunk az ezzel az egyszerű eljárással kapott relációkat. Az ilyen relációk javítását a 3.7. részben fogjuk venni. Látni fogjuk, hogy a sok-sok kapcsolatokat le lehet választani a kapcsolt osztályok bármelyik relációjáról. Az eredményül kapott relációsémák összessége nagyon hasonló az E/K tervekből közvetlenül kapott sémákhoz[7].

[7] Természetesen először átalakíthatjuk az ODL terveket vele ekvivalens E/K tervekké, és azt írjuk át relációs tervekké. Ebben a lehetőségben kicsit csökkennek azok a problémák, amik a 3.2. rész közvetlen megközelítéséből adódnak, de nem lényegesen. A 3.7. rész relációs tervezési technikái, amelyeket egyébként is meg kell ismernünk, ezt a feladatot is megoldják.

3.5.6. Feladatok

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

* **3.5.2. 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?

! **3.5.3. feladat:** A 2.3.2. feladatban megvizsgáltunk a *Születések* kapcsolatra négy különböző feltételt. Ezek mindegyikére válasszuk ki annak a relációnak a kulcsát vagy kulcsait, amelyet ebből a kapcsolatból alakítottunk át.

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

3.6. Funkcionális függőségekre vonatkozó szabályok

Ebben a részben azt tanuljuk meg, milyen levezetési szabályok érvényesek a funkcionális függőségekre. Tegyük fel, hogy tudjuk, hogy a reláció milyen függőségi halmazt elégít ki. Anélkül, hogy pontosan ismernénk a reláció sorait, levezethetjük, hogy a relációnak milyen bizonyos további függőségeket kell kielégítenie. Ez a képesség, hogy fel tudjuk tárni a további függőségeket a relációséma-tervek tárgyalásánál válik majd jelentőssé a 3.7. részben.

3.26. példa: Ha tudjuk, hogy az A, B és C attribútumokkal rendelkező R reláció eleget tesz az $A \to B$ és $B \to C$ funkcionális függőségeknek, akkor ebből levezethetjük, hogy

R eleget tesz az $A \to C$ függőségnek is. Lássuk, hogyan érvelhetünk! Ahhoz, hogy bebizonyítsuk $A \to C$ érvényességét, vennünk kell R bármely két olyan sorát, ha egyáltalán lehet, amelyek megegyeznek az 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.* □

A funkcionális függőségeket gyakran többféleképpen is megadhatjuk anélkül, hogy változna a reláció érvényes előfordulásainak a halmaza. Ilyenkor azt mondjuk, hogy a két függőségi halmaz ekvivalens. Általánosabban azt mondjuk, hogy az S funkcionális függőségi halmaz következik a T funkcionális függőségi halmazból, ha minden olyan relációelőfordulás, amely eleget tesz az összes T-beli függőségnek, eleget tesz minden S-beli függőségnek is. Az S és T függőségi halmazok ekvivalensek, ha S következik T-ből és T pedig következik S-ből.

Ebben a részben 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 3.26. példában láttuk. Megadunk egy algoritmust is majd, aminek segítségével válaszolhatunk arra az általános kérdésre, hogy egy funkcionális függőség következik-e egy vagy több függőségből.

3.6.1. Szétvághatósági és összevonhatósági szabály

Idézzük fel, hogy a 3.5.1. részben definiált alábbi funkcionális függőség

$$A_1A_2\ldots A_n \to B_1B_2\ldots B_m$$

csupán a rövidítése a következő függőségi halmaznak.

$$A_1A_2\ldots A_n \to B_1$$
$$A_1A_2\ldots A_n \to B_2$$
$$\ldots$$
$$A_1A_2\ldots A_n \to B_m$$

---

* Szerkesztői megjegyzés: a bizonyításból igazolódott az is, hogy A kulcs, tehát nem is lehet a két függőséget kielégítő relációban két olyan sor, amelyik A-ban megegyezik. A bizonyítás valójában indirekt volt.

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 $A_1A_2...A_n \rightarrow B_1B_2...B_m$ függőséget helyettesíthetjük $A_1A_2...A_n \rightarrow B_i$, $i = 1, 2, ..., m$ funkcionális függőségekből álló halmazzal. Ezt az átalakítást *szétvághatósági szabálynak* nevezzük.

- Az $A_1A_2...A_n \rightarrow B_i$, $i = 1, 2, ..., m$ funkcionális függőségekből álló halmazt helyettesíthetjük $A_1A_2...A_n \rightarrow B_1B_2...B_m$ egyetlen függőséggel. Ezt az átalakítást *összevonhatósági szabálynak* nevezzük.

Például említettük már korábban a 3.20. példában, hogy az alábbi függőségi halmaz

```
cím év → hossz
cím év → szalagFajta
cím év → stúdióNév
```

ekvivalens egyetlen függőséggel.

```
cím év → hossz szalagFajta stúdióNév
```

Azt lehetne képzelni, hogy a szétvághatóság a funkcionális függőségek bal oldalaira ugyanúgy alkalmazható, mint a jobb oldalakra, azonban ez téves, nincs a bal oldalakra vonatkozó szétvághatósági szabály. Ezt a következő példa mutatja.

3.27. példa: Legyen a 3.20. példa `Film` relációjára érvényes

```
cím év → hossz
```

függőség. Ha megpróbáljuk a bal oldalt szétvágni

```
cím → hossz
év → hossz
```

akkor hibás függőségeket kapnánk. Ugyanis a `cím` nem határozza meg funkcionálisan a `hosszt`, mivel lehet két azonos című film (pl. *King Kong*), amelyeknek a hossza különbözik. Hasonlóan az `év` sem határozza meg funkcionálisan a `hosszt`, hiszen biztosan több különböző hosszú film készült ugyanabban az évben. □

3.6.2. Triviális függőségek

Azt mondjuk, hogy az $A_1A_2...A_n \rightarrow B$ funkcionális függőség *triviális*, ha B az A-k közül az egyik. Például

```
cím év → cím
```

triviális függőség.

Minden triviális függőség érvényes minden relációban, mivel amikor azt mondjuk, hogy „két sor megegyezik minden A_1, A_2, ..., A_n attribútumon, akkor megegyezik ezek bármelyikén is". Tehát feltehetjük bármelyik triviális függőséget anélkül, hogy indokolnunk kellene az adatokról alkotott elképzelések alapján.

A funkcionális függőségekre vonatkozó eredeti definíciónkban nem engedtük meg a triviális függőséget. Mégsem okoz zavart, ha bevesszük őket, mivel ezek mindig érvényesek, és néha leegyszerűsítik a szabályokra vonatkozó állításokat.

Ha megengedjük a triviális függőségeket, akkor megengedünk olyan függőségeket is (mint rövidítéseket), amelyekben a jobb oldali attribútumok közül valamelyik a bal oldalon is előfordul. Azt mondjuk, hogy az $A_1A_2...A_n \rightarrow B_1B_2...B_m$ funkcionális függőség

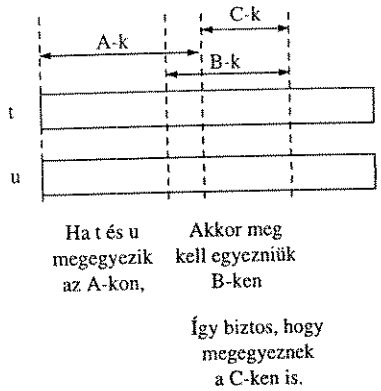
- *Triviális*, ha a B-k az A-k egy részhalmazát alkotják.

- *Nem triviális*, ha a B-k közül legalább egy nincs benne az A-kban.

- *Teljesen nem triviális*, ha a B-k egyike sem egyezik meg az A-k valamelyikével.

Tehát

```
cím év → év hossz
```



3.28. ábra. *A triviális függőségi szabály*

nem triviális függőség, de nem teljesen nem triviális. Ha törölnénk az évet a jobb oldalról, akkor kapnánk teljesen nem triviális függőséget.

Mindig eltávolíthatjuk a funkcionális függőség jobb oldaláról azokat az attribútumokat, amelyek előfordulnak a bal oldalon. Azaz:

- Az $A_1A_2...A_n \rightarrow B_1B_2...B_m$ funkcionális függőség ekvivalens $A_1A_2...A_n \rightarrow$ $\rightarrow C_1C_2...C_k$-val, ahol a C-k az összes B-k közül éppen azok, amelyek nem szerepelnek A-k között.

Ezt a szabályt *triviális függőségi szabálynak* nevezzük, és a 3.28. ábrán mutatjuk be.

3.6.3. Attribútumhalmazok lezárásának kiszámítása

Mielőtt a többi szabállyal folytatnánk, megadunk egy általános elvet, amely az összes szabályra következik. Legyen $\{A_1, A_2, ..., A_n\}$ egy attribútumhalmaz, S pedig funkcionális függőségeknek egy halmaza. Az $\{A_1, A_2, ..., A_n\}$ 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 $A_1A_2...A_n \rightarrow B$-nek is. Azaz $A_1A_2...A_n \rightarrow B$ az S-beli függőségekből következik. Jelöljük az $\{A_1, A_2, ..., A_n\}$ attribútumhalmaz lezártját $\{A_1, A_2, ..., A_n\}^+$-szal. Ahhoz, hogy egyszerűsítsük a lezártak kiszámolásának a tárgyalását, megengedjük a triviális függőségeket, így $A_1, A_2, ..., A_n$ mindig benne van $\{A_1, A_2, ..., A_n\}^+$-ban.

A 3.29. 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, és ez az eredményhalmaz lesz a lezárás. Nézzük meg részletesebben milyen lépésekből áll az $\{A_1, A_2, ..., A_n\}$ attribútumhalmazra egy S funkcionális függőségi halmaz szerint vett lezárásának kiszámolási algoritmusa. (A szétvághatósági szabály miatt feltehetjük, hogy S-ben minden függőség jobb oldala egyelemű.)

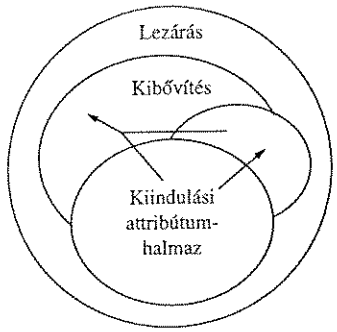
1. Legyen X attribútumhalmaz, amely végül maga a lezárt lesz. Legyen először X kezdőértéke $\{A_1, A_2, ..., A_n\}$.

2. Ismételten keresünk olyan

   $B_1B_2...B_m \rightarrow C$

   funkcionális függőséget S-ből, amelyre a teljes $B_1, B_2, ..., B_m$ benne van az X attribútumhalmazban, de a C nincs. Ekkor C-t hozzávesszük az X halmazhoz.

3. A 2. lépést mindaddig ismételjük, ameddig már nem tudunk több attribútumot hozzávenni az X-hez. Mivel X csak növekedhet, és bármely relációnak véges sok attribútuma lehet, tehát S-ben is csak véges sok függőség van, végül már nem tudjuk az X-et tovább bővíteni.



3.29. ábra. *Egy attribútumhalmaz lezárásának kiszámítása*

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

3.28. példa: Legyen egy reláció, amelynek attribútumai A, B, C, D, E és F. Legyenek ehhez a relációhoz tartozó funkcionális függőségek $AB \rightarrow C$, $BC \rightarrow AD$, $D \rightarrow E$ és $CF \rightarrow B$. Mi $\{A, B\}$ lezártja, azaz $\{A, B\}^+$?

Induljunk ki $X = \{A, B\}$-ből. Először vegyük észre, hogy az $AB \rightarrow C$ funkcionális függőség bal oldalán szereplő attribútumai mind benne vannak az X-ben, így hozzávehetjük a függőség jobb oldali attribútumát, azaz C-t az X-hez. Tehát a 2. lépés első ciklusában X egyenlő lesz $\{A, B, C\}$-vel.

Ezután látjuk, hogy a $BC \rightarrow AD$ bal oldala benne van az X-ben, így hozzávehetjük A és D attribútumokat az X-hez.[8] Az A már benne van, a D nincs, amivel bővítjük az X-et, így ez egyenlő lesz $\{A, B, C, D\}$-vel. Most a $D \rightarrow E$ függőséget használva, E-vel bővítjük az X-et, ami egyenlő lesz $\{A, B, C, D, E\}$-vel. Ezután az X-en már nem tudunk tovább változtatni. Ebben a példában a $CF \rightarrow B$ funkcionális függőséget nem vettük figyelembe, mivel X soha nem tartalmazta a függőség bal oldalát. Tehát $\{A, B\}^+ = \{A, B, C, D, E\}$. □

Ha tudjuk hogyan számoljuk ki tetszőleges attribútumhalmaz lezárását, akkor ellenőrizhetjük, hogy egy adott $A_1A_2...A_n \rightarrow B$ funkcionális függőség következik-e az S függőségi halmazból. Ha B benne van $\{A_1, A_2, ..., A_n\}^+$-ban, akkor

$$A_1A_2...A_n \rightarrow B$$

[8] Ne feledjük, hogy a $BC \rightarrow AD$ jelölés a $BC \rightarrow A$ és $BC \rightarrow D$ függőségekből álló párt jelenti, ezért, ha szükséges, akkor a pár bármelyik elemét külön is használhatjuk az érvelésünk során.

következik az S-ből, ha pedig B nincs benne $\{A_1, A_2, ..., A_n\}^+$-ban, akkor ez a függőség nem következik az S-ből. Általánosabban, amikor a függőség jobb oldalán több attribútum áll, akkor ezt is hasonlóan ellenőrizhetjük, hiszen ez a függőség rövidítése egy fenti formájú függőségekből álló halmaznak. Tehát az $A_1A_2...A_n \rightarrow B_1B_2...B_m$ funkcionális függőség akkor és csak akkor következik az S függőségi halmazból, ha $B_1, B_2, ..., B_m$ benne van $\{A_1, A_2, ..., A_n\}^+$-ban.

3.29. példa: Vegyük a 3.28. példában szereplő relációt és funkcionális függőségeket. Tegyük fel, hogy ellenőrizni szeretnénk, vajon az $AB \rightarrow D$ következik-e ezekből a függőségekből. Számoljuk ki $\{A, B\}^+$-t, amely $\{A, B, C, D, E\}$, ahogyan az előző példában beláttuk. Mivel D eleme az utóbbi halmaznak, 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. □

Miért működik a lezárási algoritmus?

Egyszerű az oka, hogy a lezárást kiszámoló algoritmus jól működik. Indukcióval bizonyíthatjuk be – aszerint, hogy az X-ben szereplő egyes D attribútumokra a második lépés bővítési műveletét hányszor kellett alkalmazzuk –, hogy az $A_1A_2...A_n \rightarrow D$ funkcionális függőség fennáll (speciális esetben, amikor a D az A-k közül való, ez a függőség triviális). Eszerint minden olyan R reláció, amely eleget tesz az összes S-beli függőségnek, eleget tesz az $A_1A_2...A_n \rightarrow D$-nek is.

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

Az indukcióhoz tegyük fel, hogy D-vel akkor bővítünk, amikor felhasználjuk a $B_1B_2...B_m \rightarrow D$ függőséget. Az indukciós feltétel miatt R eleget tesz az $A_1A_2...A_n \rightarrow B_i$ függőségeknek, az összes $i = 1, 2, ..., m$ esetre. Másképpen szólva az R-nek bármelyik két olyan sora, amely az összes $A_1, A_2, ..., A_n$-n megegyezik, megegyezik az összes $B_1, B_2, ..., B_m$-n is. Mivel R eleget tesz a $B_1B_2...B_m \rightarrow D$-nek, azt is tudjuk, hogy ez a két sor megegyezik D-n. Tehát az R eleget tesz $A_1A_2...A_n \rightarrow D$-nek.

A fenti bizonyítás mutatja, hogy a lezárási algoritmus *helyes*, azaz amikor D bekerül $\{A_1, A_2, ..., A_n\}^+$-ba, akkor az $A_1A_2...A_n \rightarrow D$ érvényes függőség. Nem mutattuk meg a másik irányt, a *teljességet*, azaz amikor $A_1A_2...A_n \rightarrow D$ fennáll, akkor D bekerül $\{A_1, A_2, ..., A_n\}^+$-ba. Ennek a bizonyítása meghaladja e könyv kereteit.

3.6.4. Tranzitivitási szabály

A tranzitivitási szabállyal két funkcionális szabályból következtethetünk egy újabbra.

- Ha $A_1A_2...A_n \rightarrow B_1B_2...B_m$ és $B_1B_2...B_m \rightarrow C_1C_2...C_k$ teljesül az R relációban, akkor $A_1A_2...A_n \rightarrow C_1C_2...C_k$ szintén teljesül az 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 3.6.3. rész ellenőrzési algoritmusát. Ahhoz, hogy ellenőrizzük $A_1A_2...A_n \rightarrow C_1C_2...C_k$ teljesül-e, ki kell számolnunk $\{A_1, A_2, ..., A_n\}^+$-t.

Az $A_1A_2...A_n \rightarrow B_1B_2...B_m$ funkcionális függőség miatt az összes $B_1B_2...B_m$ benne van $\{A_1, A_2, ..., A_n\}^+$-ban. Ezután alkalmazhatjuk $B_1B_2...B_m \rightarrow C_1C_2...C_k$ függőséget, hogy a $C_1, C_2, ..., C_k$-t hozzávegyük $\{A_1, A_2, ..., A_n\}^+$-hoz. Mivel az összes C benne van $\{A_1, A_2, ..., A_n\}^+$-ban, kapjuk, hogy

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

fennáll minden olyan relációban, amely eleget tesz mind az $A_1A_2...A_n \rightarrow B_1B_2...B_m$ és mind a $B_1B_2...B_m \rightarrow C_1C_2...C_k$ függőségeknek.

3.30. példa: Kezdjük el a 3.12. ábrán szereplő `Film` relációval, amely a `Film` osztályt reprezentáló négy attribútumból plusz a `Stúdió` osztállyal való `gyártó` kapcsolatából hoztunk létre a 3.2.4. részben. Maga a reláció néhány mintaadattal együtt legyen az alábbi:

cím	*év*	*hossz*	*szalagFajta*	*stúdióNév*
Csillagok háborúja	1977	124	színes	Fox
Rút kiskacsa	1991	104	színes	Disney
Wayne világa	1992	95	színes	Paramount

Lezárások és kulcsok

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

Tegyük fel, hogy elhatároztuk, hogy a gyártó stúdiókról is reprezentálunk pár adatot ugyanebben a relációban. Az egyszerűség kedvéért csak a stúdió városát vegyük hozzá, amivel a címét reprezentáljuk. A reláció ekkor a következő:

cím	*év*	*hossz*	*szalagFajta*	*stúdióNév*	*stúdióCím*
Csillagok háborúja	1977	124	színes	Fox	Hollywood
Rút kiskacsa	1991	104	színes	Disney	Buena Vista
Wayne világa	1992	95	színes	Paramount	Hollywood

Az alábbi két függőségről feltehetjük, hogy fennállnak:

```
cím év → stúdióNév
stúdióNév → stúdióCím
```

Az első azért indokolt, mert a `Film` osztály gyártó kapcsolata egyértékű, azaz a filmet csak egy stúdió gyártotta. A második azért igaz, mert a `Stúdió` osztályban a `cím` attribútum egyértékű, nevezetesen `karakterlánc` (lásd a 2.6. ábrát).

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

```
cím év → stúdióCím
```

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

3.6.5. Funkcionális függőségi halmazok lezárása

Láttuk, hogy egy adott függőségi halmazból gyakran tudunk következtetni más függőségekre, beleértve mind a triviális, mind a nem triviális függőségeket. A további részekben megkülönböztetjük az *adott* függőségeket, amelyeket kezdetben kötöttünk ki a relációra és a *levezetett* függőségeket, amelyek valamelyik szabály alkalmazásával, vagy az attribútumhalmaz lezárási algoritmusának az alkalmazásával következnek.

Ezen felül időnként lehetőségünk van arra, hogy megválasszuk azokat a függőségeket, amelyek a reláció teljes függőségi halmazát reprezentálják. Függőségek bármely olyan halmazát, amelyből a reláció összes függőségére tudunk következtetni, az adott reláció *bázisának* nevezzük. Ha a bázisban nem található a függőségeknek olyan valódi részhalmaza, amelyből a teljes függőségi halmazt le tudnánk vezetni, akkor a bázist *minimálisnak* nevezzük.*

3.31. példa: Legyen $R(A, B, C)$ olyan reláció, hogy mindegyik attribútuma funkcionálisan meghatározza a másik két attribútumát. A levezetett függőségek teljes halmaza

* Szerkesztői megjegyzés: a minimális bázis precíz definíciója még azt is megköveteli, hogy egyetlen függőség bal oldala se legyen csökkenthető. Ehhez a jobb oldalakat először egyeleművé kell tenni a szétszedéssel.

A levezetési szabályok egy teljes halmaza

Ha meg akarjuk tudni, hogy egy funkcionális függőség következik-e néhány adott függőségből, a 3.6.3. részben 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 $\{B_1, B_2, ..., B_m\} \subseteq \{A_1, A_2, ..., A_n\}$, akkor

   $A_1A_2...A_n \rightarrow B_1 B_2, ... B_m$.

Ezeket neveztük triviális függőségeknek.

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

   $A_1A_2...A_n C_1C_2...C_k \rightarrow B_1B_2... B_m C_1C_2...C_k$

bármely $C_1C_2...C_k$ attribútumhalmazra.

3. *Tranzitivitás.* Ha

   $A_1A_2...A_n \rightarrow B_1B_2...B_m$ és $B_1B_2...B_m \rightarrow C_1C_2...C_k$

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

így hat olyan nem triviális függőséget tartalmaz, amelynek a bal oldala és a jobb oldala is egy attribútumból áll: $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow A$, $B \rightarrow C$, $C \rightarrow A$ és $C \rightarrow B$. Tartalmaz továbbá három olyan nem triviális függőséget, amelynek a bal oldalán kettő, a jobb oldalon pedig egy attribútum áll: $AB \rightarrow C$, $AC \rightarrow B$ és $BC \rightarrow A$. Tartalmazza még a függőségi pároknak a rövidítéseit, mint amilyen például $A \rightarrow BC$, és tartalmazhatja a triviális függőségeket, mint például $A \rightarrow A$ vagy $AB \rightarrow BC$, amely nem teljesen nem triviális (bár a funkcionális függőséget a szigorúbb definícióink szerint véve nem kell felsorolnunk a triviális vagy részben triviális függőségeket, vagy az olyan függőségeket, amelyeknek a jobb oldala több attribútumból áll).

Ennek a relációnak ezekhez a függőségekhez több minimális bázisa van. Az egyik

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

* Szerkesztői megjegyzés: levezetésen az axiómák alkalmazásának olyan véges sorozatát értjük, ahol egy axióma feltétel részébe behelyettesíthetünk tetszőleges kiindulási, vagy már előző lépésben levezetett függőséget, ahol ennek értelme van (2. és 3. axióma). Egy függőséget akkor vezetünk le, ha van olyan levezetés, amelynek utolsó lépésében a felhasznált axióma következmény része éppen az adott függőség. Az Armstrong-axiómák nem csak teljesek, de helyesek is, azaz csak olyan függőség vezethető le, amelyik valóban következmény.

Egy másik

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

Ezeken kívül még több más bázis és minimális bázis létezik a relációhoz, amelyeknek a feltárását meghagyjuk feladatnak. □

3.6.6. Feladatok

* **3.6.1. 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?

b) Melyek az R összes kulcsai?

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

3.6.2. feladat: Ismételjük meg a 3.6.1. 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.

3.6.3. feladat: Mutassuk meg, hogy az alábbi szabályok érvényesek, használjuk a 3.6.3. rész lezárási algoritmusát.

* a) *Bővítés a bal oldalakon.* Ha $A_1A_2...A_n \to B$ funkcionális függőség és C egy másik attribútum, akkor $A_1A_2...A_nC \to B$ következik.

b) *Teljes bővítés.* Ha $A_1A_2...A_n \to B$ funkcionális függőség és C egy másik attribútum, akkor $A_1A_2...A_nC \to BC$ következik. Megjegyezzük, hogy ebből a szabályból az Armstrong-axiómák „bővítési" szabálya, amelyet a 3.6.5. rész bekeretezett részében állítottunk, könnyen belátható.

c) *Pszeudotranzitivitás.* Tegyük fel, hogy $A_1A_2...A_n \to B_1B_2...B_m$ és $C_1C_2...C_k \to D$ funkcionális függőségek fennállnak, és B-k mindegyike a C-k között előfordul. Ekkor $A_1A_2...A_nE_1E_2...E_j \to D$ fennáll, ahol az E-k éppen azok a C-k közül valók, amelyek nincsenek B-k között.

d) *Additivitás.* Ha $A_1A_2...A_n \to B_1B_2...B_m$ és $C_1C_2...C_k \to D_1D_2...D_j$ funkcionális függőségek fennállnak, akkor

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

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

! **3.6.4. feladat:** Mutassuk meg, hogy az alábbiak nem érvényes szabályok a funkcionális függőségekre úgy, hogy adjunk példát olyan relációra, amely eleget tesz az adott függőségeknek, de nem tesz eleget az állítólagos 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 $B \to C$.

! **3.6.5. 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.

! **3.6.6. 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.

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

!! **3.6.8. 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 attribútumból álló összes halmaz zárt.

b) Az egyetlen zárt halmaz $\emptyset$* és $\{A, B, C, D\}$.

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

! **3.6.9. feladat:** Keressük meg a 3.31. példában szereplő relációhoz és függőségekhez az összes minimális bázist.

---

* Szerkesztői megjegyzés: az olvasót meglepheti az üres halmaz megjelenése zárt halmazként. A lezárás definíciójában ez értelmezhetetlen, az algoritmus megadja az értelmét. Van értelme az $\emptyset \to A$ funkcionális függőségnek is, azt jelenti, hogy az A oszlop konstans értékű.

!! **3.6.10. feladat:** Mutassuk meg, hogy ha F funkcionális függőség következik adott funkcionális függőségekből, akkor az F-et be tudjuk bizonyítani az adott függőségekből az Armstrong-axiómák (lásd 3.6.5. rész bekeretezett definíciójának) segítségével. *Útmutatás:* Vizsgáljuk meg az attribútumhalmaz lezárását számoló algoritmust, és lássuk be, hogy az algoritmus minden lépését helyettesíthetjük az Armstrong-axiómákból következő funkcionális függőségekkel.

3.7. Relációs adatbázissémák tervezése

Többször megjegyeztük, hogy az objektumorientált ODL tervek közvetlen átírása relációs adatbázissémákká (az E/K tervek esetén kisebb mértékben) problémákhoz vezet. Az alapvető probléma, amit eddig megismertünk, a redundancia, amikor a tényeket több sorban is megismételjük. Azt is tudjuk már, hogy ennek a redundanciának a legáltalánosabb oka az, hogy egyetlen relációba próbáljuk csoportosítani az objektumnak mind az egyértékű, mind a többértékű tulajdonságait. Például a 3.2.2. részben ott láttunk redundanciát, amikor megpróbáltuk tárolni a filmek egyértékű tulajdonságait, mint amilyen a hossz, együtt a többértékű tulajdonságokkal, mint amilyen egy filmben szereplő filmszínészek halmaza. Ezt a problémát a 3.27. ábrán láthattuk, amelyet itt megismétlünk a 3.30 ábrán. Ehhez hasonló redundanciával találkoztunk akkor, amikor megpróbáltuk együtt tárolni egy színész egyértékű születésnapjáról az információt a színész lakcímeinek a halmazával.

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

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

2. Ezután bevezetjük a „felbontás" („dekompozíció") ötletét, 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 feltételt a relációsémák felbontásánál.

3.7.1. Problémák

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

1. *Redundancia.* Az információk feleslegesen ismétlődhetnek több sorban. A 3.30. ábrán a filmek hossza és a szalagFajta erre példa.

cím	*év*	*hossz*	*szalagFajta*	*stúdióNév*	*színészNév*
Csillagok háborúja	1977	124	színes	Fox	Carrie Fisher
Csillagok háborúja	1977	124	színes	Fox	Mark Hamill
Csillagok háborúja	1977	124	színes	Fox	Harrison Ford
Rút kiskacsa	1991	104	színes	Disney	Emilio Estevez
Wayne világa	1992	95	színes	Paramount	Dana Carvez
Wayne világa	1992	95	színes	Paramount	Mike Meyers

3.30. ábra. *A* `Film` *reláció az anomáliák bemutatására*

2. *Módosítási problémák.* 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, akkor, ha nem vagyunk elég gondosak, a 3.30. 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 `Film` relációt át lehet úgy tervezni, hogy az ilyen hibák veszélye egyáltalán ne merülhessen fel.

3. *Törlési problémák.* Ha az értékek halmaza üres halmazzá válik, akkor ennek mellékhatásaként más információt is elveszíthetünk. Például ha törölnünk kell a *Rút kiskacsá*ban szereplő színészek közül Emilio Estevezt, akkor az adatbázisban ehhez a filmhez nem maradna több színész, így a `Film` relációban a *Rút kiskacsá*ra vonatkozó sor eltűnne azzal az információval együtt, hogy a film 104 perces és színes.

3.7.2. Relációk felbontása

Az anomáliák megszüntetésének az elfogadott útja a relációk *felbontása (dekompozíciója).* R felbontása egyrészt azt jelenti, hogy R attribútumait szétosztjuk úgy, hogy ezáltal két új reláció sémáját alakítjuk ki belőlük. A felbontási szabályunk másrészt azt is jelenti, hogyan töltsük fel a relációt olyan sorokkal, amelyeket az R sorainak „vetítésével" kapunk. Miután megadjuk a felbontási eljárást, megmutatjuk hogyan válasszunk ki egy olyan felbontást, amellyel megszüntetjük az anomáliákat.

Legyen adott R reláció $\{A_1, A_2, ..., A_n\}$ sémával, R-et felbonthatjuk S és T két relációra, amelyeknek sémái $\{B_1, B_2, ..., B_m\}$ és $\{C_1, C_2, ..., C_k\}$ úgy, hogy

1. $\{A_1, A_2, ..., A_n\} = \{B_1, B_2, ..., B_m\} \cup \{C_1, C_2, ..., C_k\}$.

2. Az S reláció sorai az R-ben szereplő összes sornak a $\{B_1, B_2, ..., B_m\}$-re vett vetületei, azaz R aktuális előfordulásának minden egyes t sorára vesszük a t azon komponenseit, amelyek $B_1, B_2, ..., B_m$ attribútumokhoz tartoznak. Ezek az értékek egy sort alkotnak és ez a sor fog belekerülni az S aktuális előfordulásába. Jóllehet a re-

cím	*év*	*hossz*	*szalagFajta*	*stúdióNév*
Csillagok háborúja	1977	124	színes	Fox
Rút kiskacsa	1991	104	színes	Disney
Wayne világa	1992	95	színes	Paramount

3.31. ábra. *A* `Film1` *reláció*

lációk halmazok, az R két különböző sorának a projekciója ugyanazt a sort is eredményezheti az S-ben. Ha így lenne, akkor az ilyen sorokból csak egyet kell belevennünk az S aktuális előfordulásába.

3. Hasonlóan a T reláció sorai az R aktuális előfordulásában szereplő sorok $\{C_1, C_2, \ldots, C_k\}$ attribútumok halmazára vett projekciói.

3.32. példa: Bontsuk fel a 3.30. ábrán látható `Film` relációt. Először felbontjuk a sémát. A választott felbontást, amelynek az előnyeit később a 3.7.3. részben vizsgáljuk meg, legyen

1. A `Film1` elnevezésű relációsémája legyen a `színészNév` kivételével az összes attribútum.

2. A `Film2` elnevezésű relációsémája álljon a `cím, év` és `színészNév` attribútumokból.

A relációelőfordulások felbontásának a folyamatát a 3.30. ábra mintaadatainak segítségével mutatjuk be. Először állítsuk elő a projekciót a `Film1` sémára:

```
{cím, év, hossz, szalagFajta, stúdióNév}
```

A 3.30. ábra első három sorának ezen az öt attribútumon megegyeznek a komponensei:

```
(Csillagok háborúja, 1977, 124, színes, Fox)
```

A negyedik sor különböző sort állít elő az első öt komponensre, az ötödik és hatodik sor pedig ugyanazt az öt komponenst adja. A `Film1` reláció eredményét a 3.31. ábra mutatja.

cím	*év*	*színészNév*
Csillagok háborúja	1977	Carrie Fisher
Csillagok háborúja	1977	Mark Hamill
Csillagok háborúja	1977	Harrison Ford
Rút kiskacsa	1991	Emilio Estevez
Wayne világa	1992	Dana Carvez
Wayne világa	1992	Mike Meyers

3.32. ábra. *A* `Film2` *reláció*

Tekintsük a 3.30. ábra projekcióját a `Film2` sémára. Az ábrán található mind a hat sor különbözik legalább az egyik attribútumon a `cím, év` és `színészNév` attribútumok közül, így eredményül a 3.32. ábrán látható relációt kapjuk. □

Megjegyezzük, hogy ez a felbontás megszünteti a 3.7.1. részben említett anomáliákat. A redundanciát kizártuk, például minden film hossza csak egyszer fordul elő a `Film1` relációban. A módosítási anomália kockázata megszűnt. Például, mivel csak `Film1` egy sorában kell megváltoztatnunk a *Csillagok háborúja* film hosszát, ezért nem fordulhat elő, hogy ennek a filmnek két különböző hosszát tároljuk véletlenül.

Végül a törlési anomália kockázata is megszűnt. Ha töröljük a *Rút kiskacsa* filmben szereplő összes színészt, akkor ez a film kiesik a `Film2` relációból, de az összes többi információ erről a filmről még megtalálható lesz a `Film1`-ben.

Még mindig előfordulhat redundancia a `Film2`-ben, mivel egy film címe és éve többször is előfordulhat. Jóllehet ez a két attribútum a filmeknek egy kulcsát alkotja, mégsem tudjuk másképpen szabatosan reprezentálni a filmet. Továbbá a `Film2`-ben mégsem merül fel a módosítási anomália lehetősége. Ha megváltoztatjuk a *Csillagok háborúja* évét mondjuk 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 1997-ben, 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.

3.7.3. 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 normálformának* vagy *BCNF*-nek nevezzük.

- Az R reláció BCNF-ben van akkor és csak akkor, ha minden olyan esetben, ha az R-ben érvényes egy $A_1A_2\ldots A_n \rightarrow B$ nem triviális függőség, akkor az $\{A_1, A_2, \ldots, A_n\}$ halmaz R szuperkulcsa kell hogy legyen.

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.

Amikor találunk egy, a BCNF feltételt megsértő függőséget, akkor időnként érdemes megkeresnünk az összes többi függőséget is, amelyeknek ugyanez a bal oldala, független attól, hogy ezek is megsértik-e a BCNF feltételt vagy nem. Az alábbi definíció a BCNF egyik alternatív definíciója, amelyben olyan közös bal oldallal rendelkező

függőségeknek a halmazát keressük, hogy legalább az egyik ezek közül nem triviális és megsérti a BCNF feltételt.

- R reláció BCNF-ben van akkor és csak akkor, ha bármikor fennáll az R-ben az $A_1A_2...A_n \rightarrow B_1B_2...B_m$ nem triviális függőség, akkor az $\{A_1, A_2, ..., A_n\}$ halmaz R szuperkulcsa kell hogy legyen.

Ez a követelmény az eredeti BCNF feltétellel ekvivalens. Idézzük fel, hogy $A_1A_2...A_n \rightarrow B_1B_2...B_m$ függőség az $A_1A_2...A_n \rightarrow B_i$, $i = 1, 2, ..., m$ függőségek halmazának a rövidítése. Mivel legalább az egyik B_i-nek különbözőnek kell lennie az A-któl (máskülönben $A_1A_2...A_n \rightarrow B_1B_2...B_m$ triviális lenne), ez a BCNF megszegését jelenti az eredeti definíciónk értelmében.

3.33. példa: A 3.30. ábrán látható `Film` reláció nincs BCNF-ben. Ahhoz, hogy belássuk miért nem, először meg kell határoznunk mely attribútumhalmazok alkotják a kulcsokat. A 3.21. példában láttuk, hogy a {`cím, év, színészNév`} miért alkot kulcsot. Tehát bármely olyan attribútumhalmaz, amely tartalmazza ezt a hármat, az szuperkulcs. A 3.21. példa megfontolásaihoz hasonlóan beláthatjuk azt is, hogy nem lehet szuperkulcs az olyan attribútumhalmaz, amely nem tartalmazza mindhármat a `cím, év` és `színészNév` közül. Tehát azt kaptuk, hogy a {`cím, év, színészNév`} az egyetlen kulcs a `Film` relációhoz.

Tekintsük a következő funkcionális függőséget

`cím év → hossz szalagFajta stúdióNév`

amelyről tudjuk, hogy fennáll a `Film` relációban. Idézzük fel, hogy miért tételezhetjük fel ezt a függőséget: az eredeti ODL tervben a kulcs a {`cím, év`} volt, a `hossz` és `szalagFajta` egyértékű attribútumok voltak, és az egyértékű `gyártó` kapcsolat mutatta a gyártó stúdiót.

Sajnos a fenti függőség bal oldala nem szuperkulcs, hiszen tudjuk, hogy a `cím` 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 `Film` nincs BCNF-ben. Továbbá a BCNF eredeti definíciójához, amikor azt követeltük meg, hogy a jobb oldalon egy egyszerű attribútum legyen, választhatjuk a három funkcionális függőség bármelyikét, mint például `cím év → hossz`-t, mivel mindegyik megsérti a BCNF-et. □

3.34. példa: Másrészt a 3.31. ábrán látható `Film1` reláció BCNF-ben van. Mivel

`cím év → hossz szalagFajta stúdióNév`

fennáll ebben a relációban, és korábban beláttuk, hogy sem a `hossz` sem a `cím` önmagában nem határozza meg funkcionálisan a többi attribútum egyikét sem, ezért a `Film1` reláció egyetlen kulcsa a {`cím, év`}. Továbbá a nem triviális funkcionális

függőségek legalább a `cím` és `év` attribútumokat tartalmazzák a bal oldalon, emiatt a bal oldalak szuperkulcsok. Tehát a `Film1` BCNF-ben van. □

3.35. 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, emiatt 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 dolgozóazonosítót adhat meg és a személyi számukat is bejegyzik. Abban a relációban, amelyiknek a `dolgozóAzonosító` és a `személyiSzám` az attribútumai bármelyik attribútum funkcionálisan meghatározza a másikat. Másképpen kifejezve, mindegyik attribútum kulcs, mivel várhatóan nem találhatunk két olyan sort, ahol a két attribútum bármelyiken megegyezne. □

3.7.4. Boyce-Codd normálformájú felbontás

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

---

* Szerkesztői megjegyzés: a relációséma felbontását adott függőségi rendszer mellett végezzük, a függőségi rendszert a séma részének tekintjük a következőkben. Végül adatbázissémát kapunk.