Adatbázis fejlesztés és üzemeltetés I.

Szabó Bálint

MÉDIAINFORMATIKAI KIADVÁNYOK

Adatbázis fejlesztés és üzemeltetés I.

Szabó Bálint

Eger, 2013

Korszerű információtechnológiai szakok magyarországi adaptációja

TÁMOP-4.1.2-A/1-11/1-2011-0021

Lektorálta:
Nyugat-magyarországi Egyetem Regionális Pedagógiai Szolgáltató és Kutató Központ

Felelős kiadó: dr. Kis-Tóth Lajos
Készült: az Eszterházy Károly Főiskola nyomdájában, Egerben
Vezető: Kérészy László
Műszaki szerkesztő: Nagy Sándorné

Tartalom

1. BEVEZETÉS

1.1 CÉLKITŰZÉSEK, KOMPETENCIÁK A TANTÁRGY TELJESÍTÉSÉNEK FELTÉTELEI

1.1.1 Célkitűzés

A szerszámkészítés megjelenésének az emberi felemelkedésében betöltött szerepét sokan és sokféleképpen értelmezték már. Gondoljunk erről bármit is, azt nem vitathatjuk, hogy fajunk evolúciós teljesítményében rendkívüli jelentősége van annak, hogy a Homo faber megjelenése óta eszközöket készítünk legkülönbözőbb problémáink, feladataink megoldásához.

A 18–19, század a termékek tömeges előállításában és a technikai innovációban hozott – az akkori szinten – hihetetlen fejlődést, a 20. században pedig az elektronikus berendezések, elsősorban a számítógép megjelenése okozott forradalmi változásokat.

Míg az addig előállított termékek zöme természetében és a vele megoldható problémák tekintetében is szorosan kötődött az anyagi világhoz, a 20. század második felétől egyre nagyobb tömegben használunk információs rendszereket problémáink megoldására.

Információs rendszereket, amelyek legfontosabb elemei, a szoftvertermékek anyagi reprezentációja a működés tekintetében már nem releváns.

Ezzel párhuzamosan és szervesen összefonódva soha nem látott mértékben növekedett az emberiség által felhalmozott tudásmennyiség. Az információ és a tudás kezelésének jelentőségéről árulkodik, hogy köznyelvi fogalommá váltak a tudásalapú és az információs társadalom fogalmak. Az információ napjainkra a fölemelkedés, az érvényesülés és a hatalom hajtóerejévé, ugyanakkor szimbólumává is vált.

Ez azonban nem feltétlenül az egyén által fejben tartott tudás mennyiségének mindenhatóságát jelenti. Sokkal nagyobb a jelentősége a releváns adatok gyors felkutatásának, a biztonságos informálódásnak, az információ hatékony feldolgozásának, és a helyes következtetésnek.

Talán nem véletlen, hogy az egyik legismertebb magyar feltaláló, Rubik Ernő, egymáshoz viszonyítva csökkenőnek véli a birtokolt ismeretek fontosságát, és egyre komolyabb jelentőséget tulajdonít az információ fölkutatás és a kreatív alkalmazás képességének.

Törvényszerű, hogy az információszerzés és feldolgozás jelentőségének megnövekedésével párhozamosan hatalmas fejlődésen mentek keresztül az információs rendszerek is. A múlt század ’70-es éveinek elejére tehető szoftverkrízis fölrázta az informatikai szakembereket, és világossá tette, hogy az információs rendszereket alkotó szoftvertermékek előállításában legalább olyan fontos szerep jut a tudományos alapokon nyugvó tervezésnek, mint az anyagi természetű termékek gyártásában.

Az információszerzést lehetővé tévő adatok biztonságos tárolásában, a gyors hozzáférés biztosításában, a hatékony informálódásban döntő szerep jut a különböző adatbázisoknak és az azokat kezelő szoftvereszközöknek.

A legelterjedtebb, relációs adatmodell alapstruktúrájának megtévesztő egyszerűsége a megrendelő, és a tapasztalatlan fejlesztő számára egyaránt az adatbázis-kezelés banalitását sugallhatja. Ugyanakkor a többi információs rendszerekhez hasonlóan ezek hatékony működésének is záloga az alapos és átgondolt tervezőmunka.

Úgy véljük, minden informatikai szakember számára elengedhetetlen, hogy megismerje és megértse e szakterület ismeretanyagát. Tananyagunkban arra vállalkozunk, hogy átadjuk az olvasónak az adatbázis-rendszerek működésének megértéséhez, az adatbázisok tervezéséhez,fizikai kialakításához és üzemeltetéséhez szükséges elméleti, valamint gyakorlati ismereteket.

1.1.2 Ismeretek, kompetenciák

A tananyag leckéinek elolvasásával az alábbi ismeretekre és kompetenciákra tehet szert:

- Megismerheti az adatbázis-rendszerek fölépítését, az adatbázis-rendszerek és adatbázis-kezelő rendszerek közötti különbségeket.
- Megértheti az adatbázisok tervezésének fontosságát, tisztában lesz az adatbázisok koncepcionális, logikai és fizikai modellezésének eszközeivel, a különböző szintű modellezések jelentőségével.
- Alaposan megismerheti a koncepcionális, Entity-Relathionship modellezés lehetőségeit, a modell ábrázolására alkalmas ER-diagram elkészítésének módját és a legfontosabb jelölési technikákat.
- Megtanulhatja a relációs adatmodell szabályait, és megismerheti a normalizálás, a redundanciától mentes relációs adatbázisok kialakításának folyamatát.

- Elsajátíthatja az ER-modell relációs adatmodellre leképezésének szabályait, melyek alkalmazásával összetett ER-modellekből kiindulva is redundanciamentes adatbázisokhoz juthat.
- Megismerheti az Oracle ingyenesen használható, ugyanakkor professzionális lehetőségeket nyújtó adatbázis-kezelő rendszerét, a MySQL-t.
- Megtanulhatja, hogyan lehet elkészíteni az SQL nyelv segítségével a koncepcionális, és a logikai szinten korábban megtervezett adatbázisok fizikai tervét.
- Megismerheti a MySQL-adatbázisok létrehozásának és biztonságos üzemeltetésének módszereit.
- Megtanulhatja, hogyan szabályozhatók a felhasználói jogosultságok, és hogyan biztosítható a tranzakciókezelés lehetősége a MySQL-adatbázisokban.
- Megismerheti az adatbázisok mentési stratégiáit, és adatvesztés esetén a konzisztens állapot visszaállítására alkalmas módszereket.

1.1.3 A tantárgy teljesítésének feltételei

A tananyag eredményes megtanulásához operációs rendszer szintű szakértő felhasználói tudás szükséges. Ez alatt a számítógépes hardver, az operációs rendszerek, általános alkalmazások, illetve a hálózatok – elsősorban az internet – tudatosan alkalmazott elméleti ismeretekkel megalapozott használatára vonatkozó kompetenciák meglétét értjük.

1.2 A KURZUS TARTALMA

A tananyag összesen 12 leckére tagolódik. A most olvasott 1. lecke bevezető információkkal és tanulási tanácsokkal szolgál, az utolsó, 12. lecke pedig összegzi a tananyagban megszerezhető ismereteket. A 2–11. leckék tartalmazzák a korábban felsorolt kompetenciák megszerzéséhez szükséges ismeretanyagot:

A 2. leckében tárgyaljuk az adatbázis-rendszerek fogalmát. Megismerjük az adatbázisok struktúrájának ANSI-SPARK modelljét, az adatbázis-kezelő rendszerek legfontosabb szolgáltatásait, fizikai és logikai adatfüggetlenség kritériumát.

Az adatbázisok szerkezetével kapcsolatos bevezető fogalmakkal foglakozik tananyagunk 3. leckéje. Itt ismerkedünk meg az egyed, a tulajdonság, az egyedtípus, a tulajdonságtípus, a kulcs, a kapcsolat, és a kapcsolattípus fogalmak meghatározásaival, valamint az adatstruktúrák modellezésének szintjeivel és módszereivel.

Tananyagunk az adatmodellezés középső, logikai szintjének ismertetésével folytatódik. A 4. lecke a relációs adatmodellt és annak szabályait mutatja be.

Az 5. leckében tanulunk a redundancia fogalmáról, annak káros hozadékairól: a változtatási, a bővítési, és a törlési anomáliákról, majd pedig a redundancia megszűntetésének széles körben ismert módjáról, a normalizálásról.

Az adatbázis-tervezés logikai szintjének megismerése után lépünk csak vissza a fogalmi szintre. A 6. leckében fogunk megismerkedni a fogalmi szint gazdag lehetőségeket kínáló modellezési technikájával az ER-modellel, valamint a modell ábrázolására alkalmas ER-diagrammal. A lecke végén tanulhatunk az ER-modell relációs adatmodellre való leképezésének szabályairól, amelyek ismeretében összetett koncepcionális modelleket is relációs modellre tudunk transzformálni.

A 7. lecke a relációs adatbázisok fizikai modellezésére, egyben létrehozására is alkalmas SQL nyelvet, és az Oracle ingyenes adatbázis-kezelő rendszerét, a MySQL-t mutatja be. Áttekintjük az SQL történetét, résznyelveit, nyelvtani szabályait, majd megismerkedünk a MySQL szerver telepítésével és konfigurálásával, és a karakteres kliens használatával.

Az adatbázisok fizikai kialakítását tárgyalja a tananyag 8. leckéje, amelyben az adatbázisok és táblák létrehozására használható SQL-parancsokat ismerheti meg.

Az adatbázis létrehozásakor megadott beállítások alapvetően meghatározzák a későbbi hatékony és üzembiztos működést, ezért a 9. leckében a táblák létrehozásával kapcsolatos alapvető tudnivalókat kiegészítjük. A lecke a MySQL táblamotorjaira, az indexelésre, az idegen kulcsokra, a hivatkozási integritás megőrzésére és a karakterkódolásra vonatkozó kérdéseket taglalja.

A 10. lecke az adatbiztonsággal foglalkozik. Ebben az anyagrészben olvashatunk a MySQL jogosultsági rendszeréről, a felhasználói szerepekről, az azonosításról, az jogosultsági szintekről és a szabályozás lehetőségeiről.

A 11. leckében a MySQL adatbázis-adminisztrátorok további feladataival a tranzakciókezelés lehetőségének biztosításával, a mentési stratégiákkal adatvesztés, sérülés esetén, illetve az utolsó konzisztens állapot visszaállításának módjaival ismerkedünk meg.

1.3 TANULÁSI TANÁCSOK, TUDNIVALÓK

Tananyagunk 2–11. leckéje felöleli az adatbázis-menedzsment témakör szükséges ismereteit. A leckék azonos szerkezetűek, fölépítésüket úgy igyekez-

tünk kialakítani, hogy a lehető legjobban segítsék az olvasót a megértésben, és a tananyag eredményes elsajátításában.

Minden lecke a **Célkitűzés, kompetenciák** szakasszal kezdődik. Ebben a bevezetésként is felfogható leckerészben találja meg az anyag áttanulmányozásával megszerezhető kompetenciákat, illetve itt olvashat a kitűzött célokról is. Célok alatt ne egyszerű felsorolást képzeljen el. Általában olyan problémákat, kérdéseket vetünk fel, amelyek az előző fejezetek alapján már Önben is megfogalmazódhattak. A lecke célja, hogy az új ismeretekkel megkeressük, és meg is adjuk a válaszokat a felsorolt problémákra. A bevezető kérdések ennek megfelelően a lecke logikai gondolatmenetét is meghatározzák. Arra kérjük, gondolkodjon együtt a tananyag írójával. A szöveg olvasásakor keresse a válaszokat, és ne lépjen tovább a leckéből, amíg azokat meg nem találta.

A célok után a lecke ismeretanyaga következik. A szövegben eltérő formátummal jeleztük a valamilyen szempontból kiemelkedő bekezdéseket, szövegrészeket. Az alábbi formátumokkal találkozhat:

Alkalmazások menüelemei, menüparancsok

Definíciók, fogalmak

Fájlrendszerben használt elérési utak

Feladatok

Fontos szöveg

Grafikus felületen található vezérlő elemek, objektumok

Gyakorlatok

```
Kódok, SQL mondatok
```

Megjegyzések

Nyomógombok, **forró billentyűk**

Összefoglaló kérdések

- Válaszok

A leckében található **fogalmakat**, **definíciókat** igyekezzen a legpontosabban megtanulni. Természetesen nem a szószerinti ismétlés, hanem a lényeg szabatos megfogalmazása a fontos.

Fordítson különös figyelmet a **fontos** szövegrészekre!

A **gyakorlatokat**, **feladatokat** minden esetben végezze el. Ezek ugyanis hozzásegítik ahhoz, hogy a szerzett ismereteket a gyakorlatban is képes legyen kamatoztatni.

A **kódokat**, SQL-mondatokat elsősorban a szemléltetés érdekében illesztettük be, azonban igyekeztünk úgy elhelyezni őket a tananyagban, hogy vágólapon keresztül közvetlenül is bemásolhatók legyenek a felhasználás helyére.

Ha a kódok felhasználásának ezt a módját választja, legyen óvatos!

A vágólapról történő beillesztés során gyakran jelentkeznek kisebb-nagyobb hibák, amelyek a karakterek konverziójából adódnak. Tipikusan ilyen az idézőjelek fölcserélődése, vagy a sorvégjelek okozta hibák. Mielőtt futtatja a vágólappal másolt kódokat, minden esetben végezzen szintaktikai ellenőrzést!

Minden egyes lecke végén megtalálja **Összefoglalás** szakaszt, amely logikusan követhető sorrendbe szedve, tömören összegzi a leckében található ismereteket. Mielőtt elolvasná az összegzést, a lényeg kiemelésével foglalja össze fejben a tanultakat! Ha valami nem jut eszébe, olvasson vissza bátran a tananyagban! Csak az önálló összefoglalás után vesse össze saját gondolatait a lecke összefoglalásával.

Az összegzést a frissen szerzett tudás ellenőrzésére használható **önellenőrző kérdések** követik. Soha ne mulassza el ezek áttekintését! Minden kérdéshez megtalálja a helyes választ is. Ezt lehetőleg ne olvassa el mindaddig, amíg önállóan nem sikerült felelnie a feltett kérdésre. A válaszok csupán arra valók, hogy ellenőrizze saját megoldása helyességét.

1.4 FORRÁSOK

A tananyag elsajátításához különböző forrásokat, állományokat biztosítunk. Amennyiben a tananyag mellé lemezmellékletet kapott, azon megtalálja a ezeket az fájlokat. Ha tananyagot elektronikus környezetben sajátítja el, a letölthető fájlok elérhetők a kurzus felületén.

2. LECKE: AZ ADATBÁZISOK ELEMEI, AZ ADATBÁZIS-TERVEZÉS SZINTJEI

2.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

Tananyagunk most következő leckéjében az adatbázis-kezelés ismeretanyagához kapcsolódó alapfogalmakat tekintjük át. Elsőként különbséget teszünk az adat és az információ között, majd megvizsgáljuk az aszinkron kommunikáció és az adatbázis-kezelés kapcsolatát.

Ezt követően az adatbázisok strukturális elemei, az egyed, a tulajdonság, a kapcsolat, az egyedtípus, és a tulajdonságtípus fogalmát tisztázzuk. Megvizsgáljuk a kapcsolatok számosságát, és különbséget teszünk az egy-egy, egy-több és több-több kapcsolattípusok között.

Rámutatunk az adatbázis-tervezés, -modellezés szükségességére, megismerjük a modellezés koncepcionális, logikai és fizikai szintjeit, valamint az azok közötti különbségeket.

A tananyag olvasása során igyekezzen választ találni az alábbi kérdésekre:

- Miért nem adható át információ a kommunikáció során?
- Miért mondhatjuk, hogy az adatbázis-kezelés aszinkron kommunikáció?
- Milyen strukturális elemek tárolódnak az adatbázisokban?
- Mit jelent az egyed, a tulajdonság, az egyedtípus, a tulajdonságtípus, a kapcsolat és a kapcsolattípus fogalma?
- Mi a különbség az különböző szintű adatmodellek között?

2.2 INFORMÁCIÓ, ISMERET, TUDÁS

Az ember számára a környező világ számtalan objektumból álló hatalmas halmaz. A halmaz elemeit különböző tulajdonságaik jellemzik. Egy objektum számos tulajdonsággal rendelkezhet. Az ember számára minden dolog önálló objektum, ami legalább egy tulajdonságában különbözik más objektumoktól.

Ebben az értelemben objektum a reggeliző asztalunkon gőzölgő kávé, az autónk, a személyi számítógépünk, de objektum egy vizsgatétel, egy matematikai szabály, objektum ez a tananyag, és objektumok vagyunk mi magunk is.

Az objektumok tulajdonságainak egy része kifejezetten az objektumot jellemzi, más részük viszont az egyéb objektumokhoz való viszonyról, kapcsolatról árulkodik.

A világ megfigyelése közben érzékeljük a bennünket körülvevő objektumokat, és létrehozzuk azok tudati képét. A tulajdonságok szerepe ebben a leképezésben azért fontos, mert az egyes objektumokat megismerhető és számunkra fontos tulajdonságaik halmazaként tartjuk számon.

1. ábra Objektum

Ez a szál virág vörös, hullámos szirmú, illata erős, szárán szúrós tüskék találhatók. Egy idős, kedves mosolyú, ősz hajú néni árulja, 10-10 szálból álló csokrokban.

A fenti mondatok valójában adatokat tartalmazó közlések, de most tekintsünk rájuk úgy, mintha egy megfigyelés tudati leképeződései lennének.

A mondatok egy bizonyos objektum, egy virág tulajdonságait (piros, szúrós, illatos...) írják le. Ezek között vannak olyanok is, amelyek más objektumokhoz való viszonyról (virágárus néni, és a csokor más virágai) árulkodnak.

2.2.1 Az információ, adat

Az objektumok tulajdonságaihoz jelentést társítunk. Ezt a jelentést nevezzük információnak. Az információhoz egy objektum valamely tulajdonságának értelmezésével jutunk. Az információ a tudat által egy tulajdonsághoz társított jelentés. Ilyen értelemben anyagtalan dolog.

> **Az információ egy objektum valamely tulajdonságának tulajdonított jelentés, amelyet a tulajdonság értelmezésével nyerhetünk.**

Egy objektumról megszerezhető összes információk halmaza az ismeret, tudásunk pedig ismereteink összessége. Eszerint, **tudásunk alapját** az objektumok megfigyelése, tanulmányozása, és tulajdonságaik értelmezése révén szerezett **információ alkotja**.

Az ember nemcsak a világ tanulmányozásával és közvetlen megfigyelésével, hanem kommunikációval is képes tudását gyarapítani. A kommunikáció során ugyanis információt szerezhetünk, amely ismereteinkbe épülve tudásunkat gazdagítja.

A kommunikációs folyamatban általában két fél vesz részt. Egyikük, az adó, aki ismereteinek egy részét igyekszik megosztani a másik féllel, a vevővel. Az adó üzeneteket küld a vevőnek, amelyek hatására a vevőben információ alakulhat ki.

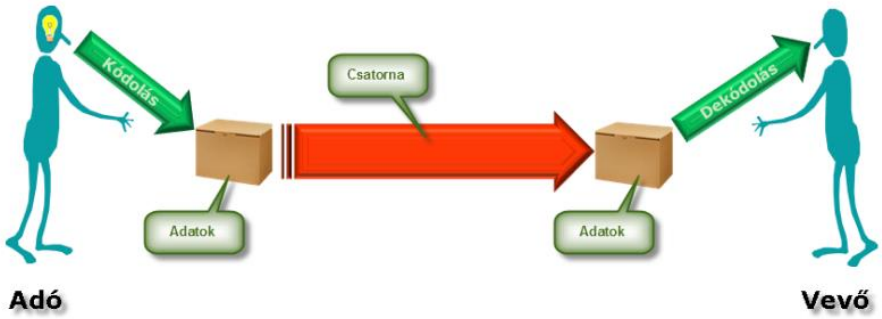
Az emberi individuumokat összeköti, ugyanakkor el is választja a kézzelfogható anyagi világ, amely a kommunikáció során az adó és a vevő közötti üzenetek átviteli közegeként, kommunikációs csatornaként szolgál. Az összekötés alatt azt értjük, hogy az üzenetek az anyagi világ közvetítésével adhatók át. Az elválasztás alatt pedig azt, hogy a kommunikációs csatornán csak anyagi természetű dolgok továbbíthatók. Az információról tudjuk, hogy nem anyagi kategória, csupán a tudatban létezik.

Emiatt a kommunikáció során az adónak kódolnia, fizikailag ábrázolnia kell az információt, ki kell alakítania annak anyagi reprezentációját, az adatot.

> **Az adat az információ anyagi reprezentációja, mások számára értelmezhető, feldolgozható, megérthető formátumú ábrázolása.**

Az információ ábrázolását kódolásnak nevezzük. A kommunikáció során az adó az információ kódolásával nyert adatokat tartalmazó üzeneteket ad át, amelyeket a vevő fogad, és értelmez. Ezt az értelmezést szokás dekódolásként

is említeni. Az értelmezés révén a vevő információhoz jut, amely ismereteibe épülve bővíti tudását.



2. ábra A kommunikáció Shannon-féle modellje

A kommunikáció kontextusában az adat és az információ egymással magyarázható fogalmak.

> **Az adat az információnak mások által értelmezhető, anyagi reprezentációja, az információ az adat értelmezésével keletkező jelentés.**

Ebben a megközelítésben csak akkor beszélünk információról, ha az értelmezéssel szerzett jelentés új elemmel egészíti ki a vevő ismereteit.

2.2.2 A kommunikáció hatékonysága

A kommunikáció során az adó fél szándéka mindig az, hogy a vevőben megfelelő információ alakuljon ki. A folyamat hatékonyságát azzal mérhetjük, hogy a vevőben létrejövő információ mennyire felel meg az adó szándékainak.

A hatékonyságot számos tényező befolyásolhatja:

- A zaj hatására a vevőhöz nem pontosan az az adat jut el, amit az adó küldött, tehát maga az üzenet sérül meg.
- A kódolás az információ ábrázolásának, formalizálásának technikája. Az üzenet struktúrája, formátuma, és az adatok sorrendje is jellemzi. Az adó akkor kódol hatékonyan, ha az üzenet alkalmazkodik a vevő várható dekódolási, értelmezési technikájához, meglévő ismereteihez és képességeihez.

- A dekódolás a kapott adatok értelmezésének folyamata, amelyet a vevő figyelme, előképzettsége, hangulata és értelmi képességei befolyásolnak.

2.2.3 Szinkron és aszinkron kommunikáció

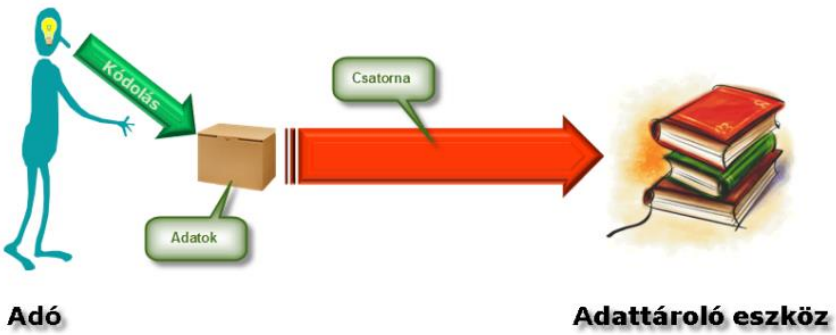
A kommunikációs folyamat különböző szempontok szerint vizsgálható,, ennek megfelelően számos jelzővel illethető. Az adó és a vevő kommunikációs csatornához való kapcsolódásának időpontját figyelembe véve szinkron és az aszinkron kommunikációt különböztethetünk meg.

A **szinkron kommunikáció** során a kommunikáló felelek egy időben kapcsolódnak a kommunikációs csatornához, így a vevő késleltetés nélkül kapja meg az adó üzeneteit. Szükség esetén a szerepek fölcserélődhetnek, a vevőből adó, az adóból vevő lesz, így azonnali válaszra, illetve párbeszédre van lehetőség.

Mindennapjainkban szinkron kommunikációt folytatunk például a személyes, illetve a telefonbeszélgetés során.

Az azonnali válasz, a visszacsatolás lehetősége miatt e kommunikációs formában az üzenetek általában rövidek és kevéssé strukturáltak.

Az aszinkron kommunikáció esetén a vevő még nincs jelen az üzenet elküldésekor. Elképzelhető, hogy az üzenet átadását, sőt az adó távozását követően, jóval később kapcsolódik a kommunikációs csatornához, és az üzenetet pedig csak kapcsolódás után kaphatja meg. A küldés és fogadás közötti időben meg kell oldani az üzenet tárolását. Ezt a feladatot mindig valamilyen adattároló eszköz segítségével valósítják meg.



3. ábra Aszinkron kommunikáció

Az aszinkron kommunikációra jellemző, hogy nem alakulhat ki valós idejű szinkron párbeszéd. Az adó az üzenetet megfogalmazásakor eleve fölkészül a vevő esetleges bizonytalanságaira. Az üzenetek általában hosszabbak, és strukturáltabbak, hiszen a vevőnek visszacsatolás nélkül is képesnek kell lennie az értelmezésre.

E kommunikációs forma hatékonyságában még nagyobb szerepet tölt be az adó kódolási technikája, üzenet megfelelő formátuma és struktúrája, az adatok sorrendje.

Napjaink kommunikációs szituációi között jellemzően aszinkron módon valósul meg az SMS- vagy az e-mail küldés. Amikor a feladó elküld egy elektronikus levelet, az a megfelelő elektronikus postaládába kerül, és mindaddig tárolódik, amíg a címzett le nem tölti azt.

2.2.4 Az adatbázis mint aszinkron kommunikációs eszköz

A könyvek, folyóiratok, sőt gyakorlatilag bármilyen adattárolásra alkalmas médium az aszinkron kommunikációs eszközök közé tartozik. Ezek után talán nem furcsa, hogy ide soroljuk az adatbázisokat is.

Az adatbázisok egy ismerethalmaz információit hordozó adatok tárolására és kezelésére alkalmas aszinkron kommunikációs eszközök. Az adó az információinak kódolásával létrehozhatja és az adatbázisban elhelyezheti, a vevő pedig kiolvashatja és értelmezheti az adatokat.

2.3 ADATBÁZISOK STRUKTURÁLIS ELEMEI

Az üzenet struktúrája nagyban befolyásolja az aszinkron kommunikáció hatékonyságát, ezért az adatbázisok szerkezete alapvetően meghatározza használhatóságukat. A struktúra jelentőségét még tovább fokozza, hogy az adatbázisokat több adó és több vevő is használhatja, akár egyidejűleg is.

Az adatstruktúra kiemelt jelentősége miatt az adatbázis-kezelést két nagy feladatcsoportra, az adatbázisok tervezésére és létrehozására, valamint azok használatára bonthatjuk.

A tervezés során megtervezzük az adatbázis leendő struktúráját, majd létrehozzuk az adatbázist. A felhasználáskor adatokat helyezünk el és kérdezünk le az adatbázisból.

Ahhoz, hogy később világosan értsük az adatbázis-tervezés lépéseit, tekintsük át, milyen szerkezeti elemekből alakíthatjuk ki azok struktúráját!

2.3.1 Egyed és tulajdonság

Az információ egy objektum valamely tulajdonságának jelentése, az adat annak formalizált ábrázolása. Az adatbázisok alapvető építőköve tehát a tulajdonságot leíró adat. A tulajdonságok az objektumok jellemzői, ezért értelmezésük csak az objektum ismeretében lehetséges. Az adatbázisokban együtt kell tárolnunk az objektumokat és az azokat jellemző tulajdonságokat.

Az adatbázis-kezelésben az **objektum szó helyett** az **egyed** kifejést alkalmazzuk. Az adatbázisok első két fontos strukturális eleme tehát az **egyed** és a **tulajdonság**.

A ***'924180OF',Andrássy Mária,(60) 832-4378,1011 Budapest, Kassa u. 94,andrassy.maria@gabriel.hu*** *mind tulajdonságok. Az a személy, akit jellemeznek, maga az egyed.*

> **Az egyed olyan objektum, amelynek valamilyen tulajdonságait tároljuk az adatbázisban. A tulajdonság olyan adat, amely az egyed valamilyen jellemzőjét írja le.**

2.3.2 Egyedtípus, tulajdonságtípus

Az adatbázisok különböző egyedeinek egyes tulajdonságai gyakran azonos jellemzőket írnak le. Az ilyen tulajdonságok azonos tulajdonságfajtába, úgynevezett **tulajdonságtípusba** tartoznak. Az **Andrássy Mária** és **Dallos Rebeka** tulajdonságok például nevek, a név tulajdonságtípusba tartoznak. A **(60) 832-4378**, és a **(30) 584-4036** számsorok a telefonszám tulajdonságtípusba sorolhatók.

Azok az egyedek, amelyek tulajdonságai azonos tulajdonságtípusba tartoznak, közös objektumhalmazba, úgynevezett **egyedtípusba** sorolhatók.

Az alábbi táblázat három egyed tulajdonságait tartalmazza. Az oszlopok tetején a tulajdonságtípusok megnevezése olvasható. Az egyedek egyes tulajdonságai ugyanazokba a tulajdonságtípusokba

tartoznak, ezért ezeket az egyedeket egy egyedtípusba soroljuk. Az egyedtípus neve lehet például személyek.

Igazolvány szám	**Név**	**Telefonszám**	**Cím**	**e-mail cím**
924180OF	Andrássy Mária	(60) 832-4378	1011, Budapest, Kassa utca, 94	andrassy.maria@gabriel.hu
094128IM	Dallos Rebeka	(30) 584-4036	3200, Gyöngyös, Mázsa utca, 5	dallos.rebeka@xmail.hu
680552DI	Galamb Huba	(60) 235-5859	3300, Eger, Kolmann utca, 55	galamb.huba@gabriel.hu

A tulajdonságtípus azonos jellemzőket leíró tulajdonságok halmaza, amelyre egy elnevezéssel hivatkozunk. Az egyedtípus az azonos típusú tulajdonságokkal jellemezhető egyedek halmaza.

Az adatbázisokban egyedeket és tulajdonságaikat tároljuk úgy, hogy a tulajdonságokat megnevezett tulajdonságtípusokba, az azonos típusú tulajdonságokkal rendelkező egyedeket pedig szintén névvel ellátott egyedtípusokba rendezzük.

2.3.3 Kapcsolat, kapcsolattípus

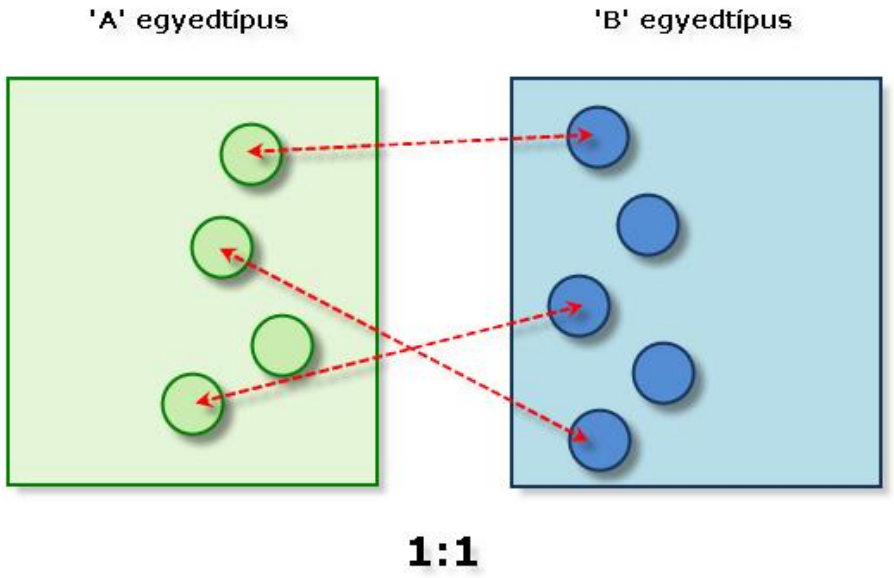
Az objektumok közötti viszonyok szintén ismereteink szerves részét képezik, ezért az adatbázisoknak az egyedek közötti kapcsolatokat is tükrözniük kell. Mivel egy egyed valamilyen másik egyedhez fűződő kapcsolata valójában az egyed tulajdonsága, a kapcsolatok tárolása nem okoz gondot.

Azt azonban tudnunk kell, hogy a tulajdonságokhoz és egyedekhez hasonlóan a kapcsolatok is tipizálhatók. Amikor egy egyedtípus valamelyik egyede egy másik egyedtípus egyedeivel van viszonyban, a kapcsolat a kapcsolat fontos jellemzője a **számosság**.

A számosság azt határozza meg, hogy az ’A’ egyedtípus egyes egyedei a ’B’ egyedtípus hány egyedéhez kapcsolódhatnak, illetve fordított irányban milyen a viszony.

Egy-egy kapcsolat

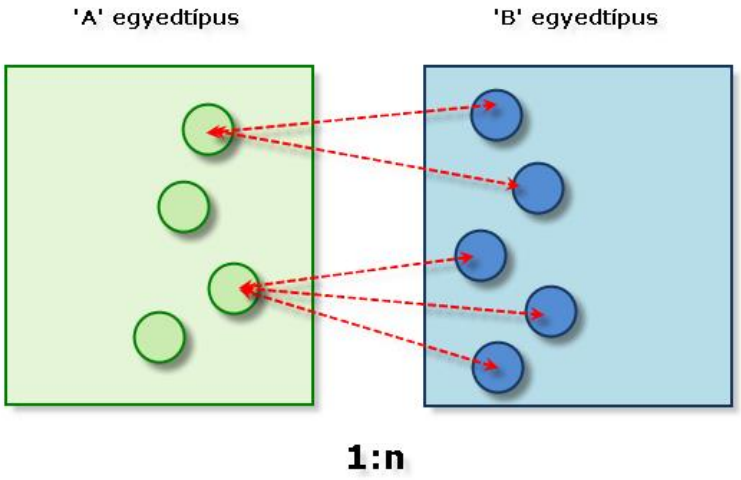
Egy-egy kapcsolatról beszélünk, ha bármely ’A’ egyedtípushoz tartozó egyed a ’B’ egyedtípusnak csak egyetlen egyedéhez kapcsolódhat és ez fordított irányban is igaz. Az egy-egy kapcsolatot 1:1 jelöléssel jelezzük.



4. ábra 1:1

Egy-több kapcsolat

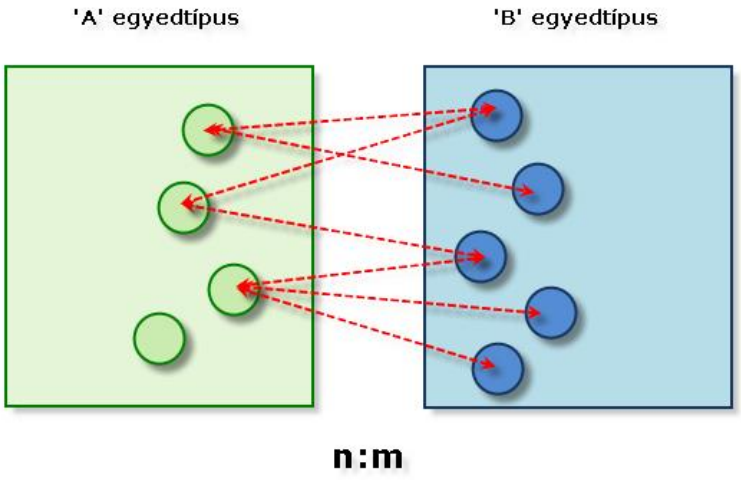
1:n formában jelöljük, és egy-több kapcsolatnak nevezzük azt a viszonyt, amikor az ’A’ bármelyik egyede ’B’-nek több egyedéhez is kapcsolódhat, de ’B’ egyedei csak egyetlen ’A’-hoz tartozó egyeddel lehetnek kapcsolatban.



5. ábra 1:n

Több-több kapcsolat

A több-több kapcsolat esetén mindkét egyedtípus egyedei több egyedhez kapcsolódhatnak a másik egyedtípusban. A ezt a kapcsolattípust n:m módon jelöljük.



6. ábra n:m

Az, hogy két egyedtípus között milyen kapcsolattípus van, elsősorban a kapcsolat jellegétől, okától függ.

Tegyük fel például, hogy egy ***munkahely osztályainak*** *adatait tároljuk az egyik egyedtípusban, a* ***dolgozók*** *jellemzőit pedig a másikban. Az egyedtípusok között azért van kapcsolat, mert minden osztálynak van a dolgozók egyedtípushoz tartozó osztályvezetője. A kapcsolat ilyenkor 1:1 típusú, hiszen egy osztálynak egy osztályvezetője van, és egy dolgozó csak egy osztályt vezethet.*

Ha azonban azért beszélünk kapcsolatról, mert minden dolgozó dolgozik valamilyen osztályon, akkor egy-több a kapcsolat típusa. Egy osztályon ugyanis többen dolgoznak, de egy dolgozó csak egy osztályhoz tartozik.

Tételezzük fel, hogy van egy különböző ***projektek*** *adatait tároló egyedtípusunk is, dolgozóink pedig különböző projektekben tevékenykednek. A két egyedtípus között n:m kapcsolat van, hiszen egy dolgozó vélhetően több projekthez, egy projekthez pedig több dolgozó is kapcsolódik.*

A fenti **dolgozó-osztály** kapcsolat jól mutatja, hogy két egyedtípus között akár több kapcsolat is lehetséges. Az itt bemutatott példák természetesen csak a jellemző esetekre épülnek. Speciális körülmények között ugyanaz a kapcsolat lehet más típusú is. Ha például egy munkahelyen előfordul, hogy ugyanaz a dolgozó több osztályt is vezethet, akkor dolgozó-osztály kapcsolat nem 1:1, hanem 1:n.

Később látni fogjuk, hogy minden ilyen tényezőt komolyan figyelembe kell venni az adatbázisok tervezésekor.

2.4 ADATBÁZIS TERVEZÉS SZINTJEI

Az legegyszerűbb adatbázisok csak néhány egyedtípus egyedeit és kapcsolatait tárolják, de nagyobbak akár százas nagyságrendű egyedtípust tartalmazhatnak, amelyek általában bonyolult kapcsolatrendszerrel kötődnek egymáshoz.

Az ilyen adatszerkezetek **csak alapos átgondolás**, részletes **tervezés** után készíthetők el.

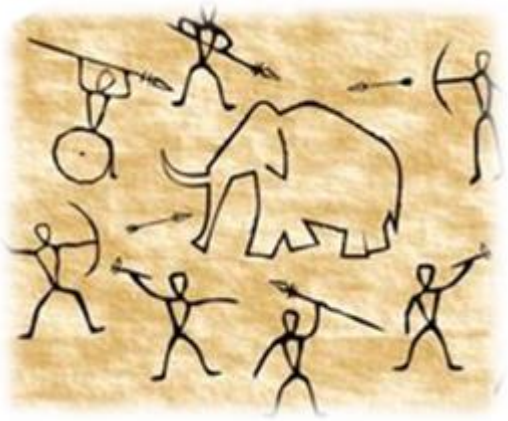
Az adatbázisok tervezésekor a leendő adatszerkezetet igyekszünk meghatározni. Eldöntjük, milyen egyedtípusokat, tulajdonságtípusokat tárolunk, hogyan kapcsolódnak egymáshoz az egyedtípusok, milyen adatszerkezeteket kell kialakítani a tárolás biztosítására, hogyan épülnek fel ezek az adatszerkezetek, és milyen úgynevezett meta-adatokra lesz szükség fizikai kialakításukhoz.

> **Végső célunk az, hogy a tervünk alapján fizikailag létező, az adatok biztonságos, perzisztens, konzisztens tárolását és kezelését lehetővé tévő tényleges adatbázist hozhassunk létre.**

2.4.1 Modellezés

A terv **modellezéssel**, az adatbázis elképzelésével, részleteinek elméleti kidolgozásával készül el. Mivel a végleges rendszer meglehetősen bonyolult is lehet, a modellezést több szinten végezzük el.

Egy adatbázis, de egyébként bármilyen rendszer modellezésekor annak **modelljét**, **elméleti leképezését** készítjük el. A **modell sohasem tartalmazza a modellezett objektum minden egyes részletét, csak azokat, amelyeket a modellezés szempontjából fontosnak tartunk**. A modellezésekor mindig kiemeljük a valóság fontos elemeit, tulajdonságait, és a modellt ezekkel írjuk le.



7. ábra Az ősember vadászatról alkotott modelljének ábrázolása

> **A** modellben **többé-kevésbé** elvonatkoztatunk **a valóságtól, a** jelentéktelen részleteket figyelmen kívül **hagyjuk, és** csak a lényegesekre koncentrálunk**. Ezt a gondolkodási műveletet nevezzük** absztrakciónak**.**

Természetesen megítélés kérdése, hogy milyen elemeket, jellemzőket tekintünk fontosnak. Modellezhetünk valamit úgy, hogy csak néhány jellemzővel törődünk, de alkothatják a modellt aprólékos részletek is. Ezért mondjuk azt, hogy a **modellek különböző szintűek** lehetnek. Modellünk annál magasabb szintű, minél erősebb absztrakció szükséges a modellezéshez. **Minél kevesebb részletre** figyelünk, minél inkább elvonatkoztatunk a rendszer konkrét elemeitől, **annál magasabb szintű absztrakcióról** és modellről beszélünk.

A **tervezéskor** általában nagyon **magas szintű modellből** indulunk ki, azaz csak a modellezett rendszer néhány elemére koncentrálunk. Ez biztosítja, hogy az első modell elkészítésekor ne vesszünk el a részletekben. Később a **részletek fokozatos feltárásával egyre alacsonyabb szintű modelleket** hozunk létre. Ezt mindaddig folytatjuk, amíg a modellünk közvetlenül felhasználhatóvá nem válik az adatszerkezet megvalósítására.

2.4.2 Modell ábrázolása

Egy modell csupán a megalkotójának tudatában létezik. Az adatbázisok megtervezésekor fontos, hogy a kialakított modellek ábrázolhatók legyenek.A modell ábrázolására sokféle technikát alkalmazhatunk. Beszélhetünk róla, leírhatjuk, ábrák formájában papírra vethetjük. A cél azonban mindig az, hogy a modell értelmezhető, megérthető legyen az érdekeltek számára.

Az értelmezhetőség érdekében a modellek reprezentációja általában valamilyen egyezményes formátum szerint történik. Az informatikai tervezés során kialakított modellek ábrázolására különböző formalizmusok, ábrázolási technikák születtek és terjedtek el.

Rövidesen látni fogjuk, hogy melyek az adatbázis-tervezés során létrehozott modellek ábrázolásának jellemzően használt technikái.

2.4.3 Az adatbázis-modellezés szintjei

Az adatbázisok modellezését három határozottan elkülöníthető szinten végezhetjük. A legmagasabb modellezési szintet koncepcionális, a középső szintet logikai, a legalsó szintet pedig fizikai modellezésnek nevezzük.

Koncepcionális szint

A legfelső, koncepcionális szinten az úgynevezett ER- (Entity-Relationship) modellt használjuk Az ER-modell csak arra koncentrál, hogy egy adatbázisnak egyedtípusokat, azok tulajdonságtípusait, más néven attribútumait, valamint az egyedtípusok közötti kapcsolatokat kell tárolnia.

A modell ezeknek az elemeknek a felhasználásával készül, és nem foglalkozik sem a tárolásukra használható adatszerkezetekkel, sem pedig a fizikai tárolással kapcsolatos problémákkal.

Logikai szint

A középső szinten az úgynevezett logikai adatmodelleket használjuk. A logikai adatmodellek közé tartozik a hálós, hierarchikus, az objektum-orientált és a relációs adatmodell. Tananyagunk következő leckéjében ez utóbbival foglalkozunk részletesen.

A koncepcionális modellel szemben a logikai adatmodellek megalkotásakor már azt is figyelembe vesszük, hogy milyen adatszerkezet formájában tárolhatók az egyedtípusok, tulajdonságtípusok, és kapcsolatok. A logikai adatmodellek tehát az adatszerkezetre, és az azzal kapcsolatos szabályokra is koncentrálnak.

Fizikai szint

Logikai szinten még mindig nem törődünk azzal, hogy hogyan tárolódnak az adatok, és hogyan biztosítható azok integritása, gyors és hatékony kezelése. Az adatok fizikai tárolására vonatkozó modellezés a legalsó, fizikai szinten történik meg.

2.5 ÖSSZEFOGLALÁS, KÉRDÉSEK

2.5.1 Összefoglalás

Mai leckénkben megismertük az objektum, a tulajdonság, az információ és az adat fogalmakat. Megtanultuk, hogy az információ az objektumok tulajdonságainak értelmezésével nyert jelentés, az adat pedig az információ, végső soron pedig egy tulajdonság mások által értelmezhető formátumú reprezentációja. Adatokra azért van szükség, hogy információinkat az anyagi világ közvetítésével megoszthassuk egymással.

Megtanultuk, hogy az adatbázis-kezelés valójában aszinkron kommunikáció, amelynek során egy jól meghatározott adatszerkezetben adatokat helyezhetünk el, majd azokat tetszőleges időben kiolvasva információhoz juthatunk. Az adatbázisokban egyedeket, azok tulajdonságait és kapcsolatait tároljuk úgy,

hogy a tulajdonságokat tulajdonságtípusokba, az azonos tulajdonságtípusokkal leírható egyedeket pedig egyedtípusokba soroljuk.

A leckében olvashattunk a kapcsolatok tipizálásáról, az 1:1, 1:n és n:m kapcsolattípusokról. Megtanultuk, hogy a kapcsolat számosságát nem csupán a kapcsolódó egyedtípusok, hanem a kapcsolat mibenléte is befolyásolja.

Leckénk végén az adatbázis-modellezés három szintjével, a koncepcionális, a logikai és a fizikai modellezésről olvashattunk.

2.5.2 Önellenőrző kérdések

1. Mi a különbség adat és információ között? Miért nem tárolhatunk információt?
    - Ha a két fogalmat egymással magyarázzuk, akkor az adat az információ mások által értelmezhető ábrázolása, az információ pedig az adat értelmezése során nyert jelentés. Az információ az emberi elme terméke és csak a tudatban létezik. Az adatbázisokban éppen ezért csak annak anyagi reprezentációját, az adatot tárolhatjuk.
2. Milyen strukturális elemek építenek föl egy adatbázist?
    - Egyedek, tulajdonságok, kapcsolatok, egyedtípusok, tulajdonságtípusok és kapcsolattípusok.
3. Mi a különbség 1:n és n:m kapcsolat között?
    - n:m kapcsolat esetén mindkét egyedtípus egyedei több egyedhez kapcsolódhatnak a másik egyedtípusban. Az 1:n kapcsolatnál azonban az egyik egyedtípus egyedei csak egy, a másik egyedtípus egyedei viszont több egyeddel is kapcsolatban állhatnak az ellenkező oldalon.
4. Milyen kapcsolattípus lehet az alábbi egyedtípusok között?
   Megrendelés – Árucikk
   Áru kategória - Árucikk
   Iskolai osztály - Osztályfőnök
    - A Megrendelés - Árucikk egyedtípusok kapcsolata n:m, mert egy megrendelésen több árucikk lehet, és egy áru több megrendelésben is előfordulhat.

Az Áru kategória – Árucikk kapcsolat általában 1:n, mert egy kategóriába több áru tartozik, az árucikkek viszont csak egy kategóriához kötődnek.
Iskolai osztály – Osztályfőnök egyedtípusok a legtöbb esetben 1:1 kapcsolatban vannak. Egy osztálynak egy osztályfőnöke van és egy tanár csak egy osztályban osztályfőnök. (Egyes iskolákban az osztályoknak több osztályfőnöke is van. Ilyen esetekben 1:n a kapcsolat.)

5. Hogyan nevezzük az adatbázis-modellezés legfelső szintjén alkalmazott modellezési technikát? Az adatbázis mely elemeit, tulajdonságait veszi figyelembe ezt a modell?
A koncepcionális szinten az ER-modellt használjuk, amelyben csak az egyedtípusok, tulajdonságtípusok és kapcsolatok segítségével alkotjuk meg a modellt.

6.

3. LECKE: KONCEPCIONÁLIS MODELLEZÉS, ER-MODELL

3.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

Az adatbázisok alapvető szerkezeti elemei meglehetősen egyszerűek, hiszen minden adatbázis egyedeket, azok tulajdonságait és kapcsolatait tartalmazza. Az egyedek a logikai adatmodelltől függően egyedtípusokká, a tulajdonságok pedig tulajdonságtípusokká szervezve tárolódnak. Az adatbázisok általában többféle egyedből épülnek fel, ezért sokszor nagyon sok egyedtípust és kapcsolatot, egyedtípusonként pedig számos tulajdonságtípust tartalmaznak. Szerkezetük emiatt meglehetősen bonyolult és nehezen tervezhető, áttekinthető. Ugyanakkor kialakításukkor egyáltalán nem mellékes elvárás, hogy a tervezésben résztvevő szakemberek, és az adatbázis leendő használói, a megrendelők egyaránt átlássák a készülő adatbázis tervét.

Mint minden termék, így a szoftvereszközök, és ezek között az adatbázisok is különböző absztrakciós szinteken tervezhetők. Minél magasabb absztrakciós szintet alkalmazunk, annál több, az adott szinten lényegtelen jellemzőt hagyunk figyelmen kívül. A legalacsonyabb szinten az adatbázis fizikai kialakításához és hatékony működtetéséhez szükséges összes tulajdonságot tartalmazó fizikai modellt használunk. A logikai szinten csak azokban a fogalmakban gondolkodunk, amelyek az egyedek, tulajdonságok, egyedtípusok, tulajdonságtípusok és kapcsolatok tárolásakor használt adatszerkezet írják le.

A modellezés legmagasabb, koncepcionális szintjén figyelmen kívül hagyjuk az adatszerkezeteket, és kizárólag az adatbázist fölépítő elemekre figyelünk. Arra koncentrálunk, hogy milyen egyedeket, tulajdonságokat, és egyedek közötti kapcsolatokat kell tárolnunk.

Mivel a koncepcionális szinten minden egyéb részlettől elvonatkoztatunk, a modell ábrázolt formája sokkal egyszerűbben áttekinthető, mint az alacsonyabb szintű modellek esetében. Ez a szakemberek számára az együttműködést, és az adatbázis későbbi továbbfejlesztését, a laikusok számára pedig a szerkezet megértését könnyíti meg. A magas absztrakciós szint ellenére, a kész modellek alapján alacsonyabb szintű modellek készíthetők. Egy koncepcionális modellnek alacsonyabb szintű modellé alakítása, leképezése csak világosan lefektetett szabályok alkalmazásával történhet. E szabályok természetesen léteznek, ezért a néhány kiegészítő adat megadásával készített koncepcionális modellből – akár szoftveres eszközökkel is – létrehozható az adatbázis fizikai

modellje. A koncepcionális modellből kész adatbázis-implementációt készítő alkalmazásokat nevezzük CASE eszközöknek.

Mint ahogyan az alacsonyabb szinteken, úgy a koncepcionális szinten is többféle modellezési technika létezik. Ezek mindegyikére igaz, hogy céljuk az egyszerűen, könnyen áttekinthető, ugyanakkor jelentésben gazdag modellezés emberi gondolkodáshoz közelálló eszközeinek biztosítása. Tananyagunk most következő leckéjében a legelterjedtebb koncepcionális modellezési technikával, az ER-modellel és ábrázolásával, az ER-diagrammal ismerkedünk meg.

A lecke anyaga alapján keressen választ az alábbi kérdésekre:

- Hányféle elemre redukálhatók az ER-modell által nyújtott lehetőségek.
- Mi a különbség gyenge és erős egyed között?
- Milyen lehetőségeket nyújt a modell az attribútumok ábrázolásában?
- Miért fontosak a kapcsolatok tulajdonságai?
- Hogyan írható le az ER-modellben a kapcsolatok számossága?
- Mit jelent egy kapcsolatban a kötelező és opcionális részvétel?
- Mit jelent az öröklődés, és hogyan ábrázolható az ER-modellben?
- Mi a különbség a specializáció és az általánosítás között?

3.2 AZ ER-MODELL

A legelterjedtebb koncepcionális modellezési technikát dr. Peter Chen dolgozta ki és publikálta 1976-ban, tehát jóval a relációs adatmodell megalkotása után. Chen nem titkolt célja éppen az volt, hogy a relációs adatbázisok tervezését és szerkezetük ábrázolását egyszerűsítse. A magas szintű absztrakció miatt modellje ugyanakkor átalakítható hierarchikus, hálós, és objektumorientált logikai adatmodellre is.

Ahogyan neve is sejteti, az ER- (Entity-Relationship) modell egyedek, tulajdonságaik és kapcsolataik segítségével igyekszik ábrázolni az adatbázis szerkezetét.

> **Mielőtt azonban továbbhaladnánk, szögezzük le, hogy a modell nem tesz különbséget az egyed és az egyedtípus között. Egyedet vizsgál ugyan, de azok jellemzői alapján automatikusan az egyedet tartalmazó halmazra, az egyedtípusra általánosít. A modellt készítő tervező tehát az egyedtípus, tulajdonságtípus, és az egyedtípusok közötti kapcsolat fogalmak felhasználásával gondolkodhat.**

Hogy ne sértsük az ER-modellben használt terminológiát, de világosan jelezhessük a halmaz és az elem közötti különbséget, a koncepcionális modellezésről szóló fejezetünkben az eredeti fogalmak megtartásával az **egyed** szóval hivatkozunk az **egyedtípusra**, és az **egyed-előfordulás** kifejezéssel az **egyedre**. A **tulajdonság** szót használjuk a **tulajdonságtípusra**, és **tulajdonság-előfordulásnak** nevezzük a **tulajdonságot**.

A modell egyszerű elemekből építkezik, azonban az egyedek, tulajdonságok és kapcsolatok számos formáját különbözteti meg. Emiatt a valóság igen kifinomult modellezésére is alkalmas.

Az ER-modell erőssége a vele szerves egységet alkotó ábrázolási technika, az ER-diagram, amely hétköznapi síkidomok, vonalak és elnevezések segítségével biztosítja az ábrázolás és megértés egyszerűségét:

- Az egyedeket **téglalapokkal**, tulajdonságaikat **ellipszisekkel**, a kapcsolataikat pedig **rombuszokkal** ábrázoljuk.
- Minden **elemet elnevezünk**, a neveket a síkidomokba írjuk. Az egyedek és tulajdonságok nevei főnevek (dolgozó, osztály, név, lakcím…), a kapcsolatok pedig általában (de nem kötelezően), a viszony jellegét leíró igék (dolgozik, lakik…).
- Az egyedekhez kötődő tulajdonságokat és a kapcsolódó egyedeket **vonalakkal kötjük össze**.
- A kapcsolatokat az összekötő vonalra rajzolt **rombusszal** ábrázoljuk.

Nem tűnik túlságosan fontosnak, mégis nagy jelentősége van az elnevezések megválasztásának. Az ER-modell nem ad semmiféle megkötést az egyedek, tulajdonságok és kapcsolatok megnevezésére, azonban célszerű beszédes, és a végleges, fizikai adatbázist kezelő rendszerben is használható neveket választani. A nevek álljanak egy szóból. Ha netán több szóból alkotott kifejezések, akkor a nyelvtani szabályokat figyelmen kívül hagyva rövidítsük és írjuk egybe őket. Lehetőleg ne alkalmazzunk grafikus karaktereket, és ékezetes betűket is csak akkor használjuk, ha az adatbázis működtetésére kiválasztott DBMS ezt lehetővé teszi!

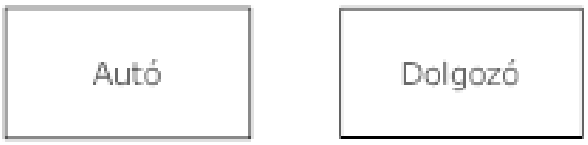
A következőkben az ER-modell fogalmait és ábrázolási technikáikat tekintjük át.

3.2.1 Az egyedek és ábrázolásuk

Mint említettük az ER-modellben az egyes adatbáziselemek különböző fajtáit használhatjuk. Az egyedek például kétfélék is lehetnek. **Erős vagy normál**

egyednek nevezzük azt az egyedet, amely **rendelkezik** az egyed-előfordulások teljes biztonsággal történő, egyértelmű megkülönböztetésére alkalmas tulajdonságtípussal, **azonosítóval**. Erős egyed például az autó, mert van rendszáma, alváz- sőt motorszáma is. Ezek mindegyike egyértelműen megkülönbözteti az egyes autókat. Erős egyed egy cég adatbázisában a dolgozó, mert minden dolgozónak van személyiigazolvány-száma (TB-száma, adószáma stb.).

Az erős egyed ábrázolására az egyed nevét tartalmazó téglalapot használunk.



8. ábra Erős egyedek

A gyenge egyednek nincs önálló azonosítója, de minden egyed-elfordulása azonosítható egy másik, erős egyed valamelyik egyed-előfordulásával, illetve annak azonosítójával együtt.

Tegyük fel, hogy adatbázisunkban egy több emeletes lakóépület emeleteinek és lakásainak jellemzőit szeretnénk tárolni.

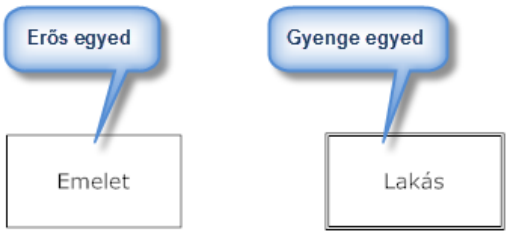
Az emelet egyedtípus tulajdonságai (sorszám, lakások száma, képviselő…) közül az emelet sorszámát használjuk azonosítójaként. A lakásokat (AjtóSzám, Alapterület, SzobaSzám) az emeletenkénti sorszámozással létrehozott ajtószám különbözteti meg. Minden emeleten van 1-es, 2-es 3-as stb. lakás.

Belátható, hogy az ajtószám nem alkalmas az épület összes lakásának megkülönböztetésére, hiszen a számozás emeletenként ismétlődik. Ha azonban az emeletszámokkal együtt használjuk ajtószámokat, akkor már lakásonként is egyedi azonosítóhoz jutunk.

> **Parciális kulcsnak nevezzük a gyenge egyed azon tulajdonságát, amely az erős egyedtípus azonosítójával együtt használva egyértelműen megkülönbözteti a gyenge egyed előfordulásait**

Példánkban az ajtószám a parciális kulcs.

A gyenge egyed ábrázolásakor dupla vonalas téglalapot használunk.

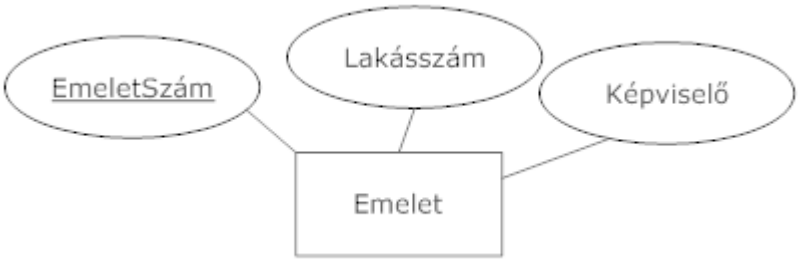


9. ábra Erős és gyenge egyed

Később látni fogjuk, hogy a gyenge és erős egyed közötti kapcsolatban is más jelölést használunk, mint az erős egyedek kapcsolata esetén.

3.2.2 Tulajdonságok és ábrázolásuk

Az ER-modellben tulajdonság vagy attribútum alatt egy egyed valamelyik tulajdonságát értjük. Ábrázolásukra nevüket is tartalmazó ellipszisekkel használunk, amelyeket vonallal kötünk a megfelelő egyedhez.



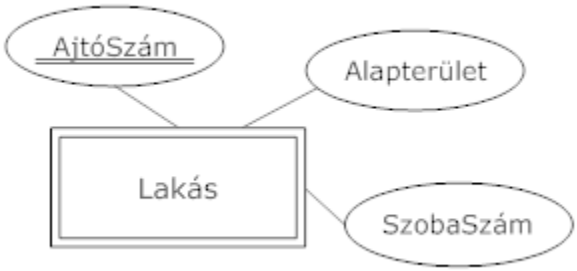
10. ábra Egyed és tulajdonságai

Azonosító

Az azonosító kiemelt fontosságú tulajdonságtípus, ezért az erős egyedekben aláhúzással jelöljük (EmeletSzám).

Parciális kulcs

A gyenge egyedek részleges megkülönböztetésére alkalmas mezőt dupla vonallal húzzuk alá.



11. ábra Parciális kulcs ábrázolása

Összetett attribútum

Rövidesen látni fogjuk, hogy a **relációs adatmodell** előírja, hogy a tulajdonságok **csak atomi értékek** lehetnek, azaz egy attribútum nem állhat sem több azonos, sem több különböző típusú elemből. Mivel az **ER-modell** magasabb absztrakciós szinten modellezi az adatbázist, semmiféle **ilyen megkötést nem tartalmaz**. Egyaránt megengedi az összetett és többértékű attribútumok használatát is. Ez egyrészt az egyszerű modellezést, másrészt az egyéb logikai adatmodellek felé történő átalakítást könnyíti meg.

Az **összetett attribútumok** több, különböző tulajdonságra bontható attribútumok. Tipikusan ilyen a lakcím, amely általában irányítószám, település, utca és házszám tulajdonságokra bontható. Ábrázoláskor az összetett tulajdonságok ellipsziseihez az azokat fölépítő elemi tulajdonságok ellipsziseit kapcsoljuk.

Többértékű attribútum

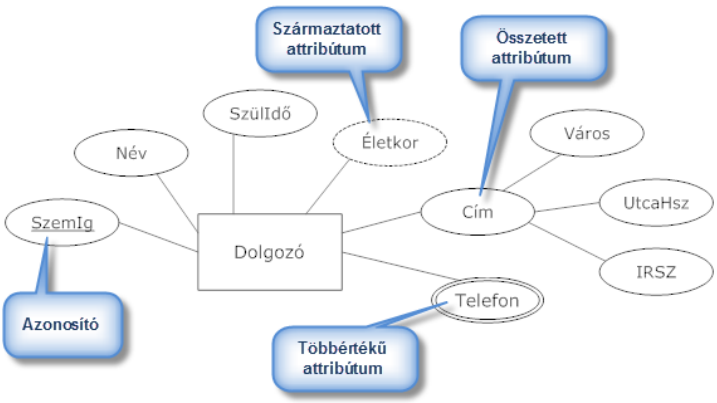
A **többértékű attribútum** olyan tulajdonság, amely egyedenként több, azonos típusú értéket is tartalmazhat. Ilyen lehet például a telefonszám, hiszen egy embernek több telefonja, ennek megfelelően több száma is lehet. Hasonló-

an többértékű lehet a nyelvtudás. Egy ember több különböző nyelven is beszélhet.

Származtatott attribútum

Előfordulhat, hogy egy adatbázisban olyan tulajdonságra is szükség van, amelynek értéke csak az egyed egy másik tulajdonságának ismeretében számolható ki. Az ilyen tulajdonságokat nevezzük **származtatott attribútumoknak**. Ha például tároljuk a dolgozók születési idejét, akkor életkoruk, az aktuális évszám és a születési időből vett évszám különbségeként határozható meg.

A származtatott attribútumokat szaggatott vonalas ellipszissel jelöljük.



12. ábra Különböző attribútumok jelölése

A fenti ábra a **`dolgozók`** egyedhez kapcsolódó különböző attribútumokat ábrázol.

3.2.3 Kapcsolatok

A kapcsolatok az adatbázis egyedei közötti viszonyt fejezik ki. Talán ez az az adatbázis elem, amelynek modellezésében a legszínesebb lehetőségek állnak rendelkezésünkre. Míg például a relációs adatmodellben csak a számosságuk alapján (1:1,1:n, n:m) teszünk különbséget, addig az ER modellben a **kötelezőség**, a kapcsolat **fokszáma**, az összekapcsolt **egyedek fajtája** (erő-erős, erős-gyenge), sőt az egyed esetleges **önmagával való** kapcsolata is jelölhető. Ezeken kívül a modellezést megkönnyítő jellemző, hogy a kapcsolatoknak az egyedekhez hasonlóan attribútumai lehetnek.

Az ER-modellben a kapcsolatot általában a viszony jellegét leíró igével nevezzük meg. A nevet a kapcsolódó egyedek közé húzott vonalra rajzolt rombuszba írjuk.

13. ábra Kapcsolat Chen-féle jelölése

Itt kell megjegyeznünk, hogy a kapcsolatok jelölésére számos különböző technikát használnak. A rombusszal ábrázolt kapcsolat az eredeti, Chen-féle jelölésnek felel meg. Akkor célszerű használni, ha a kapcsolatnak vannak attribútumai. Ilyenkor azokat úgy kötjük a rombuszhoz, mint az egyedek esetében a téglalaphoz. Más ábrázolásokban nem használnak rombuszt, hanem egyszerűen vonallal kötik össze az egyedeket és arra írják a kapcsolat nevét.

Kapcsolat számossága

A kapcsolatokat eddig csak az alapján tipizáltuk, hogy a kapcsolódó egyedek előfordulásai hány egyedhez kötődhetnek a másik egyedben. Az 1:1, 1:n és n:m kapcsolattípusok a kapcsolatban való részvétel maximumát határozzák meg, de semmilyen kitételt nem fogalmaznak meg a minimális részvétellel kapcsolatban. Az ER-modellben erre is lehetőség nyílik.

Ha például azt mondjuk, hogy a ***Dolgozó-Autó*** *kapcsolat 1:n, az azt jelenti, hogy bármelyik dolgozónak* ***lehet több autója*** *is, de egy bizonyos autó* ***maximum egy dolgozó*** *tulajdona* ***lehet****. Az 1:n kapcsolat nem árulja el, hogy minden dolgozónak van-e autója, és hogy minden autónak van-e tulajdonosa a dolgozók között.*

Kötelező, és opcionális részvétel

Míg a számosság a kapcsolatban való részvétel maximumára, addig a kötelezőség a minimális részvételi számra vonatkozó kritériumot fogalmaz meg. ’A’ és ’B’ egyed kapcsolatában a ’B’ oldalán jelezzük, hogy az egyes ’A’ egyed-előfordulásokhoz minimum és maximum mennyi ’B’ egyed-előfordulás kapcsolódik.

Az ’A’ oldal mellett az jelezzük, hogy a ’B’ egyed-előfordulásokhoz hány ’A’ egyed-előfordulás kapcsolódhat.

> **Egy kapcsolatban akkor nevezzük kötelezőnek az ’A’ egyed részvételét, ha nem lehet olyan egyed-előfordulása, amelyikhez nem kapcsolódik a ’B’ egyed egyetlen egyed-előfordulása sem.**
> **’A’ részvétele opcionális, amennyiben lehet olyan egyed-előfordulása, amely nem vesz rész a kapcsolatban.**

Ez a jelölés

Dolgozó (1,1)–(0,n) Autó

például azt jelenti, hogy egy autóhoz 1 és csakis 1 (1,1) dolgozó, egy dolgozóhoz pedig 0 vagy több (0,n) autó kapcsolódhat.

A kötelezőségről mindig a **túloldal minimális részvétele** árulkodik.

Ha rendre megvizsgáljuk mindkét egyed előfordulásainak lehetséges kapcsolatait, akkor azt látjuk, hogy minden autóhoz pontosan egy dolgozó kapcsolódik, azaz nincs olyan autó, amelyiknek ne lenne tulajdonosa. Az ellenkező oldalon azonban lesznek olyan dolgozók, akikhez több autóhoz is kapcsolódik (több kocsi tulajdonosai), de lesznek olyanok is, akiknek egy járművük sincs.

A példában az autók részvétele kötelező, a dolgozóké opcionális.

> **Kötelező részvételt totálisnak, az opcionális részvételt parciálisnak is nevezzük.**

Kötelező vagy totális részvételről beszélünk akkor, amikor egy egyed minden előfordulása részt vesz a kapcsolatban. Ilyenkor a kapcsolat másik oldalán nem lehet 0 a minimális részvétel.

Opcionális, vagy parciális egy egyednek a részvétele akkor, ha van olyan előfordulása, amelyik a másik egyed egyetlen előfordulásához sem kapcsolódik. Ezt a másik oldal 0 minimális részvétele jelzi.

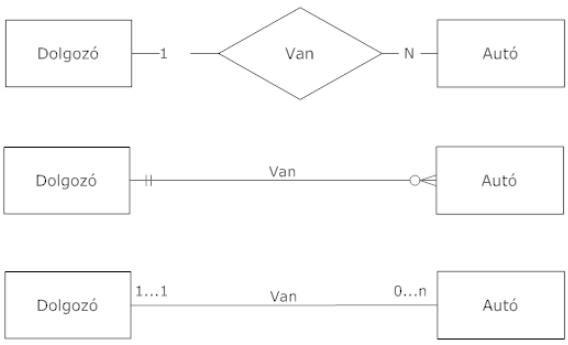
Számosság és kötelezőség ábrázolása

Mint említettük, a kapcsolatok ábrázolására több jelölés is létezik. A legelterjedtebbek az eredeti Chen-féle, a Backman-, a Crow's Foot (varjúláb), és a Martin-féle jelölések. Ezek az alábbi táblázatban látható jelekkel ábrázolják a minimális és maximális részvételt.

	Chen	Backman	Crow's Foot	Martin
0				0
egy	1			1
több	n			* v. n
min=0 ; max=1	1			0...1 v. (0,1)
min=1; max=1	1			1...1 v. (1,1)
min=0; max=n	n			0...* v. (0,n)
min=1; max=n	n			1...* v. (1,n)

14. ábra Számosság és kötelezőség jelölése

A legegyszerűbb, de a kötelezőség jelölésre alkalmatlan a Chen-féle 1,n jeleket használó módszer. Akkor használjuk, ha a minimális részvételnek nincs jelentősége a kapcsolatban. A Backman-módszer már biztosítja a számosság jelölését, de a varjúláb jelöléssel ellentétben nem túl szemléletes. A Crow's foot jelölésben a kapcsolatot ábrázoló vonal végeire helyezzük el a megfelelő szimbólumokat. A vonal közepe felé a minimális, széle felé a maximális részvételt jelöljük mindkét oldalon. A Martin-jelölésben a megfelelő helyre írt 0,1,n vagy 0,1,* szimbólumok jelölik a nulla, az egy, illetve több értékeket.



15. ábra Chen-, Crow's foot és Martin-féle jelölé-

sek

Rekurzív kapcsolat

Az egyedek közötti kapcsolat érdekes formája a rekurzív kapcsolat. Egy viszonyt akkor nevezünk rekurzívnak, ha az egyed előfordulásai ugyanannak az egyednek más előfordulásaihoz kapcsolódhatnak. Ilyenkor egyed valójában önmagához kapcsolódik.

16. ábra Rekurzív kapcsolatok

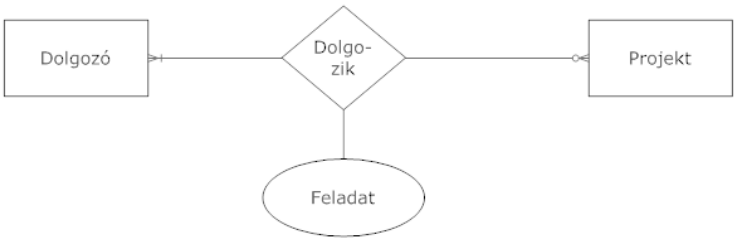
A fenti ábra két ilyen kapcsolatot mutat be. Az első esetben azt jelezzük, hogy melyik személynek ki a házastársa. Mivel egy személy egyetlen másik személlyel lehet házastársi kapcsolatban, a viszony rekurzív. A kapcsolat mindkét oldalon (0,1) jelöléssel jelzi, hogy egy embernek 0 vagy 1 házastársa van egy időben.

A másik példa az anya-gyermek kapcsolatot demonstrálja. Egy személynek nulla vagy több gyermeke lehet, de egy gyermeknek csak egy édesanyja van.

Kapcsolat attribútumai

Az ER-modell nagyszerűsége, hogy megengedi, hogy egy kapcsolat az egyedekhez hasonló tulajdonságokkal rendelkezzen. A példa bemutatja, hogy bizonyos esetekben nagyon is szükség van erre.

Tegyük fel, hogy cégünk dolgozói különböző projekteken dolgoznak. Adatbázisunkban egy-egy egyedtípusban tároljuk a dolgozók és a projektek adatait. Meg kell oldanunk a munkatársak egyes projektekben ellátott feladatának rögzítését is.



17. ábra Dolgozó-Projekt kapcsolat

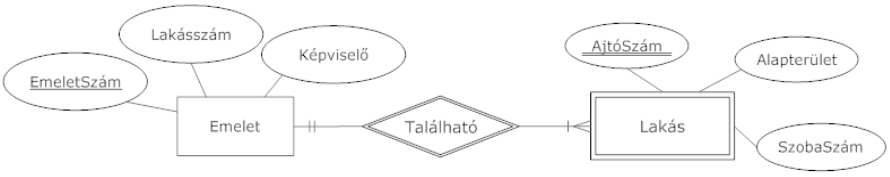
A megoldást a fenti ábra mutatja. A dolgozók és projektek **n:m** kapcsolatban állnak, hiszen egy dolgozó több projektben is részt vehet, és projektenként is több munkatárs tevékenykedhet. A dolgozók részvétele parciális (nem mindenki vesz részt a projektmunkákban), a projektek részvétele viszont totális, hiszen minden projekten legalább egy munkatárs dolgozik.

A feladat attribútumot egy egyedhez sem tudjuk rendelni, hiszen az kifejezetten a kapcsolatot, egy dolgozó egy bizonyos projektben kifejtett tevékenységét jellemzi.

Az ábrán együtt használtuk a Chen-féle kapcsolat és a Crow's foot számosság jelölést. Ez egyáltalán nem szokatlan az ER-diagramokon. Ha a megrendelő vagy a munkahelyi vezetőnk máshogyan nem rendelkezik, saját terveinkben is nyugodtan használjuk azt a jelöléskombinációt, amelyet a legcélravezetőbbnek, legszemléletesebbnek tartunk. Igyekezzünk azonban következetesek maradni.

Erős-gyenge egyed kapcsolata

A gyenge egyedek előfordulásai csak egy másik, erős egyed előfordulásaival együtt azonosíthatók. Ez azt jelenti, hogy a gyenge egyed mindig kapcsolódik egy erős egyedhez. Az ilyen kapcsolatot mindig dupla vonalas rombusszal jelöljük.



18. ábra Erős és gyenge egyed kapcsolata

Kapcsolat fokszáma

Az egyedek kapcsolatának jellemzésére szoktuk említeni a kapcsolat fokszámát. A fokszám azt határozza meg, hogy hány egyed vesz rész a kapcsolatban. A legtöbb kapcsolat bináris, azaz két egyedet, míg a terner kapcsolat hármat köt össze. Az unáris kapcsolatot nevezzük más néven rekurzív kapcsolatnak. Ebben csak egy egyed vesz részt.

3.2.4 Ábrázolás attribútumok nélkül

A modell kisebb területű ábrázolása érdekében előfordul, hogy egyes egyedeket attribútumaik nélkül ábrázoljuk. Akkor használjuk ezt a jelölést, ha az ábrázolásban az egyedek közötti kapcsolatra szeretnénk helyezni a hangsúlyt.

3.3 EER-MODELL

A koncepcionális modellezési szinten használható Entity Relationship modell alkalmas a hierarchikus, hálós, és relációs adatmodellekben megfogalmazható adatszerkezetek koncepcionális modellezésére, azonban nem kezeli az öröklődés problémáját. Ugyanakkor ez a kérdés az objektumorientált logikai adatmodell, és egyáltalán: az objektumorientált szemlélet egyik sarkalatos pontja.

Az ER modell, ’90-es évek elején kidolgozott bővítése, az Enhanced Entity Relathionship (EER) modell a fő- és alosztályok bevetésével lehetővé tette az öröklődés és a polimorfizmus adatszemléletű modellezését.

> **Az EER alkalmas olyan modellek készítésére és ábrázolására, amelyekben az egyes egyedek örökölhetik más egyed tulajdonságait és kapcsolatait, de azokat kibővítve saját, speciális attribútumokkal és kapcsolatokkal is rendelkezhetnek. A tulajdonságok örökölhetősége a specializációt, a többszörös öröklődés pedig az általánosítás megvalósítását biztosítja.**

3.3.1 Alosztályok, főosztály

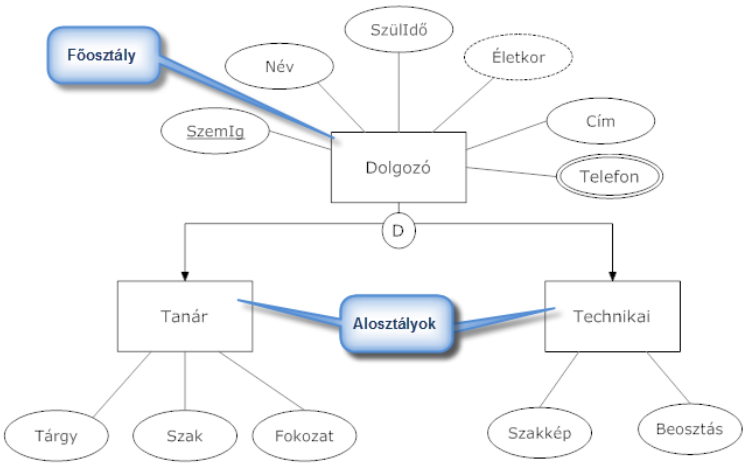
Egy egyed előfordulásainak lehet olyan részhalmaza, amelynek elemei a teljes halmazra jellemző tulajdonságoknál több attribútummal rendelkeznek. Az ilyen esetekben a teljes halmazt **főosztálynak**, a részhalmazt **alosztálynak** nevezzük.

A **főosztályban** csak a minden előfordulást jellemező **általános tulajdonságokat** hagyjuk. Az **alosztályban** pedig csak a **részhalmaz speciális attribútumait**

helyezzük el. A **főosztályban minden egyed-előfordulás**, az **alosztályban** csak a **részhalmaz** előfordulásai tárolódnak. Ez azt jelenti, hogy a részhalmaz egyed-előfordulásai a fő- és alosztályban is szerepelnek majd, de a főosztályban csak az általános, az alosztályban pedig csak a speciális attribútumaik jellemzik őket.

Példaként vegyünk egy olyan adatbázist, amely egy iskola dolgozóit tartja nyilván. A ***dolgozók*** *az összes dolgozót tárolja, és olyan attribútumokkal rendelkezik, amelyek minden egyes dolgozóra jellemzőek (****SzemigSzam, Név, SzülIdő, Életkor, Cím, Telefon****…). A dolgozók egy része technikai munkakörben, mások tanárként tevékenykednek. A technikai dolgozók esetén szükségünk van a szakképzettség és beosztás tulajdonságtípusokra, a tanárok esetén pedig a szak, a tudományos fokozat és a tantárgy jellemzőkre is. Ha mindezt egy egyedben oldjuk meg, akkor a tanárok esetében a technikai dolgozókra jellemző tulajdonságok, a technikai személyzet esetében pedig a szak, a tudományos fokozat és a tanított tantárgy attribútumok maradnak üresen.*

Az ilyen pazarlás elkerülése érdekében az összes dolgozóból főosztályt, a tanárok és a technikaiak egyedeiből egy-egy alosztályt készítünk.



19. ábra Főosztály és alosztályok

Ábrázolás

A fő és alosztályok ábrázolásakor a főosztályból többágú nyilat húzunk az alosztályokhoz. A nyíl elágazásába rajzolt körben az öröklődés típusára utaló betűjelet (D, O, U) helyezzük el (*a magyarázatot lásd alább..*).

Specializáció

Specializációról beszélünk akkor, ha egy főosztályból egy vagy több speciális tulajdonságokkal rendelkező alosztály emelhető ki. Erre láttunk példát az előbb. Előfordulhat, hogy egy alosztály egyedei a főosztály másik alosztályában is szerepelhetnek. Lehetséges az is, hogy az egyik részhalmazba tartozás kizárja a másikban való részvételt.

Az első esetben az alosztályok átfedik (overlap), a másodikban kizárják (disjoint) egymást. Ezeket a lehetőségeket jelölik a nyíl elágazásában lévő körbe írt D illetve O betűk.

Általánosítás

Az öröklődés másik esete, amikor két egymástól eltérő főosztálynak közös alosztálya van. Ez akkor fordulhat elő, ha két különböző egyedtípus egyedeinek (mindnek, vagy csak néhánynak) közös tulajdonságai is vannak.

A két főosztály egyedei számára ilyenkor közös alosztályt hozunk létre, amelyben csak a mindkét egyedtípusra jellemző tulajdonságokat tároljuk. Az így létrehozott alosztályt **kategóriának** nevezzük. A kategória egyed-előfordulásai csak az egyik, vagy csak a másik főosztályban szerepelnek, és öröklik annak tulajdonságait, azonban rendelkeznek a kategória speciális tulajdonságaival is.

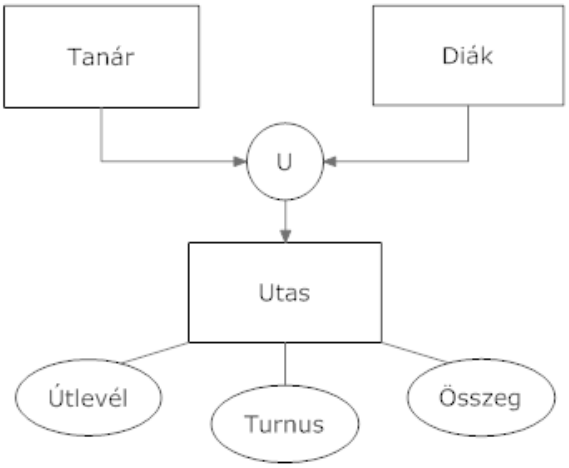
A kategóriát a specializációhoz hasonlóan ábrázoljuk, azonban a nyíl most két főosztályból indul, majd az ágak találkozása után az alosztály felé mutat. A csomópontban elhelyezkedő kör az uniót jelző **U** betűt tartalmazza. Ez arra utal, hogy a kategória egyed-előfordulásai a főosztályok egyed-előfordulásaiból képzett részhalmazok uniójaként adhatók meg.

Példaként ismét képzeljünk el egy iskolát, amelyben külön egyedekben tároljuk a ***tanárok*** *és* ***tanulók*** *egyed-előfordulásait. Erre azért van szükség, mert az oktatókról egészen más jellegű adatokat kell nyilvántartani, mint a diákokról.*

Az iskola három turnusban külföldi tanulmányutat szervez, amelyen tanárok és tanulók is részt vehetnek. Minden résztvevőről tá-

rolni kell az ***útlevele számát, az útra befizetett összeget****, és hogy melyik* ***turnusban*** *utazik.*

A kategória jelölését az alábbi ábrán láthatjuk.



20. ábra Általánosítás

3.4 ER-MODELLEZÉS

Az ER- és az EER-modellek megismerése után könnyen készíthetünk kisebb koncepcionális modelleket. A nagyobb adatbázisok tervezése természetesen gyakorlatot, tapasztalatot igényel. Miután az olvasó szert tett az említett kompetenciákra, minden bizonnyal kialakul saját modellezési technikája. A kezdéshez azonban érdemes megfogadni az alábbi egyszerű tanácsokat.

- Elsőként mindig azt próbáljuk elképzelni, hogy milyen egyedek (egyedtípusok) lesznek az adatbázisban. Egyelőre ne törődjünk az attribútumokkal, csak rajzoljuk meg az egyedek téglalapjait.
- Ezt követően gondolkodjuk el azon, hogy milyen módon kapcsolódnak egymáshoz ezek az egyedek. Rajzoljuk meg a kapcsolatokat!
- Csak ezt követően kezdjük meg az attribútumok feltárását és egyedenkénti megrajzolását. Ezt a lépést azért célszerű utolsóként megtenni, mert az egyedeknek és kapcsolatoknak egyaránt lehetnek attribútumaik.

- Ha a tulajdonságok feltüntetése közben rájövünk, hogy egy egyed egyes egyed-előfordulásainak speciális tulajdonságai vannak, vagy pedig több, eltérő egyed-előfordulásai azonos attribútumokkal is rendelkeznek, akkor alkalmazzunk specializációt vagy általánosítást.
- Amikor elkészültünk, legyünk benne biztosak, de legalábbis tételezzük föl, hogy elkövettünk néhány hibát. Feltétlenül nézzük át a modellt, és javítsunk a szükséges helyeken.

3.5 GRAFIKUS SEGÉDALKALMAZÁSOK

Az ER-modellek készítése közben hamar rájövünk, hogy papírral és ceruzával a kezünkben is készíthetünk ilyen modelleket, azonban ilyenkor néhány tucat radírt is biztosan elfogyasztunk majd. Ráadásul a diagram elkészítése sok időt vesz igénybe, és sohasem lesz olyan szép, mintha számítógéppel készült volna. Ezt felismerve számos szoftvergyártó cég készít különböző, köztük ER-modellek rajzolására is alkalmas segédprogramokat. Sajnos ezek között elég kevés olyan van, amely az eredeti Chen-féle ER-diagram megrajzolására is alkalmas, de ismeri a kapcsolatok különböző jelöléseit is. A SmartDRAW 2012 nevű szoftver azonban közéjük tartozik. Számos különböző modell készítésére alkalmas, de sajnos csak kereskedelmi változatban szerezhető be.

http://www.smartdraw.com/

1. link A SmardDRAW honlapja

3.6 ÖSSZEFOGLALÁS, KÉRDÉSEK

3.6.1 Összefoglalás

Mai leckénkben az adatbázis-tervezés és koncepcionális modellezés rendkívül kifinomult, ugyanakkor egyszerűen használható eszközével, az Entity-Relathionship modellel ismerkedtünk meg. Megtanultunk, hogyan lehet a legmagasabb absztrakciós szinten, a logikai és fizikai modellektől függetlenül elképzelni, modellezni és ábrázolni egy adatbázist.

Megismertük az ER-modellben használható fogalmakat és azok ER-diagramon történő ábrázolását.

Megkülönböztettük az erős és gyenge egyedeket, megtanultuk, mit értünk a kulcs, az összetett, a származtatott és a többértékű attribútum fogalmak alatt. Megismerkedtünk a kapcsolatok különböző jelöléseivel, a számosság és a kötelezőség ábrázolásával. Olvashattunk a rekurzív kapcsolatokról, és példával illusztráltuk a kapcsolatokhoz kötődő attribútumok használatát.

Végül az ER bővített változata, az EER-modell kapcsán megismertük az objektum- és az adatorientált tervezés együttes alkalmazásának lehetőségeit, a specializációt és az általánosítást.

3.6.2 Önellenőrző kérdések

1. Mit értünk az alatt, hogy az ER-modell magas szintű modell?
   - Azt, hogy a modellben magas szintű absztrakciót alkalmazunk. Figyelmen kívül hagyjuk az adatok fizikai tárolásával, sőt a logikai szerkezettel kapcsolatos részleteket is, és kizárólag az egyedek, attribútumok és kapcsolatok rendszerére koncentrálunk.
2. Mi a különbség gyenge és erős egyed között?
   - A gyenge egyed nem rendelkezik az előfordulások egyedi azonosítására alkalmas kulccsal. Parciális kulcsa csak egy erős egyed azonosítójával együtt használható kulcsként.
3. Hogyan ábrázolható a kulcs, a többértékű, az összetett és a származtatott attribútum?
   - Minden attribútumot a saját egyedének téglalapjához kapcsolt ellipszissel ábrázolunk úgy, hogy az ellipszisben feltűntetjük a tulajdonság nevét. A kulcs nevét aláhúzással jelöljük. A többértékű attribútumot dupla, a származtatott tulajdonságot szaggatott vonalas ellipszis jelzi. Az összetett attribútumhoz hozzákapcsoljuk annak elemi attribútumait.
4. Mit jelent egy kapcsolatban a totális és a parciális részvétel?
   - Az egyed részvétele totális, ha nem lehet olyan előfordulása, amelyik nem vesz rész a kapcsolatban. Ha

előfordulhat, hogy legalább egy egyed-előfordulás nem kapcsolódik a másik egyedtípus egyetlen előfordulásához sem, akkor beszélünk parciális kapcsolatról.

5. Mit értünk rekurzív kapcsolat alatt?

- Azt a kapcsolatot, amikor egy egyed előfordulásai ugyanannak az egyednek az előfordulásaihoz kapcsolódnak.

4. LECKE: A RELÁCIÓS ADATMODELL

4.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

Az emberiség története során a barlangrajzoktól a rovásíráson, hieroglifákon, rajzolt térképeken, kézzel írott, majd nyomtatott könyveken keresztül egészen a számítógépes adattárolásig számos adatábrázolási forma alakult ki. Ezeket külön-külön megvizsgálva is óriási változatosságra bukkanunk. Ha csak a számítógép segítségével, alfanumerikus formában ábrázolható adatokra gondolunk, akkor is különbséget tehetünk

- az adatok strukturálatlan, szöveges tárolása
- a strukturált szövegek, és
- az adatszerűen tárolt adatok között.

Az egyszerű, minden tagolást, struktúrát nélkülöző folyó szöveg nagyon közel áll a szóbeli kommunikációhoz, azonban ahhoz hasonlóan meglehetősen terjengős, redundáns. Az ilyen szöveg szemantikai jelöléseket sem tartalmaz, ezért egyes részletei csak folyamatos olvasással, vagy szabad szavas kereséssel lelhetők fel. A releváns találatok száma ilyenkor általában csekély, emiatt ez az adattárolási forma nem alkalmas korunk információéhségének hatékony csillapítására.

A strukturált szövegből jóval hatékonyabban juthatunk információhoz. Az ilyen szöveg a legegyszerűbb esetekben címsorokkal tagolt témakörökből áll, de az igazán jól strukturált szöveges adatszerkezetekben – mint például az XML-állományokban – pontos azonosítást is lehetővé tévő jelölők biztosítják a szemantikai tagolást.

Bár az értelmi jelölést tartalmazó szövegek számítógéppel is feldolgozhatók, az igazán hatékony adatkezelést az adatok „adatszerű” tárolása biztosítja.

Az **adatszerű adatkezeléskor** az egyes **egyedekre** jellemző **tulajdonságokat** értjük adat alatt. Ezeket **pontosan meghatározható** és ábrázolható **adatstruktúrákban** tároljuk, és kifejezetten erre a célra fejlesztett **számítógépes szoftverek**, adatbázis-kezelő rendszerek segítségével kezeljük.

Az adatszerű tárolásban meghatározó szerepet játszanak az adatstruktúrák és a leírásukra alkalmas úgynevezett adatmodellek.

Előbbi leckénkben az adatmodellezés legmagasabb, koncepcionális szintjén használható ER-modellről tanultunk. A logikai modellekkel foglakozunk. Elsőként röviden áttekintjük a napjainkra kialakult hierarchikus, hálós, relációs és

objektumorientált modelleket, majd részletesen tárgyaljuk a közülük leginkább elterjedt **relációs adatmodellt**.

4.2 LOGIKAI ADATMODELLEK

A logikai adatmodellek a koncepcionális modellezésnél alacsonyabb absztrakciós szintet használnak, és a fizikai adatbázisokhoz közelebb álló fogalmak segítségével teszik lehetővé az adatbázisok modellezését. Nem csupán az egyedek tulajdonságainak és kapcsolatainak leírásával, hanem az ezek tárolására alkalmas adatstruktúrákkal is foglakoznak.

A logikai adatmodellek és a fizikai adatbázisok szoros kapcsolata abban is megnyilvánul, hogy a fizikai adatbázisok meghatározó jellemzője a szerkezetük alapját képező logikai adatmodell. Ezért is beszélünk hierarchikus, hálós, relációs illetve objektumorientált adatbázisokról.

A logikai adatmodellek elemei

Minden ilyen adatmodell két nagyobb egységre, **statikus** és **dinamikus** részre bontható. A statikus elemek az egyedek tulajdonságainak és kapcsolatainak tárolására használható **adatstruktúrát**, a dinamikus elemek pedig a modellnek megfelelően tárolt adatokkal végezhető **műveleteket** írják le. A napjainkban legelterjedtebb relációs adatmodellt egy további, úgynevezett **integritási** rész is kiegészíti. Ez a tárolt adatok helyességét meghatározó alkalmas szabályokat tartalmazza.

A logikai adatmodellek megértéséhez elsősorban a strukturális elemek ismerete szükséges, ezért a hierarchikus, hálós, és objektumorientált modelleket ezeken keresztül mutatjuk be. A relációs modellel bővebben foglakozunk, és a struktúrák bemutatásán túl a következő leckék könnyebb értelmezéséhez szükséges integritási szabályokat is ismertetjük.

Az adatmodellezés történetének furcsasága, hogy a logikai adatmodellek többsége a magasabb szintű modellezésre alkalmas koncepcionális modellek kialakulása előtt jött létre. Az adatbázismodellezésben kezdetben nem foglalkoztak az olyan magas szintű absztrakciós lehetőségekkel, mint amilyeneket például az ER-modell biztosít. Talán ez az oka annak, hogy számos adatmodellezésről szóló tankönyv előbb a logikai modelleket ismerteti, és csak azok után mutatja be a koncepcionális modellezés lehetőségeit.

Az is igaz, hogy a magas szintű modellezés szépségei valójában akkor válnak igazán értékelhetővé, ha már megismertünk legalább egy ala-

csonyabb szintű adatmodellt. Tananyagunkban ennek ellenére a modellezés módszertanának megfelelő sorrendet követjük.

4.2.1 A hierarchikus adatmodell

Az 1960-as végén kialakult első logikai adatmodell a hierarchikus adatmodell volt. Alapvető szerkezeti eleme a **rekord**, amely **egy egyed tulajdonságainak** tárolására alkalmas. Minden **rekord jellemzője** az egyed tulajdonságainak tárolására alkalmas adatszerkezet, amit **rekordtípus** ír le. Bár a modell nem használja ezt a fogalmat, az azonos rekordtípusú rekordok alkotják az egyedtípusokat.

A rekordok fa struktúra alapján, hierarchikusan kapcsolódnak egymáshoz. A hierarchia a rekordok közötti **szülő-gyermek** (**PCR,** parent-child-relation) kapcsolatban nyilvánul meg. Egy szülőnek több gyermeke is lehet, azonban egy gyermek csak egy szülőhöz kapcsolódhat. Ez azt jelenti, hogy a struktúrát alapvetően **1:n kapcsolatok jellemzik**.

A hierarchikus adatmodellt később továbbfejlesztették, így alkalmassá vált a több-több kapcsolatok leírására is. Az itt olvasható jellemzők a modell eredeti változatára vonatkoznak.

Az eredeti hierarchikus adatmodell az 1:1 számosságú kapcsolatokat nem támogatja megszorításokkal, az n:m kapcsolatok létrehozását pedig nem is teszi lehetővé.

A modell kezdeti népszerűségét részben monopolhelyzetének köszönheti, részben pedig annak, hogy az akkor elterjedt szalagos háttértárakkal is lehetővé tette a hierarchikus adatbázisok tárolását és kezelését. Előnyei között felsorolható a kezelő szoftverek egyszerűsége, és a kapcsolódó egyedek gyors feltárása, a modellt mégis fokozatosan háttérbe szorították a később megjelenő riválisadatszerkezetek.

4.2.2 Hálós adatmodell

1971-ben hozták létre és publikálták a hierarchikus modell kiterjesztéseként is felfogható hálós adatmodellt.

A struktúrájának alapelemei ez esetben is a különböző **rekordtípusokba** tartozó **rekordok**, amelyeket ebben a modellben is egy **gráf köt össze**. Míg a hierarchikus modell speciális gráfot, azaz fát használ, addig a hálós modell nem alkalmaz ilyen megkötést.

A rekordok **halmazokat**, úgynevezett **set-**eket alkotnak. Minden sethez egy **tulajdonos** és több **tag** rekord tartozik. A tagok azonos, a tulajdonos pedig egy másik rekordtípus képviselői. A **tagok** bármelyike egyben **egy másik set tulajdonosa is** lehet.

A set rekordjait **mutatók**, úgynevezett pointerek kötik össze, biztosítva ezzel a **kapcsolatok** leírását. Minden rekordban megtalálható a set tulajdonosára, illetve a set-en belüli előző és következő tagrekordra mutató pointer.

A hálós modell szerkezetéből adódóan közvetlenül az 1:n kapcsolatokat támogatja, ugyanakkor megengedi, hogy egy **rekord egyszerre több set tagja is lehessen**. Ezt felhasználva kapcsoló set-ek közbeiktatásával az n:m kapcsolat is modellezhető. A modell további érdekessége, hogy **megengedi a többértékű, és az összetett attribútumok** használatát is.

A hálós adatmodell kiváló lehetőségeket biztosít az adatok rugalmas kezelésére, azonban az adatszerkezet nehezen átlátható, és a kezelőprogramok elkészítése is bonyolult.

4.2.3 Relációs adatmodell

A relációs modell **napjaink legelterjedtebb logikai adatmodellje,** így különálló szakaszban foglalkozunk vele. Ebben a felsorolásban csak annyit hadd említsünk meg, hogy a modell úgynevezett **relációkban tárolja** az azonos típusú egyedeket. Sikerei minden bizonnyal éppen ennek köszönhetők. A relációk ugyanis – kevéssé informatív nevük ellenére – roppant egyszerű adatszerkezetek, gyakorlatilag sorokból és oszlopokból álló mátrixok, azaz **táblázatok**.

4.2.4 Objektumorientált adatmodell

Az objektumorientált modell a relációs adatmodell továbbgondolásával, az objektum- és adatorientált szemlélet elveit egyesíti. Az adattárolás, egyedeknek megfelelő strukturális egysége az **objektum**, amely az egyedtípussal analóg **osztály** egy példánya. Az osztály-objektum viszony tehát hasonlít az egyedtípus és az egyed kapcsolatához.

Az **osztályok** az adatok tárolására alkalmas **tulajdonságokat**, változókat, és a tárolt adatok feldolgozására alkalmas eljárásokat, **műveleteket** írnak le. Az adatok és a feldolgozó műveletek közös struktúrában történő tárolását nevezzük **egységbe zárásnak**.

Minden objektum a saját osztályában definiált tulajdonságok egyedi értékeit hordozza, amelyeken elvégezhetők az osztályban leírt összes műveletek.

Az objektumorientált adatmodellben az egységbe zártság mellett további jellemzők az **öröklődés,** és az ezzel összefüggő a **polimorfizmus**.

Az öröklődés azt jelenti, hogy szülőosztályok alapján alosztályokat tudunk létrehozni. Az alosztály örökli a szülő minden tulajdonságát és műveletét, de az örökölt elemek kiegészíthetők, sőt meg is változtathatók az alosztályban. A polimorfizmus erre az utóbbi lehetőségre utal. Talán sejthető, hogy az objektumorientált adatmodellben ez a két jellemző teszi lehetővé a specializációt és az általánosítást.

4.3 A RELÁCIÓS ADATMODELL

Napjaink legnépszerűbb, egyben legelterjedtebb adatmodellje a relációs adatmodell. Sikerét elsősorban roppant egyszerű és átlátható strukturális elemeinek, és a relációs elven működő adatbázis-kezelő rendszerek – ebből következően – nagy számának köszönheti.

Az adatmodell megalkotója **Edgar Frank Codd** 1970-ben publikálta a modellre vonatkozó elképzeléseit. Célja egy **matematikailag** jól meghatározott eszközöket és **fogalmakat használó**, ugyanakkor **könnyen ábrázolható**, **egyszerű adatkezelést**, és **gyors feldolgozást** biztosító modell létrehozása volt.

Codd ez időben az IBM munkatársaként dolgozott. A hardver és szoftver gyártásáról is ismert informatikai óriás a System R nevű projekt keretében elkészítette a relációs modell alapján működő első adatbázis-kezelő szoftvert is. A ’79-ben debütáló adatbázis-kezelő rendszer nagy sikereket ért el, így rövidesen számos nagynevű szoftvergyártó fogott hozzá saját relációs rendszere fejlesztéséhez. A relációs adatmodell ezzel megkezdte napjainkig tartó hódítását az adatbázis-rendszerek piacán.

4.3.1 A modell strukturális elemei

A relációs adatmodell különböző strukturális elemeket használ az egyedtípusokba tartozó egyedek tulajdonságtípusokba sorolt tulajdonságainak és kapcsolatának modellezésére. A modell szerkezeti elemei az alábbiak:

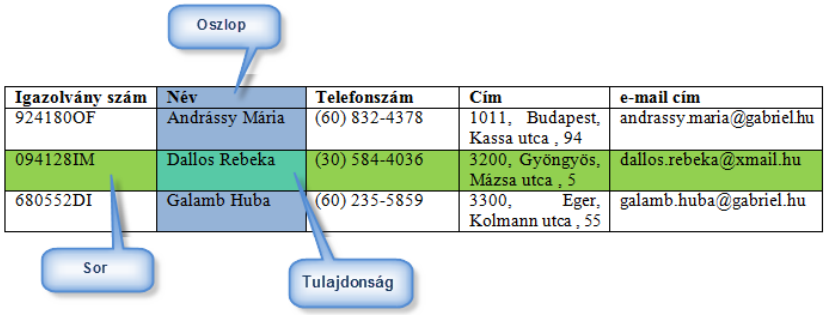
Tábla, reláció

Az azonos egyedtípusba tartozó egyedek tárolása egyszerűen áttekinthető adatszerkezetekben, úgynevezett **táblákban**, más néven **relációkban** történik.

A tábla egyes oszlopai az egyedtípus tulajdonságtípusainak, a sorok pedig az egyedeknek felelnek meg. A táblában annyi tulajdonság tárolódik, amennyi a

sorok és oszlopok szorzata, hiszen minden egyed sorában tároljuk az összes tulajdonságtípus oszlopához tartozó tulajdonságot.

A matematikában két halmaz Descartes-féle szorzatát **relációnak** nevezik. Egy táblában annyi tulajdonság tárolódik, amennyi a sorok és oszlopok halmazának Descartes-szorzata, ezért mondhatjuk, hogy a **tábla sorok és oszlopok relációja**. A modell neve innen származik.



Igazolvány szám	Név	Telefonszám	Cím	e-mail cím
924180OF	Andrássy Mária	(60) 832-4378	1011, Budapest, Kassa utca , 94	andrassy.maria@gabriel.hu
094128IM	Dallos Rebeka	(30) 584-4036	3200, Gyöngyös, Mázsa utca , 5	dallos.rebeka@xmail.hu
680552DI	Galamb Huba	(60) 235-5859	3300, Eger, Kolmann utca , 55	galamb.huba@gabriel.hu

21. ábra A tábla szerkezete

Domain

A domain egy tulajdonság lehetséges értékeinek halmazát, azaz értékkészletét, és a tárolt értékkel végezhető műveleteket határozza meg.

A fenti ábra ***név*** *oszlopában lévő tulajdonságok közös domainbe tartoznak, hiszen mind szövegek. Az* ***Igazolvány szám*** *oszlop tulajdonságai másik domain elemei. Speciális szövegek, amelyek 8 karakterből állnak, első hat karakterük arab számjegy, az utolsó kettő pedig betű.*

Rekord

A **rekord** a tábla egy sora, a szó tehát az **egyed szinonimája**. Minden rekord egy egyed tulajdonságait tárolja. A táblákban tetszőleges számú rekord helyezhető el, amelyek **sorrendje nem kötött.**

Szabály azonban, hogy a táblán belül nem lehet két teljesen azonos mezőértékeket tároló rekord**, azaz egy rekord sem szerepelhet többször.**

Mező

A modellben a **mező** megnevezést használjuk a **tulajdonságtípus** szinonimájaként. A mező a tábla egy **oszlopa**. A mezők oszlopaiban helyezzük el az egyes rekordokra jellemző tulajdonságokat. A mezők száma szintén nem korlátozott.

> **Az adatszerkezet igen fontos szabálya azonban, hogy egy táblán belül minden rekordban azonosnak kell lennie mezők számának! Nem lehet olyan rekord, amelyikben több vagy kevesebb mező van, mint a többi rekordban.**

Mezőérték

A **mezőértékek** a tárolt **tulajdonságok**.

> **Az adatszerkezet előírja, hogy egy mező összes mezőértékének azonos domainhez kell tartoznia.**

Egy későbbi kivételtől eltekintve a mezők lehetnek üresek. Az **üres** mezők az adatbázis-kezelés egy speciális értékét, **NULL** értéket tartalmaznak. A NULL a semmit jelenti. Nem azonos a nullával, vagy a két egymás mellé írt idézőjellel „”. Az előbbi egy konkrét számot, utóbbi üres szöveget jelent. A NULL jelentését leginkább a következő egyenlőségek mutatják:

```
0    + 100  = 100
NULL + 100  = NULL
```

A modell kapcsán említett **tábla** és **reláció** szinonim fogalmak, mindkettő az **egyedtípus** tárolására használt struktúrát jelöli.

A **mezőket** szokás **attribútumoknak,** vagy **oszlopnak** nevezni, a **rekord** helyett a **sor**, az angol nyelvű irodalomban pedig a **tuple** szót használni.

Mint láttuk, a modell egyedtípusok modellezésre alkalmas eleme roppant egyszerű felépítésű. Ez csak úgy érhető el, hogy egyes **mezők** nem pusztán adathordozó, hanem az **adatbázis szerkezete szempontjából** kiemelkedő **funkcionális szerepet is játszanak**.

Kulcs

Mivel egy az előírásoknak megfelelő **táblában nem lehet két egyforma rekord,** biztos, hogy létezik a mezőknek olyan részhalmaza, amelyek mezőértékeit kombinálva **egyedi értéket** kapunk. Az érték alapján teljes biztonsággal, egyértelműen megkülönböztethetjük a tábla rekordjait. Az ilyen mezőkombinációt nevezzük **szuperkulcsnak**.

Szélsőséges esetben a tábla összes mezője együtt alkotja a szuperkulcsot, de az esetek többségében rövidebb szuperkulcs is létezik.

A legrövidebb (legkevesebb mezőből álló) szuperkulcsot **kulcsjelöltnek** hívjuk. Ha több kulcsjelölt is van, akkor ezeket **alternáló kulcsoknak** tituláljuk.

A táblákban elvégzett műveletek esetén fontos, hogy az **egyes rekordokra félreérthetetlenül tudjunk hivatkozni**. Erre egy kiválasztott kulcsjelölt értékeit használhatjuk. Hogy a hivatkozások egyértelműek legyenek, a tábla tervében előre ki kell jelölni az azonosításra szánt kulcsjelöltet. A rekordok azonosítására kiválasztott kulcsjelöltet **elsődleges kulcsnak**, vagy **azonosítónak** nevezzük.

Ha az azonosítót egy mező alkotja, akkor **egyszerű**, ha több, **akkor összetett kulcsról** beszélünk.

A gyakorlatban ritkán használjuk azonosítóként egy egyedtípus tényleges tulajdonságait. A leggyakoribb technika az, hogy minden táblába beszúrunk egy új, számok tárolására alkalmas mezőt, és ezt használjuk azonosítóként. A mezőben úgy biztosítjuk az egyedi értékeket, hogy minden rekordot saját, egyedi sorszámmal látunk el.

A legtöbb adatbázis-kezelő rendszer képes automatikusan elvégezni az ilyen mezők sorszámozását.

Idegen kulcs

Az adatmodell egyes táblái az egyedtípusok közötti viszonyoknak megfelelően kapcsolódnak egymáshoz. Két tábla kapcsolata esetén az egyik táblát **elsődleges**, a másikat **kapcsolódó táblának** hívjuk.

A relációs adatmodellben a kapcsolódó táblában elhelyezett speciális funkciójú mezőkkel, az úgynevezett **idegen kulcsokkal** írjuk le a táblák kapcsolatát. Az **idegen kulcs** egy kapcsolódó tábla olyan mezője, amely az **elsődleges tábla kulcsának értékeit** tárolja, így teljes biztonsággal megmutatja, hogy egy rekord a másik tábla melyik rekordjához kapcsolódik.

A relációs adatmodellben két közvetlenül kapcsolódó tábla csak 1:1, vagy 1:n típusú kapcsolatban lehet.

Az n:m kapcsolat esetén mindenképpen többértékű lenne az idegen kulcs, a relációs adatmodell azonban csak atomi értékek tárolását teszi lehetővé, a többértékű mezők használata tilos. A következő leckében látni fogjuk, hogy egy úgynevezett kapcsoló tábla közbeiktatásával mégis van lehetőség az n:m kapcsolatok tárolására.

Származtatott táblák

A relációs adatmodell tényleges tábláit **bázisrelációknak** nevezzük. Az adatbázisban különböző műveleteket végzünk a bázisrelációkkal. Segítségükkel előállíthatjuk például a bázistábla mezőinek és rekordjainak részhalmazát. A létrehozott eredmény is tábla szerkezetű, de sorai és oszlopai bázistáblából származnak, ezért **származtatott relációnak** nevezzük.

A származtatott relációk mezői, rekordjai és mezőértékei csak a bázisrelációban tárolódnak. A származtatott reláció használatakor az adatbázis-kezelő rendszer mindig újra és újra kiválogatja a bázistábla megfelelő mezőit és rekordjait, így a kettős adattárolást elkerülve érhetjük el, hogy ugyanabból a táblából különféle adatokat jeleníthessünk meg.

Például meg akarjuk nézni a helyben lakó dolgozók nevét és telefonszámát. Ilyenkor a dolgozói adatokat tároló táblából kiválogatjuk azokat, akik a céggel azonos településen laknak, a kiválasztottak tulajdonságai közül pedig csak a nevet és telefonszámot jelenítjük meg.

Azonosító	SzemIgSzam	Név	Telefon	Város	Utca	IRSZ	Mail
1	924180OF	Andrássy Mária	(60) 832-4378	Budapest	Kassa utca , 94	1011	andrassy.maria@gabriel.hu
3	680552DI	Galamb Huba	(60) 235-5859	Eger	Kolmann utca , 55	3300	galamb.huba@gabriel.hu
4	277976ZE	Károly Bertalan	(30) 891-6709	Eger	Cseresznyés utca , 74	3300	karoly.bertalan@epost.hu
6	450724RW	Károly Jácint	(70) 264-7074	Eger	Szalag utca , 12	3300	karoly.jacint@xmail.hu
10	167688MY	Liktor Bertalan	(20) 304-2231	Eger	Platán utca , 17	3300	liktor.bertalan@xmail.hu
11	995369TN	Lénárd Domokos	(60) 638-9383	Eger	Kard u		mokos@epost.hu
14	765531HB	Kállai Gáspár	(20) 471-5961	Eger	Latabá		par@gabriel.hu
17	886071GH	Moka Ibolya	(70) 542-6072	Eger	Révés		lya@gabriel.hu
19	069829XZ	Kovács Lehel	(70) 909-4964	Eger	Thorm		hel@quickpost.com
2	094128IM	Dallos Rebeka	(30) 584-4036	Gyöngyös	Mázsa		beka@xmail.hu
8	984830UG	Dévényi Jácint	(20) 696-4781	Kazincbarcika	Henge		jacint@quickpost.com
5	863390VA	Keller Berta	(20) 942-4648	Miskolc	Epresk		rta@answer.com
7		Pomogács Egyed	(60) 968-2916	Miskolc	Szilvás		s.egyed@gabriel.hu
9		Alapi Domokos	(30) 583-3755	Miskolc	Kovács		okos@xmail.hu
13		Pintye Lajos	(20) 737-7775	Miskolc	Szilvás kö		pintye.lajos@quickpost.com
12	582783PJ	Szél Domokos	(70) 525-8777	Ozd	Kovács P		szel.domokos@xmail.hu
16	290647KL	Dirkó Rebeka	(60) 634-8517	Ozd	Bajnár utca , 97	3600	dirko.rebeka@epost.hu
15	361965CW	Szél Erika	(60) 756-3432	Pétervására	Venyige utca , 10	3250	szel.erika@gabriel.hu
18	473495CD	Pelle Boglárka	(30) 695-5918	Pétervására	Dékáni A. utca , 97	3250	pelle.boglarka@answer.com

Név	Telefon
Galamb Huba	(60) 235-5859
Károly Bertalan	(30) 891-6709
Károly Jácint	(70) 264-7074
Liktor Bertalan	(20) 304-2231
Lénárd Domokos	(60) 638-9383
Kállai Gáspár	(20) 471-5961
Moka Ibolya	(70) 542-6072
Kovács Lehel	(70) 909-4964

Bázis reláció

Származtatott reláció

22. ábra Bázis- és származtatott reláció

A relációs adatmodell kétféle származtatott táblát, a snapshotot (pillanatfelvételt) és a view-t (nézetet) különbözteti meg.

Snapshot

A művelet eredményeként létrehozott származtatott relációt **eredményreláció**nak is nevezzük. Az eredményrelációkban megjelenő adatokat általában **nem megváltoztatni, csak megjeleníteni** szeretnénk.

> **A snapshot olyan származtatott reláció, amelynek eredményrelációjában megjelenő mezőértékek nem változtathatók meg, és nem tárolódnak újra az adatbázisban. A pillanatfelvételek csak olvashatók.**

View

Előfordulhat, hogy eredményrelációt rendszeresen meg akarunk jeleníteni, és a bázisrelációval hasonló módon akarunk kezelni Nemcsak megtekinteni, hanem szükség esetén változtatni az akarunk az adatokon. Ilyenkor lehetnek hasznosak a **view**-k, más szóval **nézetek**.

Például lehetővé akarjuk tenni, hogy az egyes osztályvezetők csak saját beosztottaik rekordjait lássák, de azokon bármilyen műveletet elvégezhessenek.

> **A nézet olyan származtatott reláció, amely a felhasználók előtt bázis relációként jelenik meg. Eredményrelációja olvasható és írható, a változtatások aktualizálódnak az érintett bázistáblában. A nézeteket más néven virtuális tábláknak is hívjuk.**

Séma

A séma egy leendő relációs adatbázis teljes logikai modellje. Tartalmazza a táblák, a kapcsolatok és a nézetek modelljét.

4.3.2 Integritási szabályok

A relációs adatmodell fontos részét képezik az úgynevezett **integritási szabályok**. Ezek a szabályok biztosítják, hogy az adatbázis működés közben is mindig konzisztens maradjon. Azaz a felhasználó által elvégzett adatváltoztatások

közben se kerüljön olyan állapotba, amikor a táblákban egymásnak ellentmondó hibás adatok vannak.

Az **integritási szabályok az adatbázisban tárolható értékekre**, illetve azok helyességére **vonatkozó előírásokat** tartalmaznak.

A legfontosabb ilyen előírások az **egyedintegritás** és a **hivatkozási integritás** szabálya.

Az egyedintegritás szabálya

Az **egyedintegritás** szabálya azt mondja ki, hogy egy tábla **elsődleges kulcsának** egyedi értéket kell biztosítania a táblán belül, valamint hogy az azonosító egyik eleme **sem tartalmazhat NULL** értéket. Erre azért van feltétlenül szükség, mert ha a kulcsban megengednénk a NULL érték tárolását, akkor előfordulhatna, hogy két rekordban is üres lenne az azonosító, amely így elveszítené eredeti funkcióját.

Hivatkozási integritás szabálya

A **hivatkozási integritás** szabálya a kapcsolatokat leíró idegen kulcsokra vonatkozik. Kimondja, hogy az **idegen kulcs értéke csak akkor helyes**, ha a **hivatkozott** (elsődleges) tábla **elsődleges kulcsának egy létező értékét** tárolja, **vagypedig** üres, azaz **NULL** értékű.

A szabály betartásával elérhető, hogy egy tábla rekordja egy másik táblának csak létező rekordjára hivatkozhat. Az idegen kulcsban akkor használunk NULL értéket, ha az egyedtípus parciálisan vesz részt a kapcsolatban.

4.4 A RELÁCIÓS MODELL REPREZENTÁCIÓI

A modellek fontos részét képezik a különböző ábrázolási formák. A relációs adatmodell tábláinak, azok mezőinek és kapcsolatainak ábrázolásakor többféle formalizmust alkalmaznak.

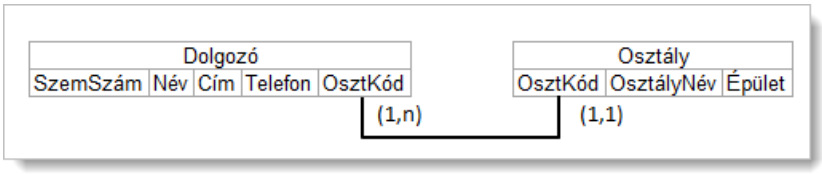
4.4.1 Szöveges leírás

A legegyszerűbb technika, amikor a táblanevek után kerek zárójelek között sorolják fel a mezőneveket, és aláhúzással jelölik az azonosítót. A kapcsolatokat vagy az elsődleges és a kapcsolódó idegen kulcs azonos nevével, vagy nyilakkal jelzik.

```
Dolgozó(SzemSzam, Név, Cím, Telefon, OsztKod)
Osztály(OsztKod, Osztálynév, Épület)
```

4.4.2 Bachman-diagram

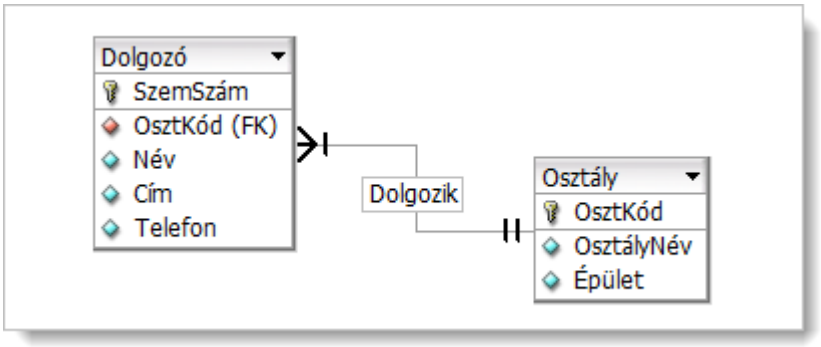
A Bachman-diagram téglalapokat használ a táblák és mezők megnevezésére. A kapcsolatokat vonalak mutatják, amelyek végén az ER-modellben tanult jelölések mutatják a számosságot, illetve a kötelezőséget.



23. ábra Bachman-diagram

4.4.3 Grafikus jelölés

A modell leginformatívabb ábrázolása szintén téglalapokat használ a táblák jelölésére, de a mezőnevek mellet azok funkciójára utaló jelölést is használ. A kapcsolatok jelölése itt is az ER-diagramnál látott eszközökkel történhet.



24. ábra Grafikus jelölés

4.5 ÖSSZEFOGLALÁS, KÉRDÉSEK

4.5.1 Összefoglalás

Mai leckénkben napjaink legnépszerűbb logikai adatmodelljével, a relációs adatmodellel ismerkedtünk meg. Megtudtuk, hogy az adatmodellek az adatstruktúrát leíró statikus, illetve az adatokon végezhető műveleteket meghatározó dinamikus részből épülnek fel. A modellezés szempontjából elsősorban a statikus rész jelentősége nagy, ezért megtanultuk a négy logikai adatmodell strukturális elemeit. A relációs adatmodell esetén részletesebben beszéltünk a szerkezeti elemekről, és a modellt kiegészítő integritási szabályokat is megismertük.

A relációs modell szerkezeti elemei között első helyen említettük a mezőkből, rekordokból és mezőértékekből álló relációkat vagy más néven táblákat, amelyek az egyedtípusok modelljei.

Különbséget tettünk az adatbázisban ténylegesen tárolódó bázis-, illetve a rekordokon végrehajtott műveletek révén előállított származtatott relációk között. A származatott relációk kapcsán megemlítettük a csak olvasható pillanatfelvételeket, és a virtuális táblaként használható, olvasható és írható nézeteket. A szerkezeti elemek között tanultunk az azonosítókról, és az olyan kapcsolódó fogalmakról, mint a szuperkulcs, kulcsjelölt, elsődleges kulcs, és idegen kulcs.

A lecke végén az adatbázisban tárolt adatok helyességét biztosító integritási szabályokról olvashattunk. Két legfontosabb előírásként emeltük ki az egyed és a hivatkozási integritás szabályait. Megtanultuk, hogy előbbiek a rekordok minden körülmények között megmaradó azonosíthatóságát, utóbbiak pedig az idegen kulcsok helyességét biztosítják.

4.5.2 Önellenőrző kérdések

6. Milyen részekből tevődik össze egy logikai modell?
    - A struktúrát leíró statikus, valamint az elvégezhető műveleteket meghatározó dinamikus részből. A relációs adatmodell fontos elemeit képezik az integritási szabályok is.
7. Minek köszönheti népszerűségét a relációs adatmodell?

- Elsősorban az egyedtípusok tárolására használt adatstruktúra egyszerűségének és könnyű áttekinthetőségének.

8. Mondja el, mit jelent az, hogy a mezőkben csak atomi értékek tárolódhatnak?
    - Azt, hogy a mezők nem tartalmazhatnak sem összetett, sem több értéket.
9. Mit tud az egyedintegritás szabályáról?
    - Kimondja, hogy az elsődleges kulcsban egyedi értéket kell tárolni, és hogy a kulcs nem lehet NULL értékű.
10. Mi a különbség bázis- és származtatott reláció között?
    - A bázisreláció tárolódik az adatbázisban, a származtatott reláció eredményrelációja nem.

5. LECKE: A REDUNDANCIA KÖVETKEZMÉNYEI ÉS CSÖKKENTÉSE

5.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

Az eddigi leckék tananyagából megtanulhattuk, hogy az adatbázisokat különböző absztrakciós szinteken modellezhetjük. A koncepcionális, logikai és fizikai modellezés különválasztásával, az absztrakció fokozatos csökkentésével a teljes rendszer bonyolultságát eltakarva mindig csak az adott szinten fontos elemekre koncentrálhatunk. A koncepcionális szinten az egyedek, tulajdonságok és kapcsolatok rendszerét a tárolhatósággal nem törődve alakítjuk ki, de a logikai szinten már az adatstruktúrákra is ügyeltünk.

Előző leckénkben alaposan áttanulmányoztuk napjaink legnépszerűbb logikai modelljét. Mivel ígéretünk szerint a tananyag további részében már csak ezzel a logikai modellel foglalkozunk, az olvasó joggal gondolhatja, hogy most következő leckénkben már a modellezés fizikai szintjén fogunk dolgozni.

Mielőtt azonban erre is sort kerítenénk, kisebb kitérőt téve mindenképpen beszélnünk kell a relációs logikai adatmodell szabályai szerint megtervezett adatbázisokban előforduló hatékonysági problémákról és azok megoldásairól.

A most következő leckében a relációs adatbázisok hibás tervezését jelző redundanciáról, ennek következményeiről, és megszűntetésének lehetőségeiről olvashat. Tananyagunk bemutatja a normalizálás folyamatát, majd ismerteti azokat a módszereket, amelyekkel a normalizálás lépéseit részben megkerülve, az ER-modell leképezésével a juthatunk helyes szerkezetű, hatékonyan működő relációs adatbázisokhoz.

5.2 ROSSZUL MODELLEZETT ADATBÁZIS

A relációs adatmodell pontosan leírja a felhasználható strukturális elemeket, és integritási szabályokkal biztosítja, hogy a modell alapján készülő adatbázisok mindig konzisztensek maradjanak. Rugalmas lehetőségeinek köszönhetően azonban ugyanaz az adathalmaz többféleképpen is modellezhető. Ez természetesen nem jelentene problémát, ha az egyes változatok ugyanolyan biztonságosak és hatékonyak lennének. Rövidesen látni fogjuk azonban, hogy az eddig szerzett ismereteink alapján helyesen modellezett relációs adatbázisokkal is előfordulhatnak problémák. Méretük indokolatlanul megnövekedhet, előfordulhat, hogy az egyes műveletek csak sok munka árán végezhetők el, esetleg

inkonzisztens állapotba hozzák az adatbázist. Akár az is megeshet, hogy bizonyos adatkezelő műveltek egyáltalán nem lesznek elvégezhetők.

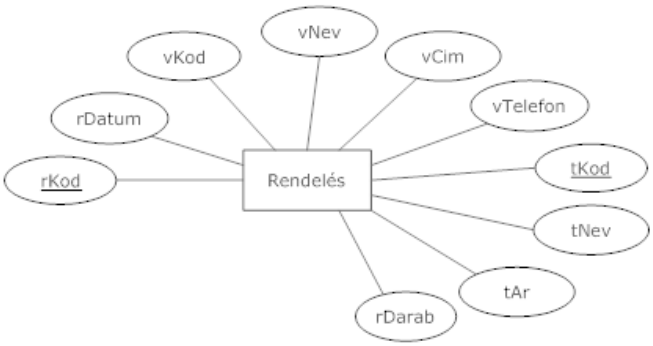
A hibák ilyenkor a rosszul megtervezett adatbázisok sajátosságára, az adatok többszörös tárolására, az úgynevezett **redundanciára** vezethetők vissza. A redundancia olyan többszörös adattárolását jelent, ahol az ismételten tárolt adatok értelmezésével már nem nyerhető új információ.

> **Azt az ismétlődő adatot, amelynek értelmezésével már nem jutunk új információhoz, redundáns adatnak nevezzük. Az adatok fölösleges ismétlését redundanciának hívjuk.**

A leckében egy **informatikai termékeket árusító cég adatbázisának** tervezését, és az eközben jelentkező problémákat mutatjuk be.

> Szeretnénk előre hangsúlyozni, hogy a szándékunk nem egy valós adatbázis-kezelési feladat bemutatása. Példaadatbázisunkkal csupán a redundancia felszámolásához kapcsolódó ismereteket kívánjuk illusztrálni. Az adatbázis ennek megfelelően erősen redukált, valós adatkezelési problémák megoldására csak jelentős kiegészítés után lenne alkalmas.

Példánkban cégünk vevőinek különböző hardvertermékekre leadott megrendeléseit tartjuk nyilván. A rendeléseknek tartalmazniuk kell a rendelés – azonosítóként használt – kódját, a dátumát, a megrendelő vevőkódját, nevét, lakcímét, telefonszámát, a rendelt termék kódját, darabszámát, nevét, és darabonkénti árát. Az adatbázist úgy kell elkészíteni, hogy egy rendelés több terméket is tartalmazhasson.



25. ábra Az adatbázis egyetlen egyedtípust tartalmazó ER-modellje

ER-diagrammal a fenti módon ábrázolhatnánk az adatok tárolására használható **egyetlen egyedtípussal kialakított** adatbázis szerkezetét. Az ennek megfelelő relációs modell reprezentációja is egyszerű:

```
RENDELÉS(rKod, rDatum, vKod, VNev, vCim, vTelefon,
   tKod, rDarab,tNev, tAr)
```

A következő ábrán a fenti, szándékosan elhibázott modelleknek megfelelően elkészített táblát és az abban tárolt néhány rendelés adatait látjuk.

rKod	rDatum	vKod	vNev	vCim	vTelefon	rDarab	tKod	tNev	tAr
R123	2010.06.07.	V19	Kovács Lehel	3300 Eger Thorma János utca , 13	(70) 909-4964	1	T224	Samsung SCX-4216F nyomtató-másoló-scanner+FAX	87 989
R123	2010.06.07.	V19	Kovács Lehel	3300 Eger Thorma János utca , 13	(70) 909-4964	3	T232	HP LaserJet 1015	47 366
R123	2010.06.07.	V19	Kovács Lehel	3300 Eger Thorma János utca , 13	(70) 909-4964	2	T237	Chicony KB9810 Win98 HUN PS/2	1 034
R124	2010.04.14.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T151	160 GB Freecom FHD-3 7200rpm USB2+FWire	48 290
R125	2010.12.09.	V05	Keller Berta	3501 Miskolc Epreskert utca , 33	(20) 942-4648	1	T215	DVD+-RW MSI DR16-B2 dob. Double Layer	12 639
R125	2010.12.09.	V05	Keller Berta	3501 Miskolc Epreskert utca , 33	(20) 942-4648	1	T238	AMD Athlon 64+FAN 3200+ dob. 939 pin	44 990
R126	2011.03.05.	V29	Keller Berta	3300 Eger Kolmann utca , 47	(70) 626-6488	1	T233	Chicony 790C-BS Mult. fekete-ezüst HUN PS/2	1 089
R126	2011.03.05.	V29	Keller Berta	3300 Eger Kolmann utca , 47	(70) 626-6488	1	T196	17" LG L1720P TFT 3 év DVI, 16ms	65 890
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T238	AMD Athlon 64+FAN 3200+ dob. 939 pin	44 990
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T196	17" LG L1720P TFT 3 év DVI, 16ms	65 890
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	4	T152	250 GB Freecom FHD-3 7200rpm USB2	51 700
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T239	Celeron 1800A+FAN S478	11 649
R128	2011.01.05.	V16	Dirkó Rebeka	3600 Ózd Bajnár utca , 97	(60) 634-8517	2	T182	Leadtek A180 DDR 64MB AGP8X T	6 270
R128	2011.01.05.	V16	Dirkó Rebeka	3600 Ózd Bajnár utca , 97	(60) 634-8517	1	T221	DVD+-RW Plextor PX-716A DL OEM	24 090

26. ábra Rendeléseket tároló tábla

A tábla mindenben megfelel a relációs adatmodell előírásainak. Első pillantásra is nyilvánvaló, hogy minden sor egy rendelés rekordja, és minden oszlop a rendelések egy tulajdonságtípusa, mezője. A mezőértékek azonos domainbe tartoznak, a rekordok különböznek egymástól.

> **Emeljük ki, hogy a tábla elsődleges kulcsa összetett, az `rKod`, és a `tKod` mezők kombinációjából áll! Ennek rövidesen komoly jelentősége lesz.**

Az ábrán azonnal látszik a vevők adatainak redundanciája, de kis böngészés után a termékek között is találunk ismétlődést. Az, hogy a **`V19`** kódú vevőt **`Kovács Lehelnek`** hívják, már az első rekordban kiderül. **`Kovács Lehel`** adatainak ismételt tárolásával tehát redundanciát okoztunk. Hasonló a helyzet a **`T238`**-as kódú termékkel, amelyről már az **`R125`**-ös rendelésben kiderül, hogy **`AMD Athlon 64 processzorról`** van szó.

5.3 ANOMÁLIÁK

A példánkhoz hasonlóan rosszul megtervezett adatbázisokban különböző problémákat okozhat a redundancia. Az első, azonnal szembeötlő hiba a fölöslegesen **megnövelt méret**. Természetes, hogy ha bizonyos tulajdonságokat

többször is tárolunk, akkor ezzel megnövekszik az adatbázis tárigénye. Ez a költségek megnövekedésén túl hatékonysági problémákat is okoz.

Az úgynevezett **anomáliák** ennél is súlyosabb gondokat jelentenek. Az adatbázis rekordjaival elvégezhető legalapvetőbb műveleteket: egy tábla új rekorddal való bővítését, a meglévő rekordok módosítását, illetve a fölöslegessé vált egyedek törlését nehezítik meg.

5.3.1 Bővítési anomália

A bővítési anomáliával akkor találkozhatunk, amikor a rosszul megtervezett adatbázist egy új egyeddel szeretnénk bővíteni.

A rendeléseket tároló adatbázisunkat például csak akkor bővíthetjük új termékkel (kód, név és ár), ha az árucikket azonnal hozzákapcsoljuk egy rendeléshez is.

Erre azért van szükség, mert a táblában lévő összetett kulcs egyik eleme a termék kódja, a másik elem pedig a rendelés azonosítója. Mivel a kulcs egyik elme sem lehet NULL (egyed-integritás szabálya), nem tudjuk megoldani olyan termék tárolását, amelyet még nem rendeltek meg.

A rekordok fölvételét meghiúsító kezelési problémát bővítési anomáliának nevezzük.

5.3.2 Módosítási anomália

A rekordok módosításakor a redundáns tárolás miatt előfordulhat, hogy a változtatást több rekordban is el kell végezni. Ha valamelyik frissítés elmarad, az adatbázis inkonzisztens lesz.

Tegyük fel, hogy ***Kovács Lehel*** *lakcíme megváltozik. Az adatbázis mostani változatában a vevő által vásárolt összes termék rekordjában el kell végezni a cím átírását. Ez fölöslegesen sok munka, azontúl hibalehetőséget is rejt. Ha a három rekord közül akár egyben is elfelejtjük megváltoztatni a címet – vagy valamelyik változtatáskor gépelési hibát ejtünk –, az adatbázisunk hibás adatokat*

fog tartalmazni. Az adatbázis inkonzisztens lesz, azaz ellentmondásokat tartalmaz, nem tükrözi pontosan a valóságot.

A redundáns tábla rekordjainak változtatásakor fellépő kezelési nehézségeket nevezzük változtatási anomáliának.

5.3.3 Törlési anomália

A törlési anomália akkor jelentkezik, ha egy egyed törlésekor másik egyed adatai is elvesznek.

Tegyük föl például, hogy ***Moka Ibolya*** *visszavonja* ***R124****-es kódú megrendelését. Ha rendelést töröljük, elvész a* ***160 GB-os Freecom*** *merevlemez összes jellemzője is.*

Törlési anomáliáról beszélünk, ha a redundáns adatszerkezet miatt egy egyed rekordjának törlésekor más egyed tulajdonságai is elvesznek.

5.4 MEZŐK FÜGGŐSÉGE

Példánkban azért keletkezett redundancia, mert nem „összetartozó” attribútumokat helyeztünk közös táblába. A termékek tulajdonságtípusait a vevők mezőivel azonos relációba illesztettük. Ha helyesen jártunk volna el, akkor minden egyedtípus saját táblát kapott volna. A táblákba csak az egymáshoz valamilyen módon kötődő, az adott egyedtípust jellemző mezők kerültek volna. Az ilyen mezők között sajátos viszony van, amelyet az úgynevezett **funkcionális függés** megismerésével tárhatunk fel.

5.4.1 Funkcionális függés

Funkcionális függésről beszélünk akkor, amikor egy tábla valamelyik mezőjének értékei egyértelműen meghatározzák egy másik mező értékeit.

A ’B’ mező akkor függ funkcionálisan ’A’ mezőtől, ha ’A’ egy bizonyos értékének minden előfordulásához ’B’-nek csak egy bizonyos értéke tartozhat. Az ’A’ mezőt ilyenkor **determinánsnak**, a ’B’-t **függő mezőnek** nevezzük.

A ’B’ függ ’A’-tól funkcionális függés jelölésére az alábbi formalizmust használják:

```
A→B
```

*Példánkban kijelenthetjük, hogy a **vNev** mező (vevő neve) funkcionálisan függ az **rKod** (rendelés kódja) mezőtől.*

*Az **R123** rendeléskód minden előfordulásához azonos név, **Kovács Lehel** kapcsolódik, a **R125** rendeléskód pedig a **Keller Berta** nevet determinálja. Figyeljük meg, hogy a függés nem kölcsönös. A **Keller Berta** névhez két különböző rendeléskód (**R125, R126**) is tarozik, tehát a **vNev** függ az **rKod**-tól, de az **rKod** nem függ a **vNev**-től!*

*A mezők vizsgálata alapján hamar rájövünk, hogy a rendeléseket tároló táblában az **rDatum, vNev, vCim, vTelefon** attribútumok mindegyike függ az **rKod**-tól, a **tKod, rDarab, tNev, tAr** azonban nem.*

*A **tNev**, és a **tAr** mezőket viszont a **tKod** determinálja.*

*Érdekes az **rDarab** mező helyzete. Ezt az attribútumot ugyanis egyetlen önálló mező sem determinálja, az összetett kulcs azonban igen. Az **rDarab** az **rKod, tKod** összetett kulcstól függ.*

A relációk attribútumainak egy része szerepel a tábla kulcsában a többi mező nem. A kulcsmezők **az elsődleges kulcs elemei, a** nem kulcsmezők **pedig nem részei a kulcsnak. Amikor az „összetartozó” mezőkről beszélünk, akkor azokat az attribútumokat keressük, amelyek azonos kulcsmezőtől függnek funkcionálisan.**

Az azonos kulcsmezőtől függő attribútumok föltárásához szükségünk lesz két **speciális funkcionális függés**, a **részleges** és a **tranzitív függés** megértésére.

A funkcionális függés két tetszőleges mező függő viszonyát jelenti. A redundancia csökkentésekor azonban a kulcs- és nemkulcs mezők közötti függéseket vizsgáljuk.

5.4.2 Részleges funkcionális függés

A **részleges funkcionális függés a rosszul megtervezett** táblákat jellemzi. Akkor fordul elő, ha a determináns nem egyetlen mező, hanem több mező kombinációja.

A részleges szó arra utal, hogy a függő mező nem függ a determináns egészétől, de annak valamelyik elmétől (részétől) igen. A részleges funkcionális függés egy másik meghatározása a determinánst a kulcsra szűkíti le.

> **Részleges funkcionális függés esetén egy nemkulcs mező függ az összetett kulcs valamelyik elemétől, de a kulcs egészétől nem.**

Példaadatbázisunkban az ***rKod****, és* ***tKod*** *mezőkből álló összetett kulcs található. Láttuk, hogy az* ***rDatum, vNev, vCim, vTelefon*** *mezők csak a kulcs egyik résztől az* ***rKod****-tól függnek. A* ***tNev****, és a* ***tAr*** *viszont csak a kulcs másik részétől, a* ***tKod*** *mezőtől függ. A táblában tehát részleges funkcionális függés van.*

rKod	rDatum	vKod	vNev	vCim	vTelefon	rDarab	tKod	tNev	tAr
R123	2010.06.07.	V19	Kovács Lehel	3300 Eger Thorma János utca , 13	(70) 909-4964	1	T224	Samsung SCX-4216F nyomtató-másoló-scanner+FAX	87 989
R123	2010.06.07.	V19	Kovács Lehel	3300 Eger Thorma János utca , 13	(70) 909-4964	3	T232	HP LaserJet 1015	47 366
R123	2010.06.07.	V19	Kovács Lehel	3300 Eger Thorma János utca , 13	(70) 909-4964	2	T237	Chicony KB9810 Win98 HUN PS/2	1 034
R124	2010.04.14.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T151	160 GB Freecom FHD-3 7200rpm USB2+FWire	48 290
R125	2010.12.09.	V05	Keller Berta	3501 Miskolc Epreskert utca , 33	(20) 942-4648	1	T215	DVD+-RW MSI DR16-B2 dob. Double Layer	12 639
R125	2010.12.09.	V05	Keller Berta	3501 Miskolc Epreskert utca , 33	(20) 942-4648	1	T238	AMD Athlon 64+FAN 3200+ dob. 939 pin	44 990
R126	2011.03.05.	V29	Keller Berta	3300 Eger Kolmann utca , 47	(70) 626-6488	1	T233	Chicony 790C-BS Mult. fekete-ezüst HUN PS/2	1 089
R126	2011.03.05.	V29	Keller Berta	3300 Eger Kolmann utca , 47	(70) 626-6488	1	T196	17" LG L1720P TFT 3 év DVI, 16ms	65 890
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T238	AMD Athlon 64+FAN 3200+ dob. 939 pin	44 990
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T196	17" LG L1720P TFT 3 év DVI, 16ms	65 890
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	4	T152	250 GB Freecom FHD-3 7200rpm USB2	51 700
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T239	Celeron 1800A+FAN S478	11 649
R128	2011.01.05.	V16	Dirkó Rebeka	3600 Ózd Bajnár utca , 97	(60) 634-8517	2	T182	Leadtek A180 DDR 64MB AGP8X T	6 270
R128	2011.01.05.	V16	Dirkó Rebeka	3600 Ózd Bajnár utca , 97	(60) 634-8517	1	T221	DVD+-RW Plextor PX-716A DL OEM	24 090

27. ábra Részleges funkcionális függések a rendelés táblában

Az összetett kulcs nem feltétlenül jelent részleges funkcionális függést, hiszen előfordul, hogy a függő mező az kulcs egészétől függ, annak elemeitől viszont nem.

Azt azonban tudni kell, hogy ez a részleges függés csak összetett kulcs esetén fordulhat elő.

Amikor egy mező függ a kulcstól, de annak elemeitől nem, akkor teljes funkcionális függésről **beszélünk.**

*A **rendelés** táblában az **rDarab** mező teljes funkcionális függéssel függ az összetett kulcstól, hiszen az **rKod,tKod** mezőkombináció egészétől függ, az **rKod**, illetve a **tKod** attribútumoktól külön-külön nem. Nem a rendelésnek, és nem a terméknek, hanem a **rendelésben szereplő** terméknek van darabszáma.*

A rendelés táblában tehát találunk részleges és teljes funkcionális függést is.

5.4.3 Tranzitív függés

A tranzitív függés szintén a rosszul tervezett táblák sajátja. Azt a függést nevezzük tranzitívnek, amikor a 'C' mező a 'B' mezőn keresztül függ az 'A' mezőtől, azaz 'B' függ 'A'-tól, 'C' pedig 'B'-től.

A→B és B→C

Az alábbi ábra csak a rendelés és a vevő adatait tartalmazza.

*A **vNev, vCim**, és **vTelefon** mezők a **vKod**-tól függnek, a **vKod** attribútumot azonban az **rKod** mező determinálja.*

Ha feltételezzük, hogy a determináns a kulcs, akkor a tranzitív függés definíciója következő.

> **Egy nem kulcs mező tranzitíven függ a kulcstól, ha egy másik, a kulcstól függő mezőtől is függ.**

rKod	rDatum	vKod	vNev	vCim	vTelefon
R123	2010.06.07.	V19	Kovács Lehel	3300 Eger Thorma János utca , 13	(70) 909-4964
R124	2010.04.14.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072
R[illegible]	2010.12.09.	V[illegible]	Keller Berta	3501 Miskolc Epreskert utca , 33	(20) 942-4648
R126	2011.03.05.	V29	Keller Berta	3300 Eger Kolmann utca , 47	(70) 626-6488
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072
R128	2011.01.05.	V16	Dirkó Rebeka	3600 Ózd Bajnár utca , 97	(60) 634-8517

28. ábra Tranzitív függések

> **A tranzitív függésben lévő mező (’C’) a** tranzitív determinánson **(’B’) keresztül függ a determinánstól (’A’).**

5.5 NORMALIZÁLÁS

Egy relációs adatbázis modellje akkor lesz helyes, a redundáns tárolás akkor szűnik meg, ha csak az azonos egyedtípusokat jellemző mezőket tároljuk közös táblában.

Ilyenkor a modell minden lehetséges táblájára igazak az alábbi állítások:

- minden nemkulcs mező **függ a kulcstól**,
- a nemkulcs mezők a kulcsnak **minden elemétől** függnek,
- a nemkulcs mezők **csak a kulcstól** függnek.

A redundanciától mentes adatbázisok kialakítását a Codd által kidolgozott, és ’72-ben publikált módszerrel, a **normalizálással** végezhetjük.

> **A normalizálás több lépéses** átalakítási folyamat**, amelynek** fázisait normálformáknak **nevezzük.**

Az átalakítás során fokozatosan újabb és újabb állapotba alakítjuk a modellt. Összesen hat állapotot – úgynevezett **normálformát** – különböztetünk meg. Az első, második és harmadik normálformát (1NF, 2NF, 3NF) az úgynevezett Boyce–Codd-normálforma (BCNF), majd a negyedik (4NF) és ötödik (5NF) normálforma követi. Egy új állapot kialakításakor a meglévő táblákat szükség esetén több táblára bontjuk, azaz **dekomponáljuk**, és az azonos determinánsoktól függő mezőket külön-külön táblákba rendezzük.

A gyakorlat azt mutatja, hogy 3NF elérésével már használható, redundanciától mentes adatbázist kapunk. Ezért a relációs adatbázisok logikai modellezésekor a 3NF kialakítása az előírás. Az ezt követő normálformákat magasabb normálformákként is említik. Elérésükre csak egészen speciális esetekben van szükség.

Tananyagunkban a 3NF eléréséig tanuljuk meg a normalizálást.

5.5.1 UNF, 0NF

Adatbázisunk modellje akkor van első normálformában, ha az egyedtípusok, tulajdonságtípusok és kapcsolatok tárolását a relációs adatmodell struktúrájának és integritási szabályainak megfelelően kialakított modellben képzeljük el.

A szabályok között szerepel, hogy minden mezőnek atomi értéket kell tartalmaznia. Ha ez a feltétel nem teljesül, akkor az 1NF-et megelőző állapotról: 0NF-ről, vagy nem normalizált formáról (Unnormalised Form) beszélünk.

rKod	rDatum	vKod	vNev	vCim	vTelefon	rDarab	tKod	tNev	tAr
R123	2010.06.07.	V19	Kovács Lehel	3300 Eger Thorma János utca , 13	(70) 909-4964	3	T237	Chicony KB9810 Win98 HUN PS/2	1034
						2	T232	HP LaserJet 1015	47366
						1	T224	Samsung SCX-4216F nyomtató-másoló-scanner+FAX	87989
R124	2010.04.14.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T151	160 GB Freecom FHD-3 7200rpm USB2+FWire	48290
R125	2010.12.09.	V05	Keller Berta	3501 Miskolc Epreskert utca , 33	(20) 942-4648	1	T238	AMD Athlon 64+FAN 3200+ dob. 939 pin	44990
						1	T215	DVD+-RW MSI DR16-B2 dob. Double Layer	12639
R126	2011.03.05.	V29	Keller Berta	3300 Eger Kolmann utca , 47	(70) 626-6488	1	T196	17" LG L1720P TFT 3 év DVI, 16ms	65890
						1	T233	Chicony 790C-BS Mult. fekete-ezüst HUN PS/2	1089
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	4	T152	250 GB Freecom FHD-3 7200rpm USB2	51700
						1	T238	AMD Athlon 64+FAN 3200+ dob. 939 pin	44990
						1	T196	17" LG L1720P TFT 3 év DVI, 16ms	65890
						1	T239	Celeron 1800A+FAN S478	11649
R128	2011.01.05.	V16	Dirkó Rebeka	3600 Ózd Bajnár utca , 97	(60) 634-8517	1	T221	DVD+-RW Plextor PX-716A DL OEM	24090
						2	T182	Leadtek A180 DDR 64MB AGP8X T	6270

29. ábra UNF

A fenti ábra olyan táblát mutat, amelyben többértékű mezőkben (**`rDarab, tKod,tNev, tAr`**) tároltuk a rendelések egyes adatait. Az adatszerkezet UNF-ben van. Figyeljük meg, hogy ebben a változatban még egyszerű kulcsot (**`rKod`**) használhatnánk, hiszen a rendelésekhez kapcsolódó termékadatok a rendelés rekordjának többértékű mezőiben találhatók.

5.5.2 1NF

Az első normálformát tehát akkor érjük el, ha a relációs adatmodell minden szabályának eleget tévő struktúrát alakítunk ki.

Ha a 0NF állapotot 1NF-re akarjuk alakítani, akkor a többértékű mezők értékeit külön rekordokban kell elhelyezni, az egyértékű mezőket pedig többszö-

rözve kell tárolni! Ahogyan az ábra is mutatja, ezzel a lépéssel nem csökkent, hanem ideiglenesen növekedett a redundancia.

rKod	rDatum	vKod	vNev	vCim	vTelefon	rDarab	tKod	tNev	tAr
R123	2010.06.07.	V19	Kovács Lehel	3300 Eger Thorma János utca , 13	(70) 909-4964	3	T232	HP LaserJet 1015	47 366
R123	2010.06.07.	V19	Kovács Lehel	3300 Eger Thorma János utca , 13	(70) 909-4964	2	T237	Chicony KB9810 Win98 HUN PS/2	1 034
R123	2010.06.07.	V19	Kovács Lehel	3300 Eger Thorma János utca , 13	(70) 909-4964	1	T224	Samsung SCX-4216F nyomtató-másoló-scanner+FAX	87 989
R124	2010.04.14.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T151	160 GB Freecom FHD-3 7200rpm USB2+FWire	48 290
R125	2010.12.09.	V05	Keller Berta	3501 Miskolc Epreskert utca , 33	(20) 942-4648	1	T238	AMD Athlon 64+FAN 3200+ dob. 939 pin	44 990
R125	2010.12.09.	V05	Keller Berta	3501 Miskolc Epreskert utca , 33	(20) 942-4648	1	T215	DVD+-RW MSI DR16-B2 dob. Double Layer	12 639
R126	2011.03.05.	V29	Keller Berta	3300 Eger Kolmann utca , 47	(70) 626-6488	1	T196	17" LG L1720P TFT 3 év DVI, 16ms	65 890
R126	2011.03.05.	V29	Keller Berta	3300 Eger Kolmann utca , 47	(70) 626-6488	1	T233	Chicony 790C-BS Mult. fekete-ezüst HUN PS/2	1 089
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	4	T152	250 GB Freecom FHD-3 7200rpm USB2	51 700
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T239	Celeron 1800A+FAN S478	11 649
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T238	AMD Athlon 64+FAN 3200+ dob. 939 pin	44 990
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072	1	T196	17" LG L1720P TFT 3 év DVI, 16ms	65 890
R128	2011.01.05.	V16	Dirkó Rebeka	3600 Ózd Bajnár utca , 97	(60) 634-8517	1	T221	DVD+-RW Plextor PX-716A DL OEM	24 090
R128	2011.01.05.	V16	Dirkó Rebeka	3600 Ózd Bajnár utca , 97	(60) 634-8517	2	T182	Leadtek A180 DDR 64MB AGP8X T	6 270

30. ábra 1NF kialakítása

5.5.3 2NF

Az 1NF adatbázis táblái – mint látjuk – erősen redundánsak. Ha csökkenteni szeretnénk a fölöslegesen ismétlődő adatok mennyiségét, akkor adatbázisunkat 2NF-ba kell alakítani.

Az olvasó minden bizonnyal fölismeri, hogy az 1NF állapotot bemutató ábra megegyezik azzal az adatszerkezettel, amelyet a részleges funkcionális függés példájaként korábban már láthattunk. Ez egyáltalán nem véletlen, a 2NF elérésének kulcsa éppen részleges funkcionális függésekben, illetve megszüntetésükben van!

> **A modell akkor lesz 2NF-ben, ha már 1NF-ben van, és minden táblában megszűntetjük a részleges funkcionális függéseket.**

Dekompozíció

A 2NF kialakításakor azt a táblát, amelyben részleges funkcionális függés van, **több táblára bontjuk**. Annyi új táblát kell létrehozni, amennyi az összetett kulcs elemeinek a száma.

> **Egy tábla több relációra bontását dekompozíciónak nevezzük.**

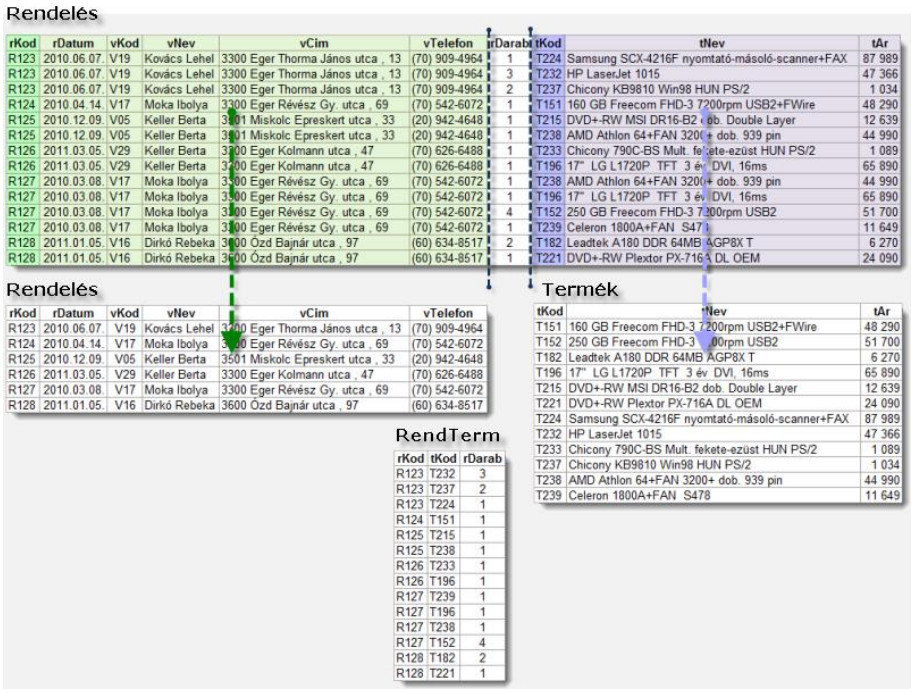
Az **új táblákba az összetett kulcs elemei** kerülnek a **csak tőlük függő összes mezővel** együtt. Az új táblák **azonosítói** az eredeti összetett **kulcs elemei** lesznek.

*Esetünkben a rendelés táblát két új táblára (**rendelés, termék**) kell bontani, hiszen az összetett kulcs két elemből áll (**rKod, tKod**)*

A rendeléseket és a termékeket külön tároló táblákban már nem lesz összetett kulcs, tehát megszűnik a részleges funkcionális függés.

*A dekompozíció eredményeként kapott **rendelés** táblába csak a rendelések, a **termék** táblába pedig csak a termékek attribútumai kerültek. Mivel a két egyedtípust különválasztottuk, nem kell – és nem is szabad – többszörözni a rekordokat.*

Az ábrán látható **RendTerm** tábla funkciójáról, és az **rDarab** mezőről pár sorral lejjebb olvashatunk.



31. ábra A rendelés tábla dekompozíciója

Még a redundancia ilyen látványos csökkenése sem vonhatja el a figyelmünket két fontos tényezőről! Előfordulhat, hogy az eredeti, 1NF-ben lévő `rendelés` **tábla tartalmaz olyan mezőt, vagy mezőket, amelyek nem részleges, hanem teljes függéssel függtek a tábla kulcsától. Az ilyen mezők – mint például az** `rDarab` **– egyik új táblába sem illeszthetők be, hiszen nem az 1NF tábla kulcsának elemeitől, hanem a teljes kulcstól függtek.**
A másik figyelemre méltó dolog, hogy amíg az eredeti táblában minden rendelés egy termék adataival együtt tárolódott, addig a dekomponálás után a rendelések és termékek közötti kapcsolatról semmi sem árulkodik.

Kapcsolatok kialakítása

A relációs adatmodellben úgy írjuk le két tábla kapcsolatát, hogy a kapcsolódó táblában létrehozott **idegenkulcs** értékeivel hivatkozunk az elsődleges tábla **azonosítójának** értékeire. A kérdés csak az, hogy melyik tábla lesz az elsődleges, és melyik a kapcsolódó. Azaz, **hová kerül az idegen kulcs**, melyik tábla rekordjai hivatkoznak a másik tábla rekordjaira?

Példánkban a ***rendelés*** *és* ***termék*** *táblák n:m kapcsolatát kell tárolni.*

Vagy a ***rendelés*** *táblába illesztett idegen kulcsban tároljuk a rendelésben szereplő termékek azonosítóit, vagy fordítva: a* ***termék*** *táblában helyezzük el a kapcsolódó rendeléseknek azonosítóját.*

n:m kapcsolat esetén sajnos bármelyik táblát is választanánk kapcsolódó táblának, az idegen kulcs mindenképpen több értéket tartalmazhatna. A relációs adatmodell azonban nem engedi meg többértékű mezők használatát!

A feladat egy harmadik, összekötő funkciót betöltő tábla, az úgynevezett **kapcsolótábla** közbeiktatásával oldható meg. A kapcsolótáblába helyezzük az **eredeti összetett kulcsot**, és minden tőle **teljes függéssel függő mezőt**. Ezzel a lépéssel mindkét problémánk megoldódik.

*Példánkban a **rendterm** tábla látja el a kapcsolótábla szerepét. Azonosítója az **rKod** és t**K**od mezőkből álló eredeti összetett kulcs, amelytől teljes függéssel függ a **rDarab** mező. A tábla minden rekordja egy **rendelés** és egy **termék** kapcsolatát írja le, de tárolja azt is, hogy az adott rendelésben hány darabot rendeltek a termékből. Az **rKod** és **tKod** mezők önmagukban idegen kulcsok, az egyik egy rendeléssel, a másik egy termékkel való kapcsolatot ír le.*

A kapcsolótábla rekordjaiban egy bizonyos rendeléskód annyiszor fordul elő (különböző termékkódokkal), ahány terméket tartalmaz a megrendelés. A termékkódok is ismétlődhetnek, hiszen egy termék több rendelésben is szerepelhet.

A kapcsolótábla **azonosítója** tehát az **eredeti összetett kulcs**, azonban annak elemei valójában a dekompozícióval létrehozott táblák rekordjaira hivatkozó idegen kulcsok. Segítségükkel a kapcsolótábla minden rekordja kapcsolódik a dekompozícióval kialakított táblák egy-egy rekordjához, azaz összeköti, összekapcsolja azokat.

A relációs adatmodellben két reláció között nem lehet közvetlen n:m kapcsolat. A több-több kapcsolatban lévő táblákat a két reláció azonosítóinak értékét idegen kulcsként tároló kapcsolótábla beszúrásával kötjük össze.

5.5.4 3NF

A részleges funkcionális függések megszüntetése után még mindig maradhat anomáliákat okozó redundancia az adatbázis tábláiban.

Példánkban is fölfedezhetjük, hogy Moka Ibolya két rendelést is leadott, személyes adatait (címét, és telefonszámát) pedig mindegyik esetben újra tároltuk.

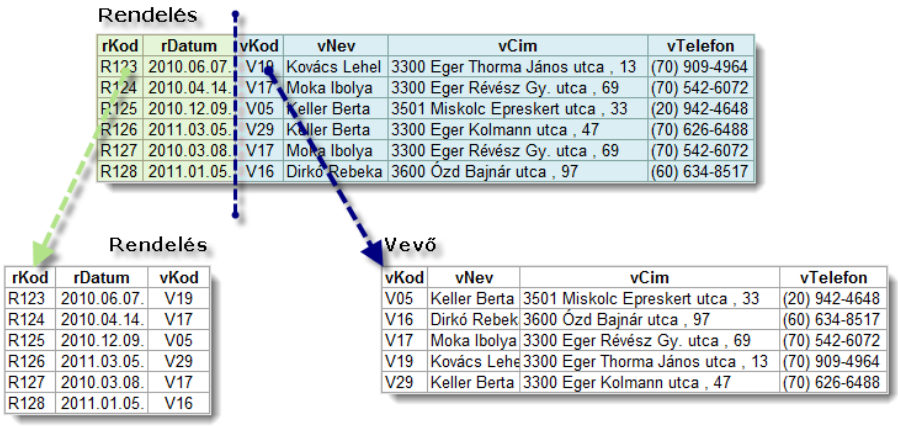
rKod	rDatum	vKod	vNev	vCim	vTelefon
R123	2010.06.07.	V19	Kovács Lehel	3300 Eger Thorma János utca , 13	(70) 909-4964
R124	2010.04.14.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072
R125	2010.12.09.	V05	Keller Berta	3501 Miskolc Epreskert utca , 33	(20) 942-4648
R126	2011.03.05.	V29	Keller Berta	3300 Eger Kolmann utca , 47	(70) 626-6488
R127	2010.03.08.	V17	Moka Ibolya	3300 Eger Révész Gy. utca , 69	(70) 542-6072
R128	2011.01.05.	V16	Dirkó Rebeka	3600 Ózd Bajnár utca , 97	(60) 634-8517

32. ábra Tranzitív függés a rendelés táblában

Ennek az az oka, hogy a részleges funkcionális függést megszüntető dekompozíció nem feltétlenül válogatja külön táblákba az összes különböző egyedtípus attribútumait. Ha a maradék redundanciától is meg szeretnénk szabadulni, akkor a tranzitív függéseket is meg kell szüntetnünk.

Egy relációs adatbázis modellje akkor van 3NF-ben, ha a 2NF változatban megszüntetjük a tranzitív függéseket.

A feladat elvégzésére ismét dekompozíciót alkalmazunk. A tranzitív determinánst a tőle függő mezőkkel együtt új tábla helyezzük. Az új relációban az eredeti tranzitív determináns lesz a kulcs.



33. ábra Tranzitív függés megszűntetése

Természetesen most is fontos az új táblák közötti kapcsolatok leírása. A leválasztott tábla azonosítóját idegen kulcsként tároljuk az eredeti tábla dekomponálás után megmaradt részében. Az új és az eredeti tábla között ilyenkor 1:1, vagy 1:n kapcsolat alakul ki. 1:n viszony esetén az idegen kulcs a kapcsolat 'N',

azaz több oldalára kerül. Így biztosítható ugyanis, hogy ne legyen többértékű mező.

Példánkban a rendelés tábla **vNev, vCim, vTelefon** *mezői a* **vKod** *mezőn keresztül tranzitíven függnek az* **rKod** *kulcstól. A* **vKod** *mezőt minden tőle függő attribútummal együtt az újonnan létrehozott* **vevő** *táblában helyezzük el. A két tábla kapcsolatát a rendelés táblában kialakított idegen kulccsal biztosítjuk. Most nincsen szükség kapcsolótáblára, hiszen a* **vevő- rendelés** *táblák 1:n kapcsolatban vannak.*

A relációs adatmodellben két tábla kapcsolatát idegen kulccsal írjuk le.

n:m kapcsolat esetén a két tábla azonosítóit tartalmazó idegen kulcsokat egy új, ún. kapcsolótáblában helyezzük el.

1:n kapcsolat esetén az 1 oldal azonosítójának értékei a több oldalon elhelyezett idegenkulcsba kerülnek.

1:1 kapcsolat esetén általában mindegy, hogy melyik táblába kerül az idegenkulcs, de ezzel kapcsolatban rövidesen megfogalmazunk egy kivételt.

Ha a normalizálást következetesen végrehajtva elérjük, hogy a modell minden táblája harmadik normálformában legyen, akkor elmondhatjuk, hogy az adatszerkezet alkalmas arra, hogy abban redundanciamentesen és kezelési anomáliákat megelőzve tároljuk adatainkat. Ezzel tulajdonképpen megnyílik a továbblépés az adatbázis fizikai modellezésének lehetősége.

5.6 ER-MODELL ÁTALAKÍTÁSA RELÁCIÓS ADATMODELLRE

Korábban megismertük az adatbázis-modellezés koncepcionális szintjén használható ER-modellt, majd pedig a logikai modellek közé tartozó relációs adatmodellt. Mindeddig azonban egy szót sem ejtettünk arról, hogyan használhatjuk fel a korábban elkészített ER-modellt az alacsonyabb szintű relációs modell megalkotáskor.

Az átalakítás egyes esetekben roppant egyszerű, máskor viszont némi gondolkodást igényel. Világos például, hogy az ER-modell egyedei a relációs adatmodell tábláinak, az egyedek tulajdonságai pedig a táblák mezőinek felelnek

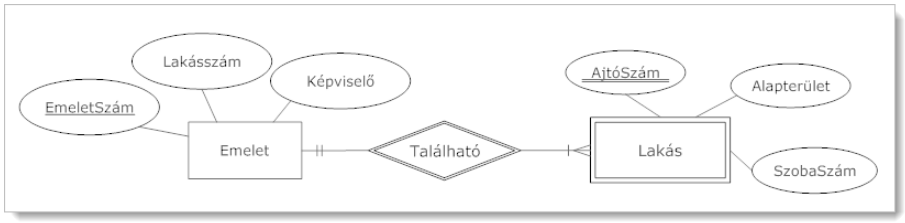
meg. Problémát az okozhat, hogy az ER-modell számos olyan elem felhasználását is lehetővé teszi, amelynek nincs explicit megfelelője a relációs modellben. Utóbbiban nem találkoztunk például olyan strukturális elemekkel, mint a gyenge egyed, az alosztály, vagy a kategória, a többértékű mezők használata pedig kifejezetten tilos. Tévedünk azonban, ha ebből arra következtethetünk, hogy az ER-modell nem alakítható át maradéktalanul relációs modellre. Egyes ER-elemek ugyanis közvetlenül, még mások a relációs struktúra rugalmasságát kihasználva képezhetők le.

Az átalakítás nem bonyolult, ugyanis a lépéseket pontosan rögzített módszerek, úgynevezett **leképezési szabályok** irányítják. Ezek betartásával a jól elkészített ER-modellből redundanciától mentes relációs modell alakítható ki.

5.6.1 Egyedek és attribútumok leképezése

Mint láttuk, az egyedek leképezése a legegyszerűbb. Az ER-modell minden erős egyede egy reláció lesz, amelynek mezői az egyed tulajdonságai. A tábla kulcsa az egyed azonosító attribútuma lesz. Az esetleges összetett attribútumok elemeikre bontva alkotnak mezőket.

A gyenge egyedek leképezésekor az erős és gyenge egyed egyaránt egy-egy táblába kerül. A gyenge egyed azonosítója az erős egyedre hivatkozó idegen kulcsból és a parciális kulcsból képzett összetett kulcs lesz.



34. ábra Erős és gyenge egyed

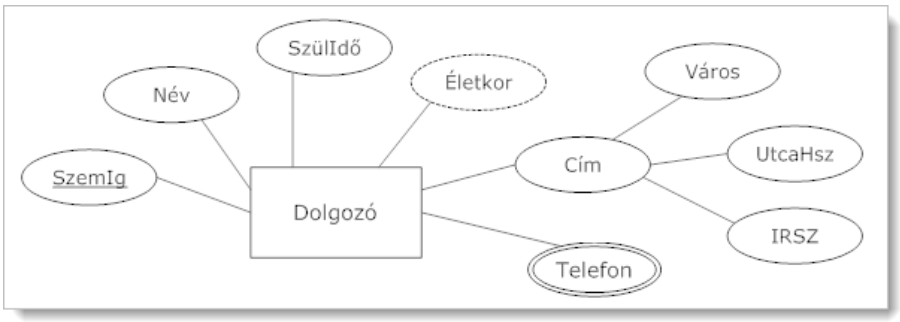
A fenti ábrán látható ER-modell relációs megfelelője az alábbi:

```
Emelet (EmeletSzám, Lakásszám, Képviselő)
Lakás(EmeletSzám, AjtóSzám, Alapterület, SzobaSzám)
```

5.6.2 Többértékű attribútumok

Mint tudjuk, a relációs adatmodell nem engedi meg a többértékű attribútumok használatát. Ezért ha ilyennel találkozunk, akkor két mezőből álló új táblát hozunk létre. Az egyik mező a többértékű attribútumnak felel meg, a másik

az ER-modell egyedének kulcsértékeit tároló idegen kulcs lesz. A két mező összetett kulcsot alkot az új táblában. Az eredetileg többértékű attribútum minden lehetséges értékét külön rekordba tesszük az új táblában.



35. ábra Többértékű attribútum

A fenti példában a ***Dolgozó*** *egyed* ***Telefon*** *attribútuma többértékű, a* ***Cím*** *pedig összetett. A leképezés a következőképpen történik.*

```
Dolgozó(SzemIg, Név, SzülIdő, Életkor, Város, UtcaHsz,
IRSZ)
Telefon(SzemIg,Telefon)
```

5.6.3 Kapcsolatok

Két egyed kapcsolatát (ER1, ER2) úgy képezzük le, hogy mindkét egyednek egy-egy táblát (T1,T2), az attribútumaiknak pedig a táblák mezőit feleltetjük meg. Az **egyik táblában idegen kulcsot** is létrehozunk, amely a **másik tábla kulcsának** értékeit tárolva hivatkozik annak rekordjaira.

A kapcsolatok leképezésének alapjai egyszerűek. Az ER-modell azonban igen árnyaltan képes kifejezni az egyedek viszonyait, azért a fenti séma számos változatával találkozhatunk.

Fontos megjegyezni, hogy a redundancia teljes kiküszöbölése az alkalmazott dekompozíciók miatt számos új tábla keletkezésével, és gyakran az adatbázis bonyolultságának növekedésével jár együtt. Az alábbi leképezési szabályok következetes betartásával teljesen le-

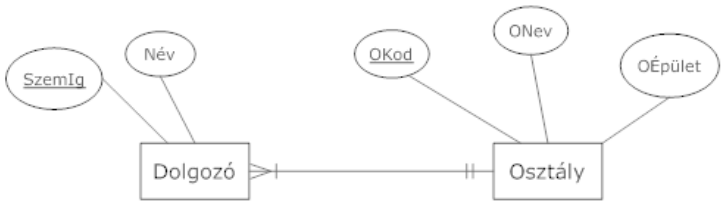
csökken ugyan a fölöslegesen tárolt adatok mennyisége, az adatbázis szerkezete azonban túlságosan is összetetté válhat.

Pontosan ezért az adatbázis-tervezők gyakran némi megalkuvást tanúsítanak, és megtűrik a nem kifejezetten veszélyes redundanciát.

Ha mégis betartunk minden szabályt, akkor nézetek kialakításával tehetjük barátságosabbá adatbázisunkat a felhasználók számára.

1:N kapcsolat

Az 1:N a leggyakoribb kapcsolattípus, amelyet az esetek többségében úgy lépezhetünk le, hogy az N oldalon helyezzük el az 1 oldalra hivatkozó idegen kulcsot.



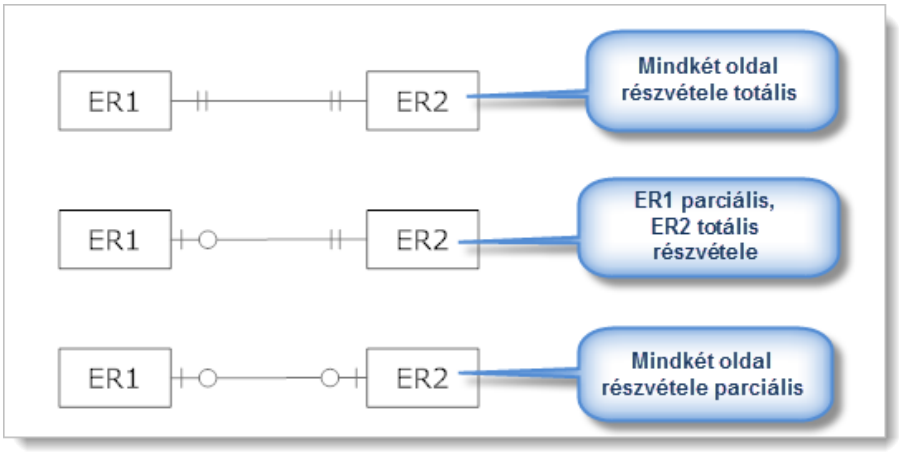
36. ábra 1:n kapcsolat

```
OSZTÁLY(OKod,ONev,OÉpület)
DOLGOZÓ(SzemIg, Név,…,OKod)
```

Amennyiben az N oldal részvétele nem kötelező (parciális), akkor a kapcsolatban nem szereplő rekordokban NULL értékek kerülnek az idegen kulcsokba. Az ismétlődő NULL értékek nem veszélyesek ugyan, de tagadhatatlanul redundanciát okoznak. Ha ettől meg akarunk szabadulni, akkor kapcsolótábla létrehozásával képezzük le a kapcsolatot. Ebben két idegen kulcsot helyezzük el. Az egyik az 1, a másik az N oldalra fog hivatkozni. A kapcsolótáblában csak a létező rekordkapcsolatokat tároljuk, így megszabadunk a NULL értékektől.

1:1 kapcsolat

Az egy-egy kapcsolat leképzését is az befolyásolhatja, hogy ER1- és ER2-egyedek kötelezően vesznek-e részt a kapcsolatban. Az alábbi három változat képzelhető el:



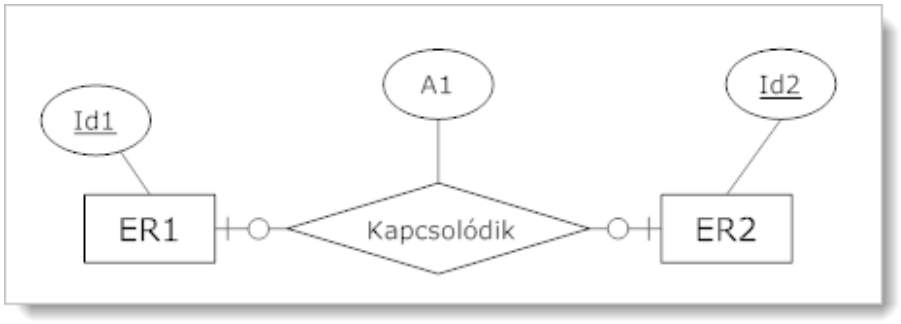
37. ábra 1:1 kapcsolatok

Az első esetben ER1-et és ER2-t egyetlen táblává képezzük le, úgy, hogy attribútumaik a tábla mezői lesznek. A tábla azonosítója lehet ER1, vagy ER2 kulcsa is.

Ez a megoldás akkor nem járható, ha ER1 és ER2 különböző kapcsolatokban vesz részt. Ilyenkor mindkét egyedet önálló táblaként képezzük le, az idegen kulcsot bármelyikben elhelyezhetjük.

A második esetben ER1 részvétel parciális. Ilyenkor az idegen kulcs az ER2-nek megfelelő táblába kerül. Ha ER1-be tennénk, akkor a kapcsolatban nem szereplő rekordokban NULL értékeket kellene tárolnunk az idegen kulcsban, ez pedig redundanciát jelentene.

A harmadik eset a legérdekesebb, ugyanis bármelyik oldalra is kerülne az idegen kulcs, abban előfordulhatnának NULL értékek. A precíz leképezés kapcsolótábla közbeiktatásával oldható meg. A kapcsolótáblában a két kapcsolt relációra hivatkozó idegen kulcsokat, és a kapcsolat esetleges attribútumait tároljuk. A tábla azonosítója bármelyik idegenkulcs lehet.



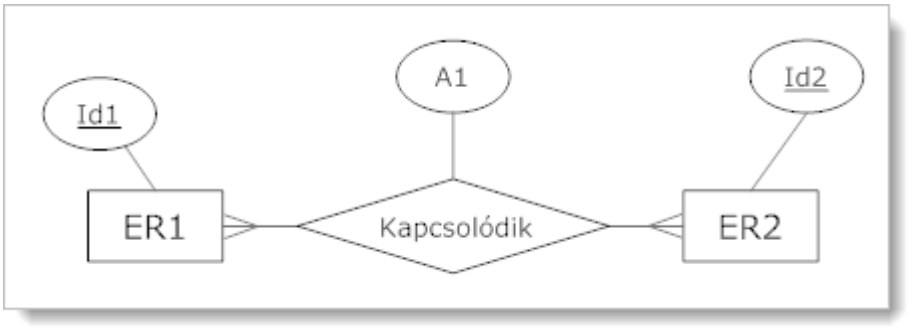
38. ábra 1:1 kapcsolat mindkét oldal parciális részvételével

A fenti 1:1 kapcsolat az alábbi módon képezhető le.

```
ER1(Id1,…)
KAPCSOLÓDIK(Id1,Id2,A1)
ER2(Id2,…)
```

n:m kapcsolat

Az n:m kapcsolat leképezése szinte teljesen megegyezik az 1:1 kapcsolat utóbbi változatával, amikor is mindkét oldal részvétele parciális volt. A különbség az, hogy a kapcsolótábla azonosítója a két idegen kulcsból képzett összetett kulcs lesz.

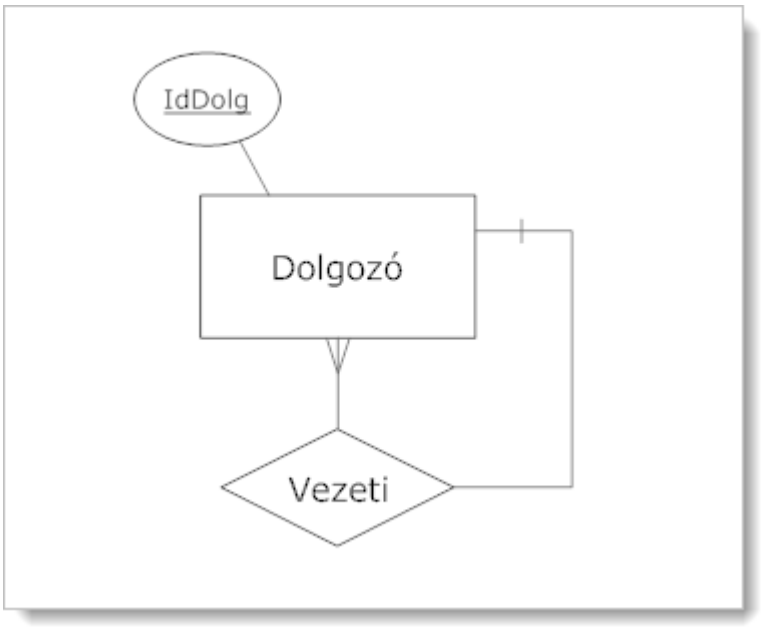


39. ábra n:m kapcsolat a kötelezőség jelölése nélkül

```
ER1(Id1,…)
KAPCS(Id1,Id2,A1)
ER2(Id2,…)
```

Rekurzív kapcsolat

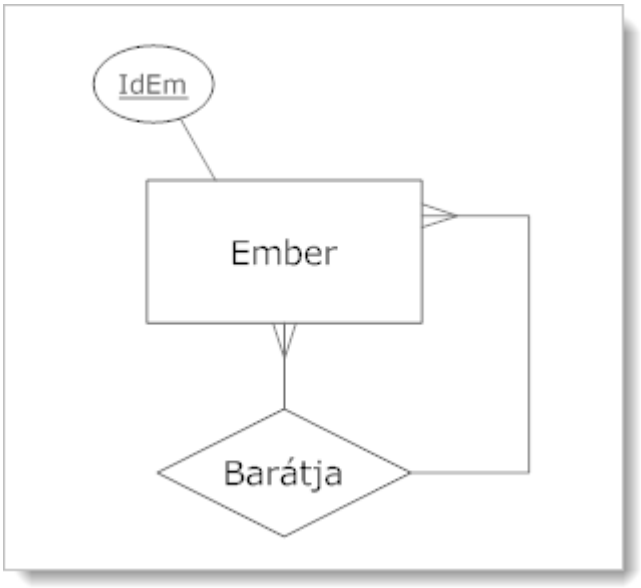
Ha a rekurzív kapcsolat számossága 1:1 (egy ember egy másik ember házastársa), vagy 1:n (egy dolgozó több dolgozó főnöke), akkor a leképezéssel létrehozott táblában elhelyezünk egy idegen kulcsot is, amely a tábla azonosítójának értékeit fogja tárolni, tehát a tábla egy másik rekordjára hivatkozik. Minden N oldali rekord a hivatkozott 1 oldali rekord azonosítójának értékét tárolja majd az idegen kulcsban.



40. ábra Rekurzív 1:n

```
DOLGOZÓ(IdDolg,…, DolgIK)
```

A rekurzív n:m kapcsolatot (bármelyik ember több embernek a barátja) az n:m kapcsolathoz hasonlóan képezzük le. Az egyedből leképzett tábla mellett kapcsolótáblát is létrehozunk, amelyben két idegen kulcsot helyezünk el. Mindkét idegenkulcs az önmagához kapcsolódó tábla egy-egy rekordjára hivatkozik majd.



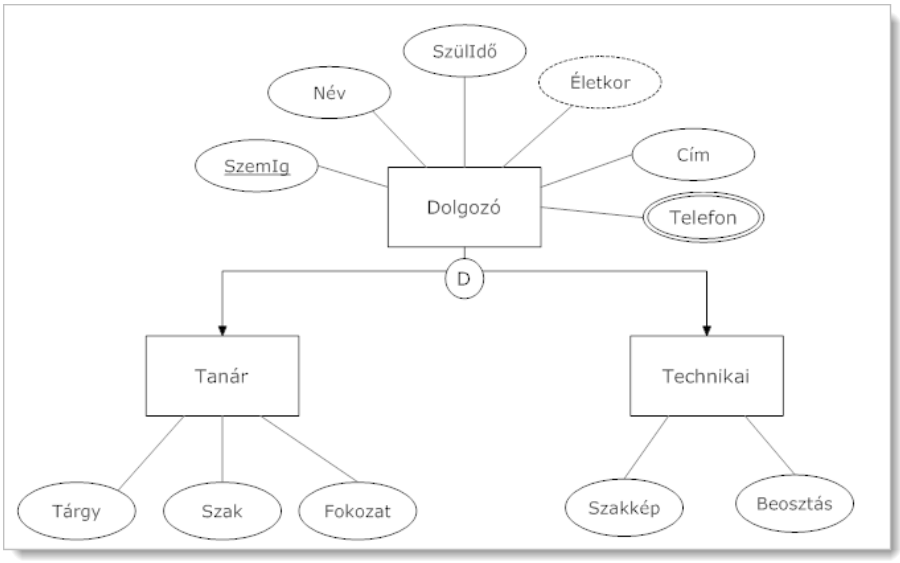
41. ábra Rekurzív n:m kapcsolat

```
EMBER(IdEm …)
BARÁTJA(IdEm1, IdEm2)
```

5.6.4 Specializáció

A specializáció – mint tudjuk – az öröklődés azon esete, amikor egy főosztályhoz egy vagy több speciális tulajdonságokkal rendelkező alosztály kapcsolódik.

Ilyenkor az alosztályok számára új relációkat hozunk létre, amelyek a kiegészítő tulajdonságoknak megfelelő mezőket tartalmazzák. Az alosztályok kulcs attribútuma azonos lesz a főosztály kulcsával.



42. ábra Specializáció

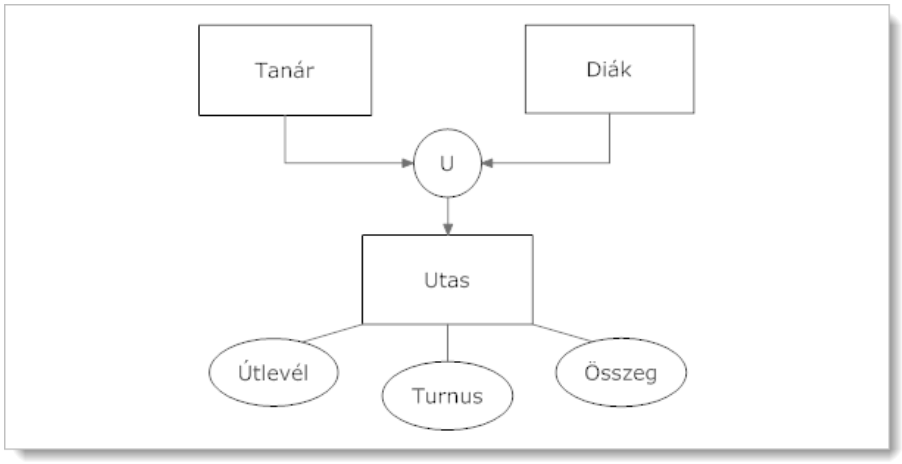
```
DOLGOZÓ(SzemIg,Név,SzölIdő,ÉletKor,Cím)
TELEFON(SzemIg,Telefon)
TANÁR(SzemIg,Tárgy,Szak,Fokozat)
TECHNIKAI(SzemIg,Szakkép,Beosztás)
```

A **Telefon** táblára azért van szükség, mert a **Dolgozó** tábla **Telefon** mezője többértékű.

5.6.5 Kategóriák

A kategóriák a főosztály-alosztály kapcsolat másik típusát képviselik. Ilyenkor két különböző főosztályhoz egyetlen alosztály kapcsolódik.

A relációs modellre történő leképezéskor a főosztályok és az alosztályok számára egyaránt egy-egy relációt hozunk létre, saját kulcsokkal. A főosztályokat kiegészítjük egy idegen kulccsal, amelyben az alosztály kapcsolódó rekordjainak azonosítóit tároljuk.



43. ábra Kategória ER-modellje

```
TANÁR(TanárId,…,UtasId)
DIÁK(DiákId,…,UtasId)
UTAS(UtasId,Útlevél,Turnus,Összeg)
```

A fő- és alosztályok kapcsolatát specializáció esetén a kizárólagosság, az átfedés és a részvétel kötelezősége, a kategóriák esetén pedig szintén a kötelezőség bonyolíthatja tovább. Ezekre a részletekre nem térünk ki.

5.7 ÖSSZEFOGLALÁS, KÉRDÉSEK

5.7.1 Összefoglalás

Mai leckénk nem véletlenül haladja meg az eddigiek terjedelmét. Tananyagában ugyanis a relációs modellek redundanciától mentes változatának kialakítása mellett a koncepcionális séma logikai modellé alakítását, az ER-modell relációs modellre történő leképezését is megtanulhattuk.

A tananyagból megtudtuk, milyen hibákat okozhat a rosszul modellezett relációs adatbázisokban kialakuló redundancia. Megismertük a bővítési, a változtatási és a törlési anomáliák jelentését, és megtanultuk, hogy a modellezési hibákat az attribútumok rossz elhelyezése okozza. A funkcionális függések megismerésével megtanultuk, hogyan ismerhetők fel egy egyedtípus saját attribútumai.

Megismertük a normalizálás technikáját, amelynek segítségével kiküszöbölhetjük a részleges és tranzitív függéseket, elérhetjük a redundanciától mentes állapot. Ebben az állapotban minden táblára igaz, hogy a nemkulcs mezők függnek a kulcstól, csak attól, és annak minden elemétől. Az így kialakított 3NF-ban lévő modell alapján olyan adatbázis készíthető, amelynek hatékony működését már nem akadályozzák a redundanciából eredő anomáliák.

Leckénk befejező szakaszában megismertük az ER-modell relációs adatmodellre alakításában felhasználható leképezési szabályokat. Megtanultuk az egyedek és attribútumaik, a különböző kapcsolatok, valamint az osztályok és alosztályok logikai adatmodellre alakításának technikáit.

5.7.2 Önellenőrző kérdések

1. Hogyan lehetséges, hogy egy minden szabályt betartó relációs modellre épülő adatbázis nem működik hatékonyan?
   - Úgy, hogy a relációs adatmodell szabályai nem korlátozzák a redundanciát. Ha az adatbázisban redundancia alakul ki, akkor különböző anomáliák nehezíthetik vagy gátolhatják meg az adatok kezelését.
2. Milyen hibákat okozhat a redundancia?
   - Megnöveli az adatbázis méretét, változtatási, bővítési és törlési anomáliát okozhat.
3. Mit értünk teljes funkcionális függés alatt?
   - Az olyan függést, amikor a függő mező a determináns egészétől függ, de annak elemeitől nem.
4. Mikor mondjuk egy relációs modellre, hogy 3NF-ban van?
   - Akkor, ha 2NF-ban van, és megszüntetjük a tranzitív függéseket. Ilyenkor minden táblában igaz, hogy a nemkulcs mezők függnek a kulcstól, csak attól, és annak minden elemétől. A 3NF-táblákban nincsenek sem részleges, sem tranzitív függések.
5. Hogyan képezhető le a relációs modellre a mindekét oldalon parciális részvételű 1:1 kapcsolat?
   - Az egyedek és a kapcsolat számára is egy-egy relációt hozunk létre. A kapcsolat relációjában a két másik táblára hivatkozó idegen kulcsokat, és a kapcsolat esetleges attribútumait tároljuk.

6. LECKE: ADATBÁZIS-KEZELÉS ÉS ADATBÁZIS-KEZELŐ RENDSZEREK

6.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

Az adatbázisok koncepcionális és logikai leképezéséhez használható modellek megismerése után elérkeztünk a fizikai modellezés szintjére. Arra a szintre, ahol az elkészített modell közvetlenül alkalmas lesz az adatbázis fizikai létrehozására. Az adatbázisok modellezésének ezen a szintjén is szükség lesz a modell valamiféle reprezentációjára, ábrázolására. Az eddig tanult modellek és reprezentációik elsősorban a tervező számára informatívak. Ezzel szemben a fizikai modellek reprezentációinak az adatbázisokat létrehozó és kezelő szoftverrendszerek, az úgynevezett adatbázis-kezelő rendszerek számára is információt kell hordozniuk.

A fizikai modellek valójában speciális, adatkezelő nyelvek segítségével megfogalmazott adatstruktúrákat definiáló leírások, amelyeket utasításként értelmezve az adatbázis-kezelő rendszerek képesek a modellnek megfelelő fizikai adatbázis létrehozására.

A fizikai modellezési szinten ennek megfelelően már komoly szerepet kapnak az adatbázis-kezelő rendszerek. Mai leckénkben az adatbázis-rendszer, adatbázis-kezelő rendszer, adatbázis-alkalmazás fogalmakat tekintjük át. Megvizsgáljuk a fogalmak jelentését, egymással való kapcsolatát, és összefoglaljuk az adatbázis-kezelő rendszerek legfontosabb jellemzőit.

- A tananyag olvasása közben tegyen különbséget az adatbázis-kezelő rendszer és az adatbázis-rendszer között!
- Összegezze az adatbázis-kezelő rendszerek funkcióit!
- Vonjon párhuzamot az ANSI-SPARC és a kétrétegű kliens-szerver architektúra között!

6.2 ADATMODELL, ADATBÁZIS

Az adatbázisok tervezésével modellezésével kapcsolatos ismereteinkben lassan eljutunk arra a legalsó absztrakciós szintre, ahol már fizikailag megvalósítható, közvetlenül implementálható, működő adatbázisok létrehozására alkalmas adatmodelleket készítünk.

Itt az ideje, hogy különbséget tegyünk az adatmodell és az adatbázis között. A modell az adatbázis szerkezetének különböző absztrakciós szinten meg-

határozott fogalmakkal és szabályokkal leírt szerkezete. Az adatbázis a modellel meghatározott szerkezetben tárolt, logikailag egymáshoz kapcsolódó egyedek és tulajdonságaik, valamint a tényleges adatokkal együtt tárolódó, továbbá azok szerkezetére és kezelésére vonatkozó kiegészítő metaadatok együttese.

Erősen leegyszerűsítve a modell a szerkezet, az adatbázis a modell alapján szervezett és tárolt adatok halmaza.

Az adatbázis tehát a ténylegesen tárolt és kezelhető adatokat jelenti. Míg a modellek elméletben is megalkothatók, addig az adatbázis fizikailag létező dolog, amelynek kezeléséhez megfelelő szoftveres eszközökre van szükség. A következő szakaszokban az adatbázisok kezelését biztosító alkalmazásokkal foglalkozunk. Áttekintjük azokat a feladatokat, amelyeket egy ilyen rendszernek meg kell valósítania.

6.3 ADATBÁZISOK ANSI-SPARC ARCHITEKTÚRÁJA

Az adatbázisokat különböző feladatokat végző és eltérő informatikai kompetenciákkal rendelkező felhasználók használják. Képzettségüknek és feladataiknak, vagyis az adatokkal általuk végzett műveleteknek megfelelően, eltérően, más-más módon látják, illetve kell látniuk az adatbázist. A fizikailag változatlan formában tárolt adatokat másképpen látja és használja a rendszeradminisztrátor, másképp az adatbázis-tervezéssel, -adminisztrációval, -kezeléssel foglalkozó szakember, és egészen más formában jelenik meg az adatbázis az adatokat a mindennapi munkájához használó, úgynevezett naiv felhasználó előtt. Ez azt is jelenti, hogy az adatbázisokat megjelenítő és kezelő szoftvereknek különböző szintekre, rétegekre kell tagolódniuk. Az egyes szinteket implementáló alkalmazásoknak eltérő formában kell megjeleníteniük az adatbázist, és a szintnek megfelelő műveletek elvégzését kell biztosítaniuk.

Az adatbázis-kezelés különböző absztrakciós szintjeinek sémáját sokféleképpen fölvázolták. Ez első, széles körben elfogadott séma a Charles Bacman által vezetett American National Standards Institute's Standards Planning and Requirements Committee nevű szervezet munkája volt. Az általuk megfogalmazott elképzelés az adatbázis-rendszerek ANSI-SPARC architektúrája néven vált ismertté az informatikai szakemberek előtt.

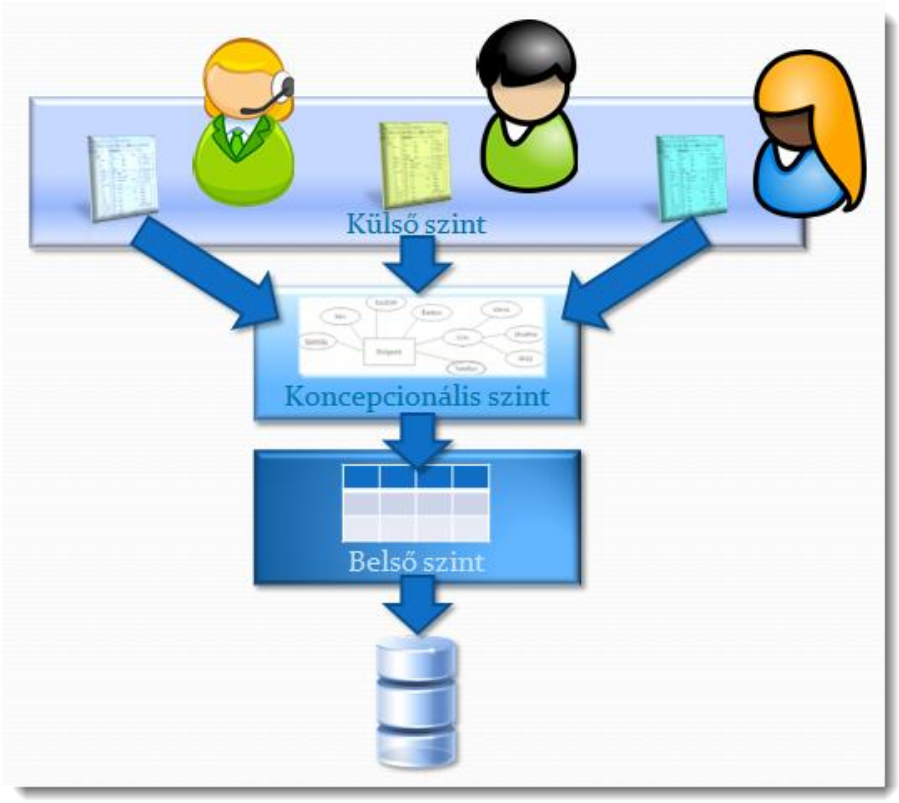
Az ANSI-SPARC architektúra az adatbázisok kezelését három különböző, de egymásra épülő absztrakciós szinten, úgynevezett nézetben képzeli el.

A legalsó szintet **belső szintnek** nevezik. Ebben a nézetben a logikai és fizikai adatmodell fogalmainak megfelelő struktúrában jelenik meg az adatbázis. A szint az adatbázis-menedzserek, adatbázis-adminisztrátorok, és szakértő fel-

használók feladatainak megfelelő formában mutatja és teszi kezelhetővé az adatokat.

A **koncepcionális szinten elérhető** nézet a megismert koncepcionális modellezés fogalmainak megfelelően jeleníti meg az adatszerkezetet. A teljes adatbázis egyedtípusok, tulajdonságtípusok, és kapcsolatok rendszerét alkotja.

A **külső szint** azt a nézetet jelenti, ahogyan az eseti, vagy naiv felhasználó látja az adatbázist. Ő az, aki a munkájával kapcsolatos információk megszerzésére használja az adatokat. Nem látja és nem is akarja látni a tárolás struktúráját, nem érdeklik sem a rekordok, sem pedig a mezők, ő pusztán az információval, a tárolt adatok jelentésével foglakozik. Mivel ugyanazt az adatbázist különböző szerepkörű felhasználók is használják, egyazon az adatbázisnak többféle külső nézete is lehet.



44. ábra Az ANSI-SPARC architektúra

Az ANSI-SPARC architektúra a mi szempontunkból elsősorban azért érdekes, mert nem csupán nézeteket, hanem az egyes szinteken elvégezhető feladatokat is megfogalmazza. Világossá teszi, hogy az adatbázisok kezelésére nem csupán egyféle programot, hanem a három szint feladatait implementáló különálló, ugyanakkor egymással szorosan együttműködő szoftvereket kell használni.

Bár az ANSI-SPARC architektúrát csak viszonylag későn, 1975-ben fogalmazták meg, maga a feladatrendszer természetesen korábban is adott volt. A **feladatokat azonban kezdetben nem bontották rétegekre**, a három szint funkcióinak támogatásra egyetlen szoftvert használtak.

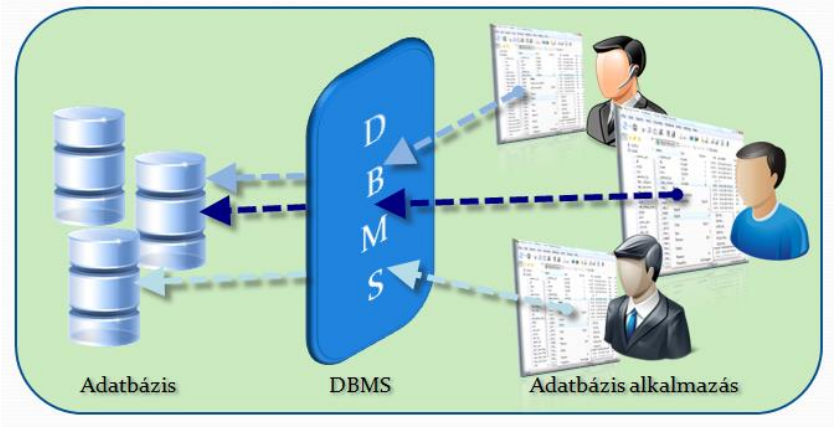
Ezek az alkalmazások olyan adatkezelő programok voltak, amelyek kódja egyaránt meghatározta a kezelt adatok szerkezetét és az adatokkal végezhető művelteket. A **szoftverek** és a velük kezelt **adatbázisok** egyedi fejlesztésűek voltak, és **elválaszthatatlanul kapcsolódtak egymáshoz**. A program csak saját adatbázisát tudta kezelni, az adatbázis csak saját programjával volt elérhető. Ha az adatbázis tárolási módját, fájlszintű hozzáférését, vagy szerkezetét meg kellett változtatni, akkor át kellett írni az alkalmazást is.

Ezek az adatbázis-kezelő alkalmazások még **nem voltak képesek a fizikai és logikai adatfüggetlenség** megvalósítására. **Ez vezetett** háttérbe szorulásukhoz, és az ANSI-SPARC architektúrát tükröző **adatbázis-rendszerek megjelenéséhez**.

6.4 ADATBÁZIS-RENDSZEREK

Az adatbázis rendszerek az ANSI-SPARC architektúra nézeteinek megfelelően három komponensből tevődnek össze:

- A logikailag összetartozó adatokat és az azokhoz kapcsolódó metaadatokat tároló, valamilyen logikai modell alapján strukturált **adatbázisokból**.
- Az adatbázisok létrehozására és manipulálására alkalmas **adatbázis-kezelő rendszerekből** (Database Management System, DBMS).
- A felhasználók munkáját közvetlenül támogató, és a DBMS-sel kommunikáló **adatbázis-alkalmazásokból.**



45. ábra Adatbázis-rendszer

Az adatbázis-kezelő rendszerek tehát azok az alkalmazások, amelyek lehetővé teszik az adatbázisok kezelését. Számos egyéb jellemzőjük közül két kiemelkedően fontos tulajdonságuk, hogy a fizikai és logikai adatfüggetlenséget megvalósítva biztosítják az adatbázisok elérését.

6.4.1 Fizikai és logikai adatfüggetlenség

A **fizikai adatfüggetlenség** azt jelenti, hogy az adatok tárolásának, hozzáférési módjának megváltozása nem teszi szükségessé az adatbázis-kezelő rendszer és az adatbázis-alkalmazások megváltoztatását.

A **logikai adatfüggetlenség** hozadéka az, hogy egy adatbázis szerkezetének megváltoztatása, például új egyedtípusok, tulajdonságtípusok beiktatása nem teszi szükségessé az adatbázis-kezelő rendszer változtatását.

A DBMS-ek tehát eltakarják a felhasználók, és az alkalmazások elől az adatbázisok fájlszintű kezelését, és valamilyen logikai adatmodellnek megfelelően teszik lehetővé azok kezelését. Ebből természetesen következik, hogy minden **adatbázis-kezelő rendszer szoros kapcsolatban van valamelyik logikai adatmodellel.** Sőt, azt is mondhatjuk, hogy minden DBMS egy logikai adatmodell műveletei részének megvalósítása, implementációja. Ezért beszélünk hierarchikus, hálós, objektum-orientált illetve **relációs adatbázis-kezelő rendszerekről**.

6.4.2 Adatbázis-kezelő rendszerek feladatai

A DBMS tehát az a szoftver, amely lehetővé teszi, hogy az adatokat fizikailag tárolódó adatbázis belső szerkezetét a logikai adatmodellnek megfelelő struktúrában lássuk és kezeljük.

Az adatbázis-kezelő rendszerek feladatrendszere természetesen jóval sokrétűbb, mint az eddig felvázoltak. Nézzük most a legfontosabb feladatok felsorolását.

- Az adatbázis-kezelő rendszereknek lehetővé kell tenniük egy bizonyos logikai adatmodellnek megfelelő szerkezetű adatbázisok létrehozását, tulajdonságaik beállítását, esetleges törlését,
- Az adatbázisok belső struktúrájának kialakítását, változtatását.
- Biztosítaniuk kell az adatok perzisztens tárolását.
- A folyamatos hozzáférést, az adatok lekérdezésének lehetőségét.
- Rendelkezniük kell a felhasználók azonosítására, jogosultságaik leírására, nyilvántartására, és ellenőrzésére alkalmas jogosultsági rendszerrel.
- Biztosítaniuk kell az adatok konkurens, távoli elérését, a tranzakciók kezelését.
- Az adatkonzisztencia párhuzamos hozzáférés melletti megőrzését.
- Rendelkezniük kell az adatok biztonsági mentését és a konzisztens állapot visszaállítását biztosító alrendszerrel.
- Biztosítaniuk kell, hogy a felhasználók, illetve az adatbázis-alkalmazások valamilyen adatkezelő nyelv segítségével fogalmazhassák meg az adatbázisok kezelésére és a DBMS irányítására vonatkozó utasításaikat.

6.5 KLIENS-SZERVER ARCHITEKTÚRA

A hálózatok elterjedése az adatbázis-rendszerekkel szemben támasztott követelményekre is meghatározóan hatott. A párhuzamos és távoli hozzáférés biztosítása érdekében a modern adatbázis-rendszerek fejlesztése fokozatosan a kliens-szerver architektúrák felé fordult. A kliens-szerver architektúra olyan szoftverstruktúra, amelyben egy bonyolult feladatrendszer folyamatait egymástól fizikailag is elkülönülő, de a feladat megoldásában együttműködő üzenetalapú kommunikációt folytató szoftverimplementációk, a kliens és a szerver valósítják meg.

A kliens feladatai

A kliens legfontosabb feladata a felhasználói felület biztosítása, a felhasználótól származó adatok fogadása, a rendszer üzeneteinek megjelenítése. A kialakításától függően több vagy kevesebb feldolgozó műveletet is végző kliens valamilyen részfeladat megoldásában mindig a szerverre támaszkodik. A kliens a szerverrel való kapcsolatfölvételben aktív, üzenetet, kérést küld a szervernek, majd földolgozza a kapott választ. Egy kliens jellemzően a felhasználó lokális számítógépén fut, és egy időben egyetlen szerverhez kapcsolódik. Működésére az időszakosság jellemző, azaz nem fut folyamatosan, a felhasználó indítja el, és állítja le.

A szerver tevékenysége

A szerver folyamatosan figyeli a kliensek felől érkező üzenteket. Ha kérést kap, feldolgozza az üzenetet, végrehajtja az ahhoz kapcsolódó művelteket, majd válaszüzenetet küld a kliens felé.

Általában nem a felhasználó gépén, hanem egy „távoli” gépen fut, és egy időben több kliens kéréseit is képes kiszolgálni. Működése általában folyamatos, bármely pillanatban alkalmas a kliensek kiszolgálására.

6.5.1 Kétrétegű kliens-szerver architektúra

A hálózati környezetben megvalósított adatbázis-rendszerek kezdetben kétrétegű kliens szerver architektúra alapján működtek.

- Ebben az architektúrában a szerver rétegében helyezkedik el az adatbázis-kezelő rendszer – amit éppen ezért gyakran említenek adatbázis-szerverként is –, a kliens rétegében találjuk az adatbázis-alkalmazást. Az adatbázis-alkalmazás biztosítja a DBMS által küldött adatok megjelenítését, az adatokkal végezhető különböző számítási műveletek elvégzését, és a felhasználóval való kommunikációt. A DBMS az adatbázis és az adatbázis-alkalmazás kapcsolatáról, az alkalmazás által küldött adatok szerveroldali ellenőrzéséről, tárolásáról, a lekérdezett adatok kiválasztásáról és visszaküldéséről, a különböző adatkezelő műveletek adatbázisszintű lebonyolításáról gondoskodik.

Ez a megoldás valójában az ANSI-SPARC architektúra hálózati környezetben történő megvalósítása. Biztosítja a felhasználók távoli és párhuzamos adathozzáférését, növeli az adatfeldolgozás hatékonyságát, csökkenti a hardver- és szoftverköltségeket, hiszen a hardver- és szoftvererőforrások költségesebb elemei a szerverhostra koncentrálódnak.

6.5.2 Háromrétegű kliens-szerver architektúra

Míg a kétrétegű architektúrában az adatbázis-alkalmazás a kliens gépen fut, háromrétegű architektúrában ez a folyamat további két rétegre bomlik. A felhasználó gépén csak egy úgynevezett vékonykliens működik, amely gyakorlatilag csupán a felhasználói felület megjelenítését, és kezelését teszi lehetővé. Az adatbázis-alkalmazás az alkalmazásszerver szolgáltatásaként egy távoli gépen fut. A felhasználó számítógépén csak az alkalmazás kimeneti adatai és az azok kezeléséhez szükséges felület jelenik meg. Ezzel a megoldással tovább csökkenthetők a költségek, egyszerűbbé válik az adatbázis-alkalmazások telepítése, karbantartása, és esetleges továbbfejlesztése.

6.5.3 'n' rétegű kliens-szerver architektúra

Az adatbázis-rendszerekkel végzett feladatok természetesen további rétegekre bonthatók, így a kliens-szerver architektúra háromnál több rétegű is lehet. A napjainkban egyre nagyobb teret hódító webes alkalmazások legalább négyrétegű architektúrát használnak.

A kliens rétegben csupán a felhasználói felületet megjelenítését és kezelését biztosító vékonykliens található, ami egyébként a webböngészőnek felel meg.

Magát a felületet a webszerver állítja elő és küldi el a kliensnek. A feldolgozó folyamatokat az alkalmazás szerver futtatja, amely szükség esetén az adatbázis-kezelő rendszerrel végezteti el az adattárolással és lekérdezéssel kapcsolatos feladatokat.

Ez a megoldás tovább növeli a rugalmasságot, jobban skálázható, modulárisan fejleszthető, és a vékonykliensek alkalmazása miatt lehetővé teszi az adatbázisok mobil eszközökkel történő elérését is.

6.6 ÖSSZEFOGLALÁS, KÉRDÉSEK

6.6.1 Összefoglalás

Mai leckénkben az adatbázis-rendszerek fölépítéséről tanultunk. Elsőként különbséget tettünk az adatmodell és adatbázis között. Ezt követően megismerkedtünk az adatbázis-rendszerek három szintű ANSI-SPARC architektúrájával, majd részletesen megvizsgáltuk az adatbázis-rendszerek összetevőit, az adatbázist, az adatbázis-kezelő rendszert és az adatbázis-alkalmazást.

Leckénk utolsó szakaszában az adatbázis-rendszerek jellemző hálózati architektúráiról tanultunk. Megismertük a kliens- és szerverszintekből álló kétréte-

gű architektúrát, az alkalmazás-szerverrel kiegészített háromrétegű architektúrát, és a webszervert is tartalmazó négyrétegű architektúrát.

6.6.2 Önellenőrző kérdések

1. Mi a különbség adatmodell és adatbázis között?
    - Az adatmodell határozza meg az adatbázis szerkezetét, az adatbázis pedig az adatmodellel leírt struktúrában tárolt adatok összessége.
2. Miért van fontos szerepük az adatbázis-kezelő rendszereknek a fizikai modellezésben?
    - Magában a modellezésben annyi szerepük van a DBMS-eknek, hogy a modellezéshez az adatbázis-kezelő rendszer által értelmezhető formalizmust kell használni. A valódi szerepük a modell értelmezésében, fizikai adatbázissá alakításában van.
3. Az ANSI-SPARC architektúra mely szintje felel meg az adatbázis logikai és fizikai modelljének?
    - A belső szint. Ezen a szinten a fizikai és logikai modellnek megfelelő szerkezetben látjuk az adatbázist.
4. Milyen összetevői vannak az adatbázis-rendszereknek?
    - Adatbázis, adatbázis-kezelő rendszer, adatbázis-alkalmazás.
5. Milyen hálózati architektúra jellemzi az adatbázis háttéren alapuló webes alkalmazásokat?
    - Az adatbázis-alapú webes alkalmazások legalább négyrétegű architektúra szerin működnek: kliens, webszerver, alkalmazásszerver, adatbázis-kezelő rendszer.

7. LECKE: AZ SQL NYELV ÉS NYELVJÁRÁSAI, A MYSQL

7.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

Az előző leckék során többször elhangzott, hogy a fizikai adatmodellek már alkalmasak arra, hogy reprezentációjuk alapján létrehozzuk az adatok fogadására alkalmas adatbázist. Ehhez egyrészt szükség van arra, hogy a modell tartalmazza a fizikai adatbázist alkotó strukturális elemeket, másrészt léteznie kell olyan formalizmusnak, amellyel a modell az adatbázis-kezelő rendszerek számára értelmezhető formában ábrázolható.

A relációs adatbázisok fizikai modelljének ábrázolására a Data Definition Language (DDL) nevű adatstruktúra-leíró nyelv alkalmas. A DDL olyan nyelvi elemeket és szintaktikai szabályokat tartalmaz, amelyekkel a relációs adatbázisok minden strukturális eleme megnevezhető, és szerkezetük egyértelműen leírható. A DDL a relációs adatbázisok napjainkra szabvánnyá vált adatkezelő nyelve, a Structured Query Language (SQL) egyik, úgynevezett résznyelve. A Structured Query Language szabványosságának köszönhetően széles körben támogatott, így az adatbázis-kezelő rendszerek szinte mindegyike képes értelmezni és végrehajtani az SQL-ben megfogalmazott adatkezelő utasításokat, az úgynevezett SQL-mondatokat.

Miután az tananyagunk előző részében általánosan fogalmaztuk meg az adatbázis-kezelő rendszerek jellemzőit, most megismerkedünk a vezérlésükre használt SQL-lel, és az SQL-utasítások szintaxisának jelölésével. Az adatkezelő nyelv ismerte kipróbálás nélkül semmit sem érne. Tananyagunk hátralévő részében már elengedhetetlen, hogy egy nagy teljesítményű, professzionális relációs adatbázis-kezelő rendszer is rendelkezésünkre álljon. Leckénk befejező részében napjaink egyik legnépszerűbb relációs adatbázis-kezelő rendszerével, a GPL licencelésű MySQL-lel ismerkedhetünk meg.

A tananyag tanulmányozása során szenteljen kiemelkedő figyelmet az SQL-utasítások szintaktikai leírásának. A hátralévő leckékben ugyanis ezt a formalizmust használjuk az egyes SQL-parancsok lehetséges használati módjának leírására.

Az SQL-mondatok megfogalmazása mellet szintén nagyon fontos, hogy képes legyen azokat továbbítani a DBMS felé. Ezért ne haladjon tovább mindaddig, amíg a MySQL parancssori kliensének megismerése és kipróbálása után kapcsolatba nem tud lépni a MySQL-szerverrel.

7.2 A FIZIKAI MODELL STRUKTURÁLIS ELEMEI

Ahogyan a koncepcionális modell strukturális elmei megjelennek a logikai adatmodellekben, úgy a logikai modell szerkezeti elmei is föllelhetők a fizikai modellekben. Minden olyan strukturális elem, amelyet a relációs adatmodellben megtanultunk, használható és ábrázolható a fizikai modell leírására alkalmas SQL nyelvben.

Rövidesen látni fogjuk, hogyan hivatkozhatunk adatbázisra, táblára, mezőre, domainre, hogyan hozhatunk létre elsődleges és idegen kulcsokat, indexeket, hogyan biztosíthatjuk a hivatkozási integritás megőrzését, milyen lehetőségeink vannak az adatbázisok olyan DBMS-specifikus tulajdonságainak a szabályozására, mint a táblák fizikai megvalósítása, a karakterkódolás, vagy a szöveg összehasonlítás módjának beállítása.

7.3 AZ SQL

Bár fizikai modellezés az SQL DDL résznyelvével történik, és tananyagunkban szinte csak ezzel a résznyelvvel foglakozunk, röviden mégis célszerű megismerkedünk az nyelv egészének történetével, általános szintaxisával és további résznyelveivel.

7.3.1 AZ SQL nyelv története

Minden bizonnyal emlékszünk rá, hogy a relációs adatmodellt Edgar Frank Codd, az IBM munkatársaként dolgozta ki. A System R projekt fejlesztéssel foglalkozó programozói csoportja az 1970-es évek elején létrehozta a **SEQUEL** (Structured English Query Language) nevű, relációs adatbázisok kezelésére, illetve az adatbázis-kezelő rendszer vezérlésére alkalmas nyelvet.

Az English jelző arra utalt, hogy a **nyelv szavai értelmes**, az **adatkezelő** műveletek jelentését kifejező **angol szavakból** állnak. A szavak felhasználásával pontosan definiált nyelvi szabályok alapján mondatok, az adatkezelő műveleteket precízen leíró utasítások voltak létrehozhatók.

A nyelv nem illeszkedett a hagyományos procedurális programozási nyelvek közé. Sokkal **inkább hasonlított a deklaratív nyelvekhez**, ugyanis az adatkezelő feladatok végrehatási módjának leírása helyett a végrajtástól várt eredmény megfogalmazására volt alkalmas. Az adatbázis kezelőjének tehát nem azt kellet leírnia, hogy hogyan kell megoldani a feladatot, hanem azt, hogy mit szeretne látni az adatbázisból.

Codd egy másik, az ALPHA nevet viselő adatkezelő nyelv fejlesztését javasolta, és nem pusztán ellenezte, de egyenesen a relációs modell hibás imple-

mentációjának vélte a SEQUEL-t. A kialakult nézetkülönbségek miatt Codd később ki is lépett az IBM-től, de a SEQUEL fejlesztése tovább folytatódott a cégnél. A nevet később SQL-re (Structured Query Language), változtatták, ami utal arra, hogy nyelv egyik legkiemelkedőbb erőssége az adatbázisokban tárolt adatok kiválogatásában, lekérdezésében rejlik.

7.3.2 Nyelvjárások kialakulása

Az IBM akkori üzletpolitikájából adódott, hogy az SQL mindig is mostohagyerek maradt a cégnél. A konkurensek termékei azonban egyre nagyobb számban támogatták, így az **SQL lassacskán kváziszabvánnyá** fejlődött. Később az Amerikai Nemzeti Szabványügyi Intézet, az **ANSI** (American National Standards Institute) **hivatalos szabvánnyá** minősítette. Ezt követően jelentek meg ISO különböző SQL szabványai (SQL86, SQL89, SQL92, SQL99) is.

A nyelv ennek köszönhetően nemcsak fennmaradt, de **standardizálódott** is. Ugyanakkor – ahogyan az az informatikában egyáltalán nem ritka jelenség a különböző gyártók implementációi között – kisebb-nagyobb eltérések alakultak ki a különböző változatok között. Ennek köszönhetően létrejöttek az SQL-szabványtól kissé eltérő gyártó-, illetve **DBMS-specifikus változati**, az úgynevezett SQL-**nyelvjárások**. Ennek negatív hozadéka, hogy az egyes nyelvjárások szerint megfogalmazott adatkezelő utasítások nem teljesen kompatibilisek egymással.

A nyelvjárások közötti különbségek – az adatbázis-kezelő rendszer vezérlésére alkalmas DCL résznyelvet nem számítva – sajnos leginkább éppen a fizikai modellek leírására alkalmas DDL nyelvet érintik. Tananyagunkban a MySQL 5.5 verziójának megfelelő nyelvtani formákat ismertetjük.

7.4 AZ SQL NYELV RÉSZEI

A Structured Query Language elnevezés azt sejteti, hogy a nyelv elsősorban lekérdezések készítésére, az adatbázisban tárolt adatok kinyerésére alkalmas. Ezzel szemben az SQL a relációs adatbázisok kezelésével kapcsolatos minden művelet leírására használható. A parancsokkal elvégezhető művelet típusa alapján **SQL-mondatokat négy csoportba**, úgynevezett **résznyelvekbe** soroljuk.

7.4.1 SQL DDL

A DDL (Data Definition Language, adatdefiníciós nyelv) a fizikai adatmodellek formális ábrázolására alkalmas. Az SQL azon nyelvi elemeit tartalmazza, amelyekkel a fizikai adatmodell szerkezeti elemeiből alkotott struktúrák írhatók le. Az adatbázis modellezője a DDL felhasználásával pontosan definiálhatja az

adatbázis, illetve az azt fölépítő objektumok szerkezetét és felépítését, az adatbázis-kezelő rendszer pedig a DDL-mondatok értelmezésével képes az adatszerkezet létrehozására.

A DDL-mondatokkal adatbázisok, táblák, mezők, indexek és kulcsok hozhatók létre, de a résznyelv természetesen a létrehozott struktúrák megváltoztatására és törlésére is biztosít nyelvi eszközöket.

7.4.2 SQL DML

A DML (Data Manipulation Language, adatmanipulációs nyelv) az SQL adatmanipulációs résznyelve. A DML-mondatokkal rekordokat illeszthetünk a létrehozott adatbázis tábláiba. Módosíthatjuk a tárolt adatokat, vagy törölhetjük a fölöslegessé vált rekordokat.

7.4.3 SQL DCL

A DCL (Data Control Language: adatvezérlő nyelv) résznyelvbe sorolható nyelvi elemek nem kötődnek szorosan a relációs adatmodellhez, de az adatbázisok kezeléséhez, illetve az adatbázis-kezelő rendszer vezérléséhez nélkülözhetetlenek.

A DCL-mondatokkal nem adatbázisban tárolt adtok közvetlen kezelését, hanem az adatbázis-kezelő rendszer működését szabályozhatjuk.

7.4.4 SQL DQL

A DQL (Data Query Language, adatlekérdező nyelv) az SQL névadó résznyelve. A legtöbb felhasználó az SQL DQL résznyelvével találkozik, hiszen az DQL-mondatok segítségével tudjuk kinyerni az adatbázisokban tárolt adatokat.

Használatával az adatbázisok tábláiban tárolt adatok különböző szempontok és összefüggések alapján kérdezhetők le. A DQL-ben pontosan megfogalmazhatjuk a megjelenítendő rekordok és mezők forrásául szolgáló táblákat, vagy táblakapcsolatokat, és feltételekkel szűrhetjük a rekordokat. A kiválasztott adatokkal statisztikai műveleteket végezhetünk, vagy egyszerűen megjeleníthetjük őket. Az SQL DML résznyelve teszi lehetővé, hogy az adatbázisokban tárolt adatokhoz hozzáférjünk, és azok értelmezésével információhoz jussunk. Egyes osztályozások DQL-t a DML részének tekintik, így számos irodalomban nem is találkozunk külön ezzel a résznyelvvel.

7.5 AZ SQL FORMÁLIS SZABÁLYAI

Ahogyan korábban említettük, az SQL nem tartozik a hagyományos értelemben vett programozási nyelvek közé, azaz nem procedurális nyelv. Nem tartalmaz vezérlési szerkezeteket, nem támogatja a felhasználói interakciót és a fájlkezelést, továbbá csak korlátozottan és nyelvjárástól függően engedi a változók használatát, illetve az eljárások, illetve függvények deklarálását.

A deklaratív nyelvekkel azért hozható összefüggésbe, mert nem támogatja, és **nem is teszi szükségessé a feladat megoldásának leírását**. Az SQL-ben a forrásadatokat, és az azok feldolgozása nyomán várt eredményt kell a lehető legpontosabban leírnunk. Az **eredmény kialakítása**, a feladat elvégzése az **adatbázis-kezelő rendszer dolga**.

A nyelv logikus fölépítésű, és gyorsan megérthető. Az SQL-mondatok mindig a műveletet meghatározó **paranccsal kezdődnek**, és **pontosvesszővel** (;) fejeződnek be. A parancsot, meghatározott **sorrendben** különböző **kötelező és opcionális elemek** követhetik:

- az SQL **fenntartott szavai** (kulcsszavak),
- tábla- és mező**hivatkozások**,
- értékek, **literálok**,
- matematikai, szöveges és logikai **kifejezések**,
- a kifejezésekbe írt **függvényhivatkozások**.

Az SQL-mondatok megfogalmazását csak az nehezítheti, hogy **a parancsok** a várt eredmény minél pontosabb megfogalmazhatósága érdekében **számos opcionális nyelvi elemmel** egészíthetők ki. Ez flexibilitás – a nyelv egyébként pontosan meghatározott nyelvtana ellenére – a műveletek árnyalt, apró részletekre is kiterjedő megfogalmazását teszi lehetővé az adatbázis kezelője számára.

A nagyszámú opcionális elem nagy szabadságot biztosít ugyan a feladatok leírásában, de megnehezíti a parancsok nyelvtani szabályainak ábrázolását. Ezért mielőtt a tényleges parancsok bemutatására rátérnénk, meg kell ismernünk a nyelvtani szabályok leírására használt formalizmusokat.

- **NAGYBETŰS** szavak: az SQL nem érzékeny a kis- és nagybetűk közötti különbségekre. A formai leírásokban használt nagybetűs szavak olyan nyelvi szavakat jelölnek, amelyeket pontosan a leírásban megadott módon, **változtatás nélkül lehet csak használni**. Ezek mindig az SQL saját kulcsszavai. A mondatokat állítmányaként felfogható **parancsok**, a parancsokat kiegészítő paraméterek megadására alkalmas **záradékok**, és az azokat kiegészítő egyéb **fenntartott szavak alkothatják**.

- **Dőlt betűk**: a nagybetűvel írt szavakkal szemben a dőlt betűs szövegek nem megadott formában kerülnek az SQL mondatokba. **Helyükre** ismert, vagy a leírás későbbi részben részletezett **nyelvi elemeket** kell **behelyettesíteni**.
- **[szögletes zárójelben megadott szöveg]**: Mint azt rövidesen tapasztaljuk, az SQL-mondatokban általában meglehetősen kevés a kötelező nyelvi elem. Ugyanakkor szinte mindegyik parancsot opcionálisan használható, elhagyható nyelvi elemek egész sora egészíti ki. Ezek közül mindig csak azokat kell használni, amelyek a kívánt eredmény pontos leírásához szükségesek.

  Az **elhagyható nyelvi elemek szögletes zárójelek között** szerepelnek.

  Ez egyben azt is jelenti, hogy **ha egy parancs,** vagy paraméter **nincsen szögletes zárójelben,** akkor használata **kötelező**!
- **| függőleges vonal** (cső karakter): Az egyes nyelvi elemek között elhelyezve azt jelzi, hogy azok közül egyet, és csak egyet választanunk kell SQL-mondat megfogalmazásakor.
- **Nyelvi elemek sorrendje**: Mint említettük az egyes SQL parancsokhoz számos opcionálisan használható elem kapcsolódik. Ezek használatáról a cél ismertében dönthetünk. A használt nyelvi elemek sorrendje azonban nem felcserélhető, mindig a formális leírást követi.
- **Kis- és nagybetűk**: Mint tudjuk, a szintaktikai leírásokban az SQL nyelv fenntartott szavait nagybetűkkel tűntetjük fel. A jobb olvashatóság érdekében a tényleges SQL mondatokban is szokás a kulcsszavak nagybetűs leírása, ez azonban nem kötelező. Az adatbázis-kezelő rendszerek nem különböztetik meg a kis-és nagybetűket. A DBMS számára a **`select`** ugyanazt jelenti, mint a **`SELECT`**. A fájlrendszerben is megjelenő adatbázis elemek – mint például a MySQL+adatbázisok a táblái – neveiben azonban a DBMS-t futtató gép operációs rendszere határozza meg a kis- és nagybetű érzékenységet. A Unix-rendszereken futó MySQL esetében van különbség egy táblanév kis- és nagybetűs változata között, Windows operációs rendszer esetében nincs.
- **Szövegliterálok:** A szövegliterálokat az SQL-ben – de az informatikában általában is – egyszeres, vagy dupla aposztrófok közé zárjuk.

```
SELECT 'szöveg')
v.
SELECT ("szöveg")
```

- **{kapcsos zárójelbe zárt szöveg}: A kapcsos vagy francia zárójelek közé zárt szöveg logikai egységet alkotó nyelvi elemeket jelöl. A tényleges SQL-mondatokban nem szabad szerepeltetni a zárójeleket.**
- **Sortörés**: Az SQL mondatok szavait egy vagy több közkarakterrel (szóköz, tabulátor, sorvég) kell elválasztani. A mondatokat tehát a szavak határán tetszőleges számú sorra tördelhetjük, a sorokban pedig további közkarakterekkel tagolhatjuk a szöveget. Az egynél több egymást követő közkaraktert egyszerűen figyelmen kívül hagyja DBMS. Ez alkalmas a mondatok olvashatóságának fokozására, ami pedig sokat segíthet az esetleges hibák felkutatásában. A hosszabb mondatok ugyanis sokkal könnyebben átláthatók és megérthetők, ha több sorba tördelve, és a sorokban behúzásokat elhelyezve gépeljük be őket. Az egyes parancsok formális leírásaiban is alkalmazzuk ezeket a sortöréseket, de használatkor szabadon dönthetünk alkalmazásukról.

Az alábbi példa egy egyszerű DCL-parancson keresztül mutatja be a formális leírás értelmezését. A parancs a felhasználók jelszavának megváltoztatására alkalmas.

```
SET PASSWORD [FOR user] =
   PASSWORD('jelszó')
   | OLD_PASSWORD('jelszó')
   | 'kódolt jelszó'
```

A fenti szintaktikai leírást az alábbiak szerint értelmezhetjük.

A jelszó megadásakor kötelező használni a SET és PASSWORD kulcsszavalat, amelyeket kötelezően egyenlőségjel követ.

Ezután három lehetőség közül az egyiket kell használnunk.

*A **PASSWORD** függvény paraméterként szövegliterál formájában adjuk meg az új jelszót,*

vagy

*az **OLD_PASSWORD** függvényt használjuk hasonló módon,*

vagy

közvetlenül a kódolt jelszót adjuk meg literálként.

*A **FOR** user elhagyható. Ha megadjuk a FOR kulcsszót, akkor utána a megfelelő felhasználó azonosítójának kell szerepelnie. A **FOR** user használatával egy pontosan megadott felhasználó, elhagyásával saját magunk jelszavát változtatjuk meg.*

7.6 A MYSQL

Az adatbázis-kezelő rendszerek funkcióiról az előző leckében tanultunk. Tudjuk, hogy minden DBMS biztosítja az adatbázisok létrehozását, szerkezetük kialakítását, a felhasználók, és felhasználói jogosultságok kezelését, az adatok manipulálását, lekérdezését, és számos egyéb, az adatkezeléshez szükséges művelet lebonyolítását.

Napjainkban jónéhány kereskedelmi és nyílt forráskódú adatbázis-kezelő rendszer létezik. Széles körben ismertek például az Oracle Database, a Microsoft MS-SQL valamint az IBM Informix nevű rendszerek, illetve a nyílt forráskódú postgreSQL, az SQLite, vagy a MySQL. Tananyagunk hátralévő leckéiben a MySQL segítségével végezzük az adatbázisok fizikai modellezését.

Bár a MySQL GPL licencelés alá tartozó, ingyenesen használható szoftver, teljesítményében és tudásában fölveszi a versenyt számos nagy, kereskedelmi forgalomban kapható adatbázis-kezelő rendszerrel. A MySQL-t a svéd MySQL AB cég fejlesztette, azóta azonban több tulajdonos kezében is megfordult, 2010 januárja óta pedig az Oracle tulajdonában van.

Számos platformon, többek között Windows, MAC OS X, Linux és Solaris operációs rendszereken is használható. Ingyenes használatának, kiváló teljesít-

ményének, illetve a PHP-integrációjának köszönhetően a webes alkalmazások támogatására talán legelterjedtebben használt adatbázis-kezelő rendszer.

A szoftver különböző platformokon futó változatai letölthetők az alábbi címről:

http://dev.mysql.com/downloads/mysql/

2. link MySQL DBMS letöltése

7.6.1 MySQL szerver és kliens

A MySQL adatbázis-kezelő rendszer szerveralkalmazásként fut. TCP/IP-hálózatokban alapértelmezésben a 3306-os porton figyeli a bejövő kapcsolatokat, és folyamatosan feldolgozza a lokális, vagy távoli gépről érkező kéréseket.

A szerver valamilyen kliensprogram, adatbázis-alkalmazás által küldött SQL nyelven megfogalmazott utasításokkal vezérelhető. Legegyszerűbb, ha a MySQL programcsomag részét képező parancssori klienst használhatjuk ügyfélként.

Ez a karakteres felületet biztosító ügyfélprogram – platformtól függetlenül – mindig rendelkezésre áll a MySQL-szervert futtató gépen, a szerver programkönyvtárának **`bin`** mappájában, ezért használatát mindenképpen célszerű megismerni.

A **`mysql`** nevű kliensprogram indításakor egy sereg opcionális paramétert adhatunk meg. Az egybetűs (-u) paraméterek egy, a szó formátumúak (--help) két kötőjellel kezdődnek.

Az összes használható paramétert megjeleníthetjük alábbi formájában kiadott a paranccsal:

```
mysql --help --verbose
```

A leggyakrabban használt paraméterek a következők:

- A szerverhez kapcsolódó felhasználó nevének megadása:
  `-u` *felhasználó*

- Jelszóhasználat jelzése (akkor használjuk, ha van beállított jelszavunk):
  -p
- Az adatbázis-kezelő rendszert futtató gép címe (csak akkor használjuk, ha a szerver távoli gépen fut):
  -h *IP_cím*
- A szerver által figyelt port száma (akkor alkalmazzuk, ha a DBMS-t úgy konfigurálták, hogy az ne az alapértelmezett 3306-os portot használja):

-**P** *port*

Az alábbi példában adatbázis adminisztrátorként jelentkezünk be a 10.4.2.5 IP-című gépen található adatbázis-szerverre, amely a 3302-es porton várja a kapcsolatokat.

```
>mysql –u root –p –h 10.4.2.5 –P 3302
```

Az alábbi módon indított kliens a localhoston futó, alapértelmezett portot figyelő szerverre jelentkezik be:

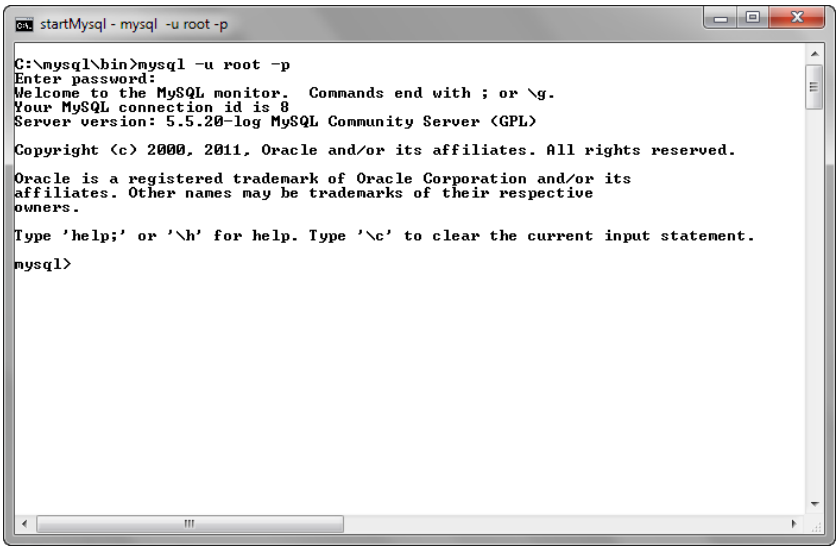
```
>mysql –u root –p
```

A jelszó begépelése után a kliens elküldi a felhasználói adatokat a szervernek, ami a jogosultságok ellenőrzése után engedélyezi, vagy hibaüzenet küldésével elutasítja a kapcsolatot:

```
>mysql -u nincs -p
Enter password:
ERROR 1045 (28000): A(z) 'nincs'@'localhost'
felhasznalo
szamara tiltott eleres. (Hasznalja a jelszot: NEM)

>
```

A sikeres bejelentkezést az üdvözlő szöveg, és a kliens készenléti jelének megjelenése jelzi.



46. ábra A MySQL kliense

A készenléti jel mellé írt SQL-mondatok formájában megfogalmazott parancsainkat pontosvesszővel kell zárni. Az ENTER billentyű leütésekor kliens az adatbázis-kezelő rendszer felé továbbítja az SQL-mondatot. A MySQL-szerver feldolgozza az utasítást, a válaszüzenet a kliens felületén jelenik meg.

Az ügyfélprogram működése az indításkor megadható opciók mellett a készenléti jelnél begépelhető egyszerű, vissza per (****) jellel bevezetett direktívákkal is szabályozható.

Ilyen például a **\h** direktíva, amelynek hatására a további lehetőségeket tartalmazó súgó jelenik meg.

> Az SQL mondatokat begépelhetjük és futtathatjuk egyenként, de arra is van lehetőség, hogy több mondatot egy szövegállományban, SQL-szkriptben tároljunk, és kötegelten hajtsunk végre. Ennek különösen az adatbázis fizikai modell alapján történő létrehozáskor lehet jelentősége.
>
> A kötegelt végrehajtáshoz a **\.** direktíva után kell megadni az SQL-mondatokat tartalmazó szkript teljes útvonalát.

A kliensből **\q** direktívával, vagy a **quit** parancs begépelésével léphetünk ki.

7.7 ÖSSZEFOGLALÁS, KÉRDÉSEK

7.7.1 Összefoglalás

Tananyagunk mostani leckéjében a relációs adatbázisok fizikai modellezéséhez, illetve a modell ábrázolásához szükséges SQL nyelvről tanultunk. Olvashattunk az SQL kialakulásának történetéről, az SQL-nyelvjárásokról, és megismertük az SQL résznyelveit. Megtanultuk, hogy az SQL filozófiája a deklaratív programozási nyelvekéhez hasonló. Utasításainkban nem az adatbázis-kezelő feladat végrehajtásának módját, hanem a kívánt eredményt kell pontosan megfogalmazni.

A leckében megismertük azt a formalizmust, amelyet a tananyag további részében az SQL-mondatok nyelvtani szerkezetének leírására fogunk használni. Ezt követően esett szó a MySQL adatbázis-kezelő rendszerről. Az eredetileg svéd fejlesztésű szoftvernek már több tulajdonosa is volt, míg 2010-ben az Oracle birtokába került. A kezdeti aggodalmakra rácáfolva a szoftveróriás jelentős fejlesztéseket hajtott végre az adatbázis-kezelő rendszeren. A MySQL – úgy tűnik – jó úton halad afelé, hogy méltó vetélytársává váljon olyan nagynevű ellenfeleknek, mint például a Microsoft SQL Server.

A MySQL-szerver elérése legegyszerűbben a DMBS parancssori kliensével lehetséges, amely a szerver programkönyvtárának **bin** mappájában található, és a **mysql** paranccsal indítható.

A kliens felületén begépelt parancsok a szerverhez kerülnek, válaszok pedig ismét az ügyfélprogram felületén jelennek meg.

A **mysql** alkalmazás működését az indításkor megadott, vagy a program készenléti jele mellett begépelt opciókkal szabályozhatjuk.

7.7.2 Önellenőrző kérdések

1. Az SQL nyelv melyik résznyelve alkalmas az adatbázisok fizikai modelljének ábrázolására?
   - A DDL.
2. Miért mondjuk, hogy az SQL a deklaratív programozási nyelvek filozófiáját követi?
   - Azért mert az SQL mondatokban egy művelet várt eredményét kell megfogalmazni.
3. Hogyan jelöljük a kötelezően használandó elemeket az SQL-parancsok formai leírásában?

- Az ilyen elemeket nem zárjuk szögletes zárójelek közé.

4. Mit tud elmondani az SQL nyelv kis- és nagybetű érzékenységéről?
   - Az SQL-mondatokban általában nincs különbség a kis- és nagybetűk között, azonban a fájlrendszerben is megjelenő objektumok neveiben az operációs rendszer szabályai a meghatározók.
5. Hogyan indítja a MySQL parancssori kliensét, ha a DBMS a 10.4.3.2 IP című hoston fut, a felhasználó neve 'designer', és van beállított jelszava?
   - mysql –u designer –p –h 10.4.3.2

8. LECKE: AZ ADATBÁZIS ÉS A TÁBLÁK LÉTREHOZÁSA

8.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

A fizikai modell valójában az adatbázist fizikailag alkotó strukturális elemek, és azok tulajdonságainak formális leírása. A modellezés az SQLnyelv DDL résznyelvének **`CREATE`** utasításával történhet. A paranccsal nemcsak megfogalmazhatjuk az adatbázis szerkezeti elemeinek jellemzőit, de egyben utasíthatjuk is az adatbázis-kezelő rendszert a struktúra létrehozására. Mivel minden elem a **`CREATE`**-tel írható le és hozható létre, a parancs meglehetősen flexibilis, több változata, és a változatokon belül is számos formája létezik.

Ebben a leckében az új adatbázis létrehozására alkalmas, és a táblák szerkezetét definiáló **`CREATE DATABASE`**, és **`CRATE TABLE`** parancsokkal foglalkozunk.

A lecke végére képes lesz saját adatbázis létrehozására, és az adatbázis tábláinak kialakítására. Meg tudja határozni egy új tábla mezőneveit, a mezők domainjét, és néhány egyéb tulajdonságukat. A táblák működésének hatékonyabbá tételével és a kapcsolatok kialakításával a következő leckében foglalkozunk.

8.2 ADATBÁZIS LÉTREHOZÁSÁNAK ELŐKÉSZÍTÉSE

A CREATE paranccsal elkészíthetjük egy adatbázis fizikai modelljét, a modell implementálásához, a valódi adatbázis kialakításához azonban teljesülnie kell néhány előfeltételnek.

8.2.1 Jogosultságok

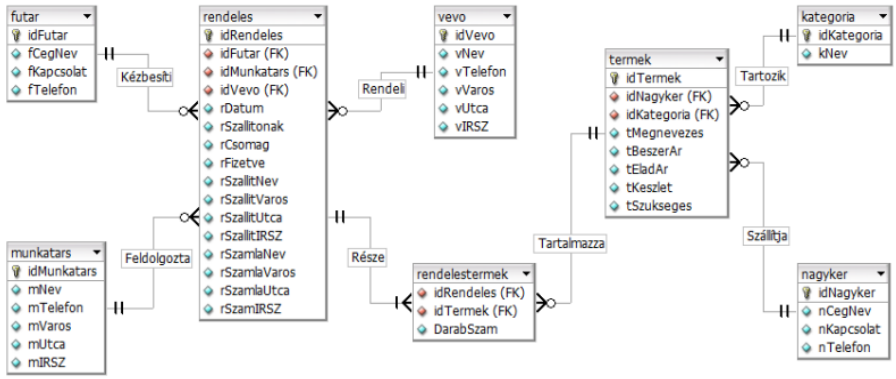
Mielőtt hozzákezdenénk a fizikai adatmodell elkészítéséhez, tudnunk kell, hogy az adatbázis-kezelő rendszerekben szigorú jogosultsági szabályok határozzák meg a felhasználók által elvégezhető műveleteket. A jogosultságokat a DBMS adminisztrátora, a **`root`** nevű felhasználó szabályozza.

A jogosultsági rendszerről később részletesen beszélünk majd. Egyelőre csak annyit szögezzünk le, hogy a lecke példáinak kipróbálásához önnek **`root`**, vagy megfelelő jogokkal rendelkező felhasználónak kell lennie. Amennyiben

saját gyakorló MySQL szervert használ, lépjen be **root**-ként, különben kérjen rendszerszintű CREATE jogot a **root** felhasználótól.

8.3 AZ ADATBÁZIS LOGIKAI MODELLJE

A nagyobb adatbázisok fizikai modelljének elkészítése előtt mindenképpen szükség van a koncepcionális, majd pedig a logikai modellezésre. Tananyagunkban egy webes kereskedést üzemeltető, hardvertermékeket forgalmazó cég adatbázisát hozzuk létre. Az adatbázis erősen egyszerűsített, nem a valós felhasználást, hanem elsősorban a gyakorlást szolgálja. Logikai modelljét az alábbi ábrán láthatjuk. A modell természetesen nem alkalmas a MySQL összes lehetőségének bemutatására, azért esetenként alkalmi példákat is bemutatunk majd.



47. ábra Az adatbázis logikai modellje

*Példaadatbázisunk a **webbolt** nevet viseli, és egy hardver árucikkeket (**termek** tábla) forgalmazó cég munkáját hivatott támogatni. Az árusított termékek különböző kategóriákba tartoznak (**kategoria** tábla), és más-más nagykereskedésből (**nagyker** tábla) származnak.*

*A **vevok** táblában regisztrált vevők rendeléseket adhatnak le (**rendeles** tábla) a **termek** táblában tárolt cikkekre. Egy megrendelés egy vagy több terméket is tartalmazhat. A rendeléseket a bolt dolgozói (**munkatars**) kezelik, és a céggel kapcsolatban álló futárszolgálatok (**futar**) szállítják házhoz.*

8.4 ADATBÁZISOK MEGJELENÍTÉSE

A MySQL adatbázis-kezelő rendszerre bejelentkezve érdemes tájékozódni a meglévő adatbázisok felől. A szerver által üzemeltetett adatbázisok, az egyébként meglehetősen sokoldalúan használható **SHOW** paranccsal listázhatók:

```
SHOW DATABASES;
```

Ha még egyetlen adatbázis sem hoztunk létre, a megjelenő lista az **information_schema** és a **mysql** adatbázisokat fogja tartalmazni. Mindkettő a DBMS működéséhez szükséges adatokat tárolja. A **webbolt** adatbázist természetesen nem látjuk, hiszen az még nem is létezik.

Mielőtt egy létező adatbázissal dolgozni kezdenénk, a **USE** parancs alább látható formájával kell jeleznünk szándékunkat. Az új adatbázis létrehozása után is a **USE**-zal kezdjük meg a munkát.

```
USE adatbázis;
```

A **USE** hatására az adatbázis-kezelő rendszer „használatba veszi" az adatbázist, és további parancsainkat már ebben a kontextusban hajtja végre. A parancs sikeres végrehajtásáról a **Database changed** üzenet tájékoztat.

Ha egy kiválasztott adatbázis helyett másikkal szeretnénk dolgozni, ismét használnunk kell USE parancsot.

8.5 ADATBÁZIS LÉTREHOZÁSA

Az adatbázisok létrehozására a DDL résznyelv **CREATE DATABASE** parancsa használható. A **CREATE DATABASE** az egyszerűbb szintaktikájú parancsok közé tartozik, általános formai leírása a következő:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] adb_neve
    [speciális_tulajdonságok] ...
;

speciális_tulajdonságok:
    [DEFAULT] CHARACTER SET [=] karakterkészlet
  | [DEFAULT] COLLATE [=] összevetés
```

Mivel a **CREATE**-tel más objektumok is létrehozhatók, a parancs után mindenképpen jelezni kell, hogy adatbázist szeretnénk kialakítani, ezért az azo-

nos jelentésű **DATABASE**, vagy a **SCHEMA** kulcsszavak valamelyikének, majd az adatbázis nevének (*adb_neve*) szerepeltetése kötelező.

Ha már létező adatbázist próbálunk létrehozni, a DBMS nem hajtja végre a parancsot, hanem hibaüzenetet küld. Ennek a kötegelt végrehajtás esetén van jelentősége, ugyanis hiba esetén az SQL-szkript futtatása megszakad. Az **IF NOT EXISTS** opció hatására a DBMS csak akkor kísérli meg az adatbázis létrehozását, ha az még nem létezik. Így egyrészt elkerülhető a fenti hiba, másrészt nem fordulhat elő, hogy véletlenül újradefiniálunk egy létező adatbázist.

A szintén opcionális *speciális_tulajdonságok*-kal a szövegmezők karakterkódolását és a szövegek összevetési módját állíthatjuk be. Ha nem szabályozzuk másként, akkor a MySQL a szerver konfigurációjában megadott alapértelmezett karakterkódolást, és szövegösszevetési módot állítja be. Az adatbázisban később kialakított táblák szövegmezőiben az adatbázis beállításai lesznek az alapértelmezettek.

> Mindkét opció esetében a DBMS telepített lehetőségeit használhatjuk az értékek megadásakor. A használható értékek a **SHOW CHARACTER SET**; paranccsal jeleníthetők meg. Szabályozásuk akkor szükséges, ha a DBMS alapértelmezett beállításai nem felelnek meg céljainknak, vagy fizikai modellt más DBMS-eken is használni akarjuk, és szeretnénk biztosítani a megfelelő szövegkezelést.

Az alábbi példa létrehozza a **webbolt** adatbázist, ha az még nem létezik, és az **utf8** karakterkódolást teszi alapértelmezetté.

```
CREATE DATABASE IF NOT EXISTS webbolt
```

```
    DEFAULT CHARACTER SET utf8;
```

A próbaként létrehozott, vagy fölöslegessé vált adatbázisok törlése a **DROP** paranccsal történik. A **DROP** véglegesen törli az adatbázist, ezért használatakor körültekintően kell eljárni. Végrehajtásához külön jogosultságra, a **DROP** jogra van szükségünk.

```
DROP DATABASE adb_neve;
```

```
DROP DATABASE webbolt;
```

8.6 TÁBLÁK MODELLJE

A táblák fizikai modelljének létrehozásakor azt kell megfogalmaznunk, milyennek kell lennie az elkészült adatszerkezetnek. Ehhez meg kell adni a tábla **nevét**, kötelezően föl kell sorolnunk a tábla **mezőit,** és azok összes **tulajdonságát**. Ha szükséges, meg kell határoznunk a **kulcsot**, létre kell hozni az **indexeket**, jelölni kell az **idegen kulcsokat**, illetve a tábla egészére vonatkozó **tulajdonságokat**.

8.6.1 A CREATE TABLE parancs

Az adatbázisok létrehozásához hasonlóan, a táblákat is a **CREATE** parancscsal készítjük el, de a parancs után a **TABLE** kulcsszót használjuk. Ezután a tábla **neve**, majd a tábla szerkezetét (mezők, indexek, megszorítások) meghatározó, kerek zárójelek közé zárt **tábladefiníció** következik.

A táblára vonatkozó **tulajdonságokat** ezután helyezhetjük el.

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] táblanév
    (tábla_definíció)
    [tábla_tulajdonságok];
```

A **TEMPORARY** opció hatására nem valódi, hanem csak memóriában tárolt virtuális tábla jön létre, ami a munkamenetünk végén automatikusan törlődik.

Az **IF NOT EXISTS** jelentését már ismerjük, az újdonságot a tábla definíció megadása jelenti. A tábla szerkezetének definiálása mezők, és megszorítások leírásából áll. Utóbbiak alatt az indexek és idegen kulcsok definiálást értjük. Ebben a leckében egyelőre csak az meződefiníciókkal foglalkozunk.

8.6.2 Meződefiníciók

A kerek zárójelek közé kerülő meződefiníciók egymástól **vesszővel elválasztott meghatározások**. Minden meződefinícióban kötelezően szerepel a **mező neve** és **adattípusa**, de tartalmazhat további **mezőtulajdonságokat** is.

```
mezőnév adattípus [mezőtulajdonságok]
```

A **mezőnév** a mező azonosítója, a logikai modellben meghatározott neve lesz. Később ezzel a névvel hivatkozhatunk az attribútumra. Mint tudjuk, a relációs adatmodell előírja, hogy egy mező minden értékének azonos domain-be kell tartoznia. A domain-t, a mezőnevet követő **adattípus** határozza meg. A

mezőtulajdonságok megadása opcionális. Ide további jellemzők, mint például a **NULL** értékek engedélyezése vagy tiltása, a mező alapértelmezett értékének megadása, kulcsként való megjelölése kerülhet. Az alábbi formai leírás a lehetséges mezőtulajdonságokat összegzi.

```
[NOT NULL | NULL] [DEFAULT alapérték]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT 'megjegyzés'] [hivatkozás]
```

Mielőtt az egyre sokasodó opciók végképp elriasztanák a kedves olvasót, lássunk egy egyszerű, de működő példát, amely fizikailag modellezi a korábban bemutatott logikai modell, **kategoria** nevű táblájának egy egyszerűsített változatát.

```
CREATE TABLE IF NOT EXISTS kategoria
(
    idKategoria     INT UNSIGNED,
    kNev            VARCHAR(30)
)
;
```

Az SQL-mondatot egyetlen sorban is begépelhettük volna, de a jobb áttekinthetőség kedvéért szándékosan tördeltük.

*A parancs a **kategoria** nevű táblát modellezi. A táblában két meződefiníció, az **idKategoria** és a **kNev** meghatározása található. Az **idKategoria** mezőbe egész szám (**INT**) kerülhet, amely nem lehet negatív (**UNSIGNED**). A **kNev** mező maximum 30 karakter hosszúságú szöveget (**VARCHAR(30)**) tárolhat.*

Az egyszerűsítés abban áll, hogy nem határoztuk meg, hogy melyik mező legyen a kulcs.

*A következő példa már a logikai modellnek megfelelően határozza meg a **kategoria** a tábla fölépítését. Figyeljük meg, hogy a mezőnevet követő első kulcsszó mindig az adattípust, a vesszőig tartó továbbiak pedig, a mezőtulajdonságokat jelölik. Ezekből több is lehet, de a fenti sorrendben kell, hogy kövessék egymást.*

Ha szeretnénk kipróbálni a példát, de az előbb már készítettük **kategoria** nevű táblát, akkor most természetesen nem készül új reláció, hiszen használtuk az **IF NOT EXISTS** opciót. Ha újra létre akarjuk hozni, előbb törölnünk kell a táblát:

```
DROP TABLE kategoria;
```

```
CREATE TABLE IF NOT EXISTS kategoria
(
  idKategoria    INT
                 UNSIGNED
                 NOT NULL
                 PRIMARY KEY
                 AUTO_INCREMENT,
  kNev           VARCHAR(30)
                 COMMENT 'A kategória neve'
)
;
```

*Az **idKategoria** mezőben a pozitív számokra korlátoztuk a domain-t (**UNSIGNED**), és nem engedjük meg a **NULL** értéket (**NOT NULL**). Ez a mező lesz a tábla elsődleges kulcsa (**PRIMARY KEY**), amelynek értékeit automatikus számozással (**AUTO_INCREMENT**) állítja elő a DBMS.*

*A **kNev** mező esetében egyetlen mezőtulajdonságot adtunk meg. A **COMMENT** után lévő szöveg a mező leírása, amivel a tulajdonságtípus szerepére emlékeztetheti magát az adatbázis tervezője.*

*A fenti SQL-mondat az ENTER leütése után a DBMS-hez kerül, amely értelmezi és végrehajtja az utasítást: létrehozza a **kategoria** táblát*

8.6.3 A mezők adattípusa

A mezők lehetséges értékkészletét, azaz domainjét elsősorban az adattípus megadásával szabályozzuk. Amikor megadjuk egy mező adattípusát, valójában három dologról döntünk.

- Korlátozzuk a mezőben tárolható értékeket, azaz beállítjuk a mező lehetséges **értékkészletét**.
- Ezzel meghatározzuk, milyen műveleteket lehet majd elvégezni a mezőben tárolt adatokkal. A szövegmezők értékéből például nem vonhatunk gyököt, míg egy számmal tetszőleges numerikus művelet végezhető el.
- Végül – közvetve ugyan, de – meghatározzuk a mező tárigényét is.

Amikor egy mező számára adattípust választunk, az alábbi szempontokat érdemes megfontolni:

- Milyen értékeket szeretnénk tárolni a mezőben? (Nem használhatunk például szám típust, ha a mezőben neveket kell tárolnunk.)
- Ha egy mező értékeinek tárolására több típus is alkalmas lenne, döntsön az, hogy milyen művelteket szeretnénk elvégezni a tárolt adatokkal. (A számokat elvileg tárolhatjuk szöveges mezőben is, de ebben az esetben numerikus műveleteket már nem tudunk végezni velük.)
- Ha a két előző szempont alapján még mindig nem tudunk dönteni, válasszuk az alkalmas típusok közül azt, amelyiknek a lehető legkisebb a tárigénye!

8.7 A MYSQL TÍPUSAI

A MySQL-ben három nagyobb csoportba sorolhatjuk a használható adattípusokat.

8.7.1 Numerikus adattípusok

A numerikus adattípusú mezőkben a számok valóban számként tárolódnak, így azok későbbi értékeivel elvégezhetők a különböző matematikai műveletek. A szövegmezőkben szintén tárolhatunk számokat, de azokkal már nem végezhetünk számításokat. A számok tárolására egyébként többféle adattípust is biztosít a MySQL.

Egész számok

A MySQL több egész típussal támogatja, hogy a lehető legjobban igazodjunk a tárolni kívánt értékek nagyságrendjéhez. A megfelelő típus kiválasztásával az adatbázis méretének és működési sebességének optimalizálásához is hozzájárulhatunk. Válasszunk mindig olyan számtípust, amelyben biztosan elférnek majd az adataink, de a mező mérete a lehető legkisebb legyen.

Minden egészszám típus esetén szabályozhatjuk a negatív számok érvényességét. Ha máshogyan nem rendelkezünk, a mezőben tárolt érték lehet nullánál kisebb is. Ha csak pozitív számokat szeretnénk megengedni, akkor típus neve után fel kell tüntetni **UNSIGNED** kulcsszót. Az alábbi táblázat összefoglalja a rendelkezésre álló egész típusok neveit, bájtokban megadott tárigényét, előjeles, illetve előjel nélküli értékkészletét:

Típus	Bájt	Legkisebb érték	Legnagyobb érték
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32 768	32 767
		0	65 535
MEDIUMINT	3	-8 388 608	8 388 607
		0	16 777 215
INT	4	-2 147 483 648	2 147 483 647
		0	4 294 967 295
BIGINT	8	-9 223 372 036 854 775 808	9 223 372 036 854 775 807
		0	18 446 744 073 709 551 615

Az egész típusok deklarálásakor a típusnév után kerek zárójelekbe írt számmal – pl. INT(10) – megadhatjuk a mező tervezett megjelenési szélességét. A szám nem korlátozza a mező értelmezési tartományát, pusztán azt állítja be, hogy az értékek legalább hány karakter szélességben jelenjenek meg a képernyőn. Ezt csak bizonyos adatbázis-alkalmazások használják, a mysql parancssori kliens például figyelmen kívül hagyja.

Ha azonban a mezőtulajdonságok között feltűntetjük ZEROFILL opciót, akkor az itt megadottnál kevesebb karakterrel ábrázolható számok megjelenítéskor azok bal oldalán megfelelő számú vezető nullát helyez el a DBMS.

Valós számok

A valós típusok tört számok tárolására teszik alkalmassá a mezőket. Az ilyen típusok mindegyikénél szabályozható az összes tárolható számjegyek száma (M), és az, hogy ezen belül mennyi a tizedesjegyek száma (D). Az alábbi táblázat a használható valós típusokat, az azokban maximálisan tárolható számjegyek számát, valamint a tárolható legkisebb és legnagyobb értékeket fogalja össze. A FLOAT és a DOUBLE lebegőpontos, a DECIMAL fixpontos ábrázolással tárolja a számokat.

Típus	Számjegyek	Mim/max érték
FLOAT(M,D)	24	Min=+/-1.175E-38 Max=+/-3.403E+38
DOUBLE(M,D)	53	Min=+/-2.225E-308 Max=+/-1.798E+308
DECIMAL(M,D)	64	Mint a DOUBLE, de fixpontos

Szöveges típusok

A szöveges típusú mezőkben bármilyen karakterekből álló sztringet elhelyezhetünk. Az egyes típusok a maximálisan tárolható karakterek számában térnek el.

Típus	Karakterszám	Megjegyzés
CHAR	0..255	A típus meghatározásakor megadható a szöveg maximális hossza. Ha a valóban tárolt karakterek száma ettől kevesebb, a mező jobbról szóközökkel töltődik fel.
VARCHAR	0..255	Változó hosszúságú szöveg tárolására alkalmas. A mező definiálásakor meg kell adni a szöveg maximális hosszát. A mező tényleges hossza a valóban tárolt karakterek számának felel majd meg.
BLOB, TEXT	0..65535	Nagyobb szöveg tárolására alkalmas adattípus. A BLOB típust (Binary Large OBject) nem csak szövegek, hanem bináris adatok, például képek tárolására is használják. A két típus közötti különbség a rendezéskor mutatkozik meg. A BLOB típusú mezők kis-és nagybetű érzékenyek, míg a TEXT típusúak nem.
TINYBLOB, TINYTEXT	0..255	
MEDIUMBLOB, MEDIUMTEXT	0..16 777 215	
LONGBLOB, LONGTEXT	4 294 967 295	

A szöveges típusok között említik még a felsorolás, azaz **ENUM** típust, amelynek értékkészlete egy maximum 65535 szöveges elemből álló halmaz. Akkor használjuk, ha egy mező értékei előre meghatározható szövegekből kerülnek ki. Az ENUM típussal elérhető, hogy a felhasználó csak a definiáláskor meghatározott értékek valamelyikét tárolhassa a mezőben.

Dátum és idő típusok

A számok és szövegek tárolására alkalmas típusokhoz hasonlóan a MySQL több típussal támogatja dátum- és időértékek tárolására alkalmas mezők definiálását. A leggyakoribb a csak dátumokat befogadó **DATE** típus használata, de a megfelelő típus megválasztása itt is sokat számíthat az adatbázis hatékony működésének finomhangolásában.

Az alábbi táblázat összefoglalja a dátum és idő tárolásra használható típusokat.

Típus	Megjegyzés
DATE	ÉÉÉÉ-HH-NN formában megadható, és tárolódó dátumtípus.
DATETIME	ÉÉÉÉ-HH-NN ÓÓ:PP:MM formában megadott dátumot és időpontot tároló típus. A megadott értéknek 1001-01-01 00:00:00 és 9999-12-31 23:59:59 között kell lennie.
TIMESTAMP	Változó hosszúságú időbélyegző, amely 14, 12, 8 vagy 6 karakteren, elválasztó jelek nélkül tárolja a dátumokat és időpontokat. (A hosszúságot a típus deklarálásakor kell megadni.) A hosszúság szerinti formátumok az alábbiak
	14 karakter: ÉÉÉÉHHNNÓÓPPMM,
	12 karakter: ÉÉHHNNÓÓPPMM
	8 karakter: ÉÉÉÉHHNN
	6 karakter: ÉÉHHNN
TIME	ÓÓ:PP:MM formátumú időpontok tárolására alkalmas típus.
YEAR	Évszámok tárolására alkalmas típus, amely négy vagy kettő karakter hosszúságot enged meg.

A TIMESTAMP típus nem azonos az úgynevezett UNIX TIMESTAMP-pel. A MySQL TIMESTAMP egy dátum/idő elválasztó jelek nélküli ka-

rakteriből áll, a UNIX TIMESTAMP pedig az 1970.01.01 óta eltelt, másodpercekben megadott idő.

8.7.2 Egyéb típusok

A MySQL-ben további típusok is rendelkezésre állnak, de ezek általában a fentebb bemutatottak specializált változatai. A **BOOLEAN** típus például logikai értékek tárolására alkalmas, de valójában **TINYINT**, amelyben a tárolt nulla (0) a logikai FALSE, a nullától eltérő érték pedig a logikai TRUE megfelelője.

Szintén jellemző, hogy egy típusra több, szinonim névvel lehet hivatkozni. Ilyen például a **DECIMAL** és a **NUMERIC** számtípus, amelyek megegyeznek egymással. Az ilyen szinonimákat azért értelmezik a DBMS-ek, hogy biztosítsák az újabb és korábbi verziók kompatibilitását.

8.8 ÉRTÉKKÉSZLETET SZABÁLYOZÓ MEZŐTULAJDONSÁGOK

A mezők domainjének szabályozása nemcsak a típusok megfelelő megválasztásával, hanem a típus meghatározást követő opcionális mezőtulajdonsággal is lehetséges.

8.8.1 Kötelező kitöltés

A **NOT NULL** opció használata esetén a mező kitöltése kötelező, az ilyen mezők nem maradhatnak üresen. A kötelező mezőkben bármilyen értéket tárolhatunk, amit a választott típus megenged. Ha az adatbázis használója **NULL** értéket próbál elhelyezni a kötelező mezőben, az adatbázis-kezelő rendszer nem hatja végre a parancsot, hanem hibaüzenet küld.

A **NOT NULL** opciónak fontos szerepe van a táblák közötti kapcsolatban, a minimális részvétel, azaz a kötelezőség beállításában. Erről a következő leckében olvashatunk részletesen.

8.8.2 Negatív számok

A szám típusú mezők esetén szabályozhatjuk, hogy a felhasználó tárolhat-e nullánál kisebb számokat. A pozitív számok előírásakor az **UNSIGNED** opciót

kell használni. Ha ezt elhagyjuk, a mezőben negatív számok tárolására is lesz lehetőség.

8.8.3 Alapértelmezett érték

A **DEFAULT** opciót követő literállal a mező alapértelmezett értékét határozhatjuk meg. Ha új rekord felvételekor nem kap értéket, akkor a DBMS az itt megadott literált tárolja a mezőben.

8.9 TÁBLÁK TÖRLÉSE

Az adatbázis készítése közben bármikor előfordulhat, hogy egy tábla teljesen fölöslegessé válik. A táblák végleges törlését a DDL **ALTER TABLE** parancsával végezhetjük el.

```
ALTER TABLE táblanév;
```

8.10 ÖSSZEFOGLALÁS, KÉRDÉSEK

8.10.1 Összefoglalás

Mai leckénkkel az adatbázis-modellezés fizikai szintjén kezdtük meg az adatbázisok szerkezetének ábrázolását. Ez a szint a legalacsonyabb absztrakciót használja, azért az itt elkészített és megfelelően formalizált modellek közvetlenül implementálhatók az adatbázis-kezelő rendszerek segítségével. A relációs adatmodellben az IBM által kidolgozott, később szabvánnyá nyilvánított SQL nyelv DDL résznyelvét használhatjuk a fizikai modellezésre. A SQL a relációs adatbázis-kezelő rendszerek standardizált nyelve, amely DCL, DDL, DML, és DQL résznyelveihez tartozó utasításaival az adatbázis-kezeléssel kapcsolatos összes műveletek leírását lehetővé teszi.

Az SQL adatdefiníciós résznyelvének **CREATE** parancsával a fizikai adatbázis különböző strukturális elemei, és azok tulajdonságai definiálhatók.

A **CREATE DATABASE** paranccsal új adatbázist, a **CREATE TABLE** kezdetű SQL-mondatokkal pedig új táblát hozhatunk létre.

A relációk kialakításakor pontosan meg kell adni a tábla nevét, majd le kell írni annak struktúráját. A leírás a meződefiníciókból és az azután következő, úgynevezett megszorításokból áll. Mai leckénkben egyelőre a meződefiníciókkal foglalkoztunk. Megtanultuk, hogy ezek a mező értékkészletét és értelmezési tartományát leginkább befolyásoló adattípusok, és egyéb mezőtulajdonságok meghatározásából állnak.

A típus leírását követően a domaint pontosító **NOT NULL, UNSIGNED,** illetve az alapértelmezett érték meghatározására alkalmas **DEFAULT** opciók használatát ismertük meg.

8.10.2 Önellenőrző kérdések

1. Mi a szerepe az SQL egyes résznyelveinek?
   - A DCL az adatbázis-kezelő rendszer közvetlen vezérlésére használható. A DDL a fizikai modellezés, a DML az adatok manipulálásának résznyelve, a DQL pedig lekérdezések készítésére alkalmas.
2. Hogyan küldhetünk SQL-mondatokat a MySQL szervernek?
   - Valamilyen kliens segítségével kell a szerverhez kapcsolódni. Legegyszerűbb esetben a MySQL parancssori kliensét használhatjuk erre a célra.
3. Hogyan tudunk több SQL-mondatot tartalmazó szkripteket végrehajtatni?
   - A parancsokat szövegfájlban kell tárolni. A parancssori kliensben a \. direktíva után a szkript nevét megadva a klines beolvassa a fájlt, és az abban található SQL-mondatokat végrehajtatja a szerverrel. Például:
     mysql>\. \sqls\script.sql
4. Milyen szempontokat érdemes figyelembe venni az adattípus megválasztásakor?
   - Tárolhatók legyenek a mező várható értékei, a mező alkalmas legyen a tervezett műveletek elvégzésére, legyen a lehető legkisebb a tárigény.
5. Mit tud elmondani az alábbi tábladefinícióról?

```
CREATE TABLE termek (
idTermek int(10)
    UNSIGNED
    NOT NULL
    PRIMARY KEY
    AUTO_INCREMENT,
  tMegnevezes varchar(50) DEFAULT NULL,
  tEladAr int(10) UNSIGNED DEFAULT 0,
```

```
  tKeszlet int(10) UNSIGNED DEFAULT 0
);
```

- A fenti sor a **termek** tábla fizikai modellje. A táblában az **IdTermek** mező az automatikus számozású kulcs. Értéke nem lehet negatív, sem pedig NULL. A **tMegnevezes** mező maximum 50 karakteres, kötelezően kitöltendő szövegmező. A **tEladAr** és a **tKeszlet** mezők pozitív egész számok, mindkettő alapértelmezett értéke nulla.

9. LECKE: INDEXELÉS, KAPCSOLATOK, TÁBLATULAJDONSÁGOK

9.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

Miután előző leckénkben megismerkedtünk a táblák fizikai modellezésének alapjaival, mai tananyagunkban rátérünk a relációs adatmodell integritási részében foglaltak fizikai megvalósítására.

Elsőként a MySQL táblamotorjaival ismerkedünk meg, és megtanuljuk, milyen jelentőségük van az integritási szabályok betartásában.

Ezután megvizsgáljuk azokat a nyelvi lehetőségeket, amelyekkel SQL-ben leírhatók az egyed- és hivatkozási integritás szabályai. Megtanuljuk az egyszerű és összetett elsődleges kulcs kialakításának módját, a kulcsérték automatikus beállításának szabályozását. Megismerjük az indexelés jelentőségét, és az indexkészítés módját.

Végül megtanuljuk, hogyan állíthatjuk be az idegen kulcsok hivatkozásait, és hogyan vehetjük rá a DBMS-t a hivatkozási integritás megőrzésére.

A lecke anyagának elsajátítása után Ön már képes lesz teljes értékű MySQL-adatbázisok kialakítására. Pontosan tudja majd modellezni a táblák fizikai szerkezetét, és le tudja írni az azok között lévő kapcsolatokat. Képes lesz indexeléssel optimalizálni az adatbázis működésének sebességét.

9.2 TÁBLAMOTOROK

Az adatbázis-kezelő rendszerek ANSI-SPARC architektúrája alapján az adatbázis a legbelső szinten is a logikai adatmodellnek megfelelő formában jelenik meg. A relációs modell esetében ez azt jelenti, hogy táblákban, mezőkben, rekordokban, és mezőértékekben gondolkodva kezelhetjük az adatokat. A táblák fizikailag mindig valamilyen háttértáron, lemezen tárolódnak. A lemezfájlok belső struktúrája természetesen nem táblaszerkezetű, hiszen a lemezeken bitek sorozata tárolódik. Ez kissé leegyszerűsítve azt jelenti, hogy egy tábla lemezre írásakor az adatbázis-kezelő rendszernek meg kell oldania a táblák bitsorozattá konvertálását, beolvasásukkor pedig a lemezről betöltött biteket kell táblaszerkezetre leképeznie.

A MySQL-ben több módszer is létezik a lemezfájlok belső struktúrájának kialakítására és kezelésére. Ezeket a DBMS különálló, akár egyenként is telepíthető moduljai, az úgynevezett **táblamotorok** biztosítják. Az egyes motorok elté-

rő kiegészítő lehetőségeket biztosítnak a relációk kezelésének alapvető funkcióin túl. Egy tábla definiálásakor megadhatjuk, hogy melyik táblamotort szeretnénk használni, a MySQL pedig mindig a kiválasztott módon fogja kezelni a relációt. Természetesen csak azok a motorok használhatók, amelyeket előzetesen telepítettek a DBMS-ben.

A telepített táblamotorok a **`SHOW ENGINES;`** DCL-paranccsal tekinthetők meg. A **`Support`** oszlopból leolvashatjuk, hogy melyik motor elérhető, sőt azt is, hogy melyik az alapértelmezett. Ha külön nem jelezzük, a MySQL mindig ezt fogja használni.

```
mysql> show engines;
+--------------------+---------+----------------------------------------------------------------+--------------+------+
| Engine             | Support | Comment                                                        | Transactions | XA   |
+--------------------+---------+----------------------------------------------------------------+--------------+------+
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   |
| MyISAM             | YES     | MyISAM storage engine                                          | NO           | NO   |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   |
| CSV                | YES     | CSV storage engine                                             | NO           | NO   |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   |
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   |
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     | YES          | YES  |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   |
+--------------------+---------+----------------------------------------------------------------+--------------+------+
9 rows in set (0.06 sec)
```

48. ábra SHOW ENGINES;

A MySQL saját, eredeti táblamotorja az úgynevezett **MyISAM** nem támogatja a tranzakció kezelést, és a hivatkozási integritás megőrzését sem. Éppen ezért általában az **InnoDB** motort állítják be alapértelmezettként. A fentiek miatt mi is ezt fogjuk használni.

> **A MySQL adatbázisokban az InnoDB motorral kezelt táblákban használhatunk tranzakciókezelést, csak az InnoDB relációkban őrizhetjük meg az idegen kulcsok helyességét.**

9.3 TÁBLÁK OPCIÓINAK BEÁLLÍTÁSA

Egy tábla létrehozásakor a tábla tulajdonságai között adhatjuk meg a használni kívánt motor nevét. Mivel ennek hatása van a tábladefinícióban használható lehetőségekre, célszerű elsőként a táblatulajdonságok beállításával foglalkoznunk.

A táblák fizikai modellezésének előző leckében megismert általános leírása szerint a táblatulajdonságok a definíciót közrezáró zárójelek után adhatók meg.

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] táblanév
    (tábla_definíció)
    [tábla_tulajdonságok];
```

A tulajdonságok egymástól szóközökkel elválasztott *név=érték* formában megadott értékadásokkal szabályozatók. Mi csupán a táblamotort (**ENGINE**) és a tábla alapértelmezett karakterkódolását (**DEFAULT CHARSET**) fogjuk beállítani, de valójában közel 20 ilyen tulajdonság létezik.

A MySQL-szerverek konfigurációjában megadható az alapértelmezett táblamotor, karakterkódolás, és a többi táblatulajdonság is. Explicit beállításra ezért csak az ettől eltérő esetekben van szükség. A használható táblamotorok a **SHOW ENGINES;**, míg a karakterkódolások a **SHOW CHARACTER SET;** parancsokkal tekinthetők meg.

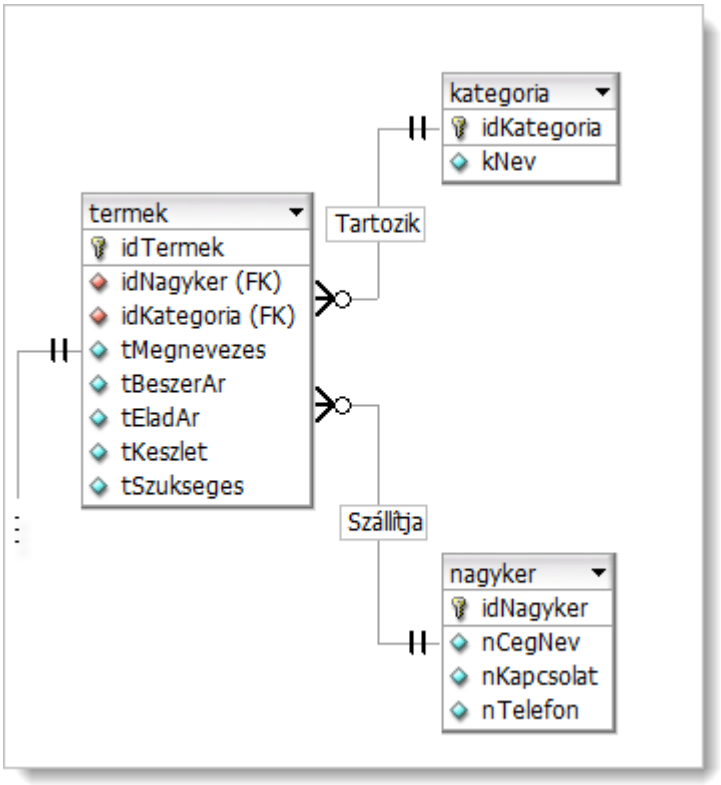
Az alábbi példa az előző leckében létrehozott **kategoria** tábla olyan modelljét mutatja be, amelyben explicit módon meghatározzuk a táblamotort, és a karakterkódolást is.

```
CREATE TABLE IF NOT EXISTS kategoria
   (
   idKategoria INT  UNSIGNED
               NOT NULL
               PRIMARY KEY
               AUTO_INCREMENT,
   kNev  VARCHAR(30) COMMENT 'A kategória neve'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

9.4 INDEXELÉS, ÉS INTEGRITÁSI SZABÁLYOK

A táblák eddig tanult definiálási lehetőségei már biztosítják, hogy a logikai adatmodellnek megfelelő táblaszerkezeteket alakítsunk ki, de mégsem alkalmasak a teljes fizikai modell helyes ábrázolására. Ennek az az oka, hogy a meződefiníciók egyszerű felsorolása még nem biztosítja a relációs adatmodell két fontos integritási szabályának, az egyed-integritás és a hivatkozási integritás megvalósítását. Emellett azt is érdemes megemlíteni, hogy az eddig tanult módszerek azt sem teszik lehetővé, hogy a későbbi rekordkezelő műveletek sebességét optimalizáló adatszerkezetet hozzunk létre. Ezeket a funkciókat fogjuk biztosítani az indexelés, az elsődleges kulcs beállítása, és az idegen kulcsok segítségével.

A következő példákban a logikai modell **termek** tábláját hozzuk majd létre.



49. ábra termek-kategoria, termek-nagyker

Az indexelés, az elsődleges kulcs megadása, és az idegen kulcsok kijelölése is megvalósítható mezőtulajdonságok megadásával, vagy a meződefiníciókat követő úgynevezett megszorítás-definíciókkal is. Az utóbbi megoldás jóval átláthatóbb táblaszerkezetet eredményez, ezért tananyagunkban ezt a lehetőséget ismertetjük.

9.5 INDEXELÉS

Ahogyan azt korábban megjegyeztük, a táblák rekordjai háttértárakon tárolódnak. Amikor a felhasználó meg szeretne nézni néhány rekordot, a DBMS-nek a lemezről kell beolvasnia az adatokat, majd relációvá alakítva el kell küldenie a felhasználó kliensprogramjához. A **lemezműveletek mindig kritikus pontot jelentenek a sebesség tekintetében**. Amikor egy tábla valamelyik mezőjének értékétől függően akarjuk a rekordok egy részét kiválogatni, ez a probléma hatványozottan jelentkezik. A DBMS ugyanis úgy állítja össze a rekordok szükséges részhalmazát, hogy minden rekordot beolvas, és a **kérdéses mező rekor-**

donkénti vizsgálata alapján eldönteni, hogy mely rekordok kerüljenek az eredményhalmazba. Előfordulhat, csak a legutolsó rekord, vagy egyetlen egyed sem elégíti ki a feltételt, de ennek eldöntéséhez mindig minden rekordot meg kell vizsgálni.

Az **indexelés** jelentősen **redukálja** az ilyen műveletek lebonyolításához **szükséges időt.**

A technika lényege, hogy a **bázistábla** (a rekordok adatait tároló reláció) létrehozásakor **bármelyik mezőt indexelhetjük**. Az indexelés azt jelenti, hogy a DBMS úgynevezett **indextáblát** (röviden indexet) hoz létre a mező számára. A bázistábla feltöltésekor az indexelt mező **összes előforduló értékét bemásolja az indextáblába**, és az index **rekordjait sorba rendezi** az értékek szerint. Az indextáblában a minden mezőérték mellett följegyzi azt is, hogy a **bázistábla** adott értéket tároló **rekordja hol található a lemezen**.

> **Az indextábla minden sora az indexelt mező egye-egy értékét és a hozzá tartozó rekordmutatót tárolja. A sorok az indexelt értékek szerint rendezettek.**

Ha a bázistáblában egy indexelt mező alapján akarunk rekordokat keresni, vagy kiválogatni, akkor a DBMS automatikusan beolvassa a memóriába a teljes indextáblát, kiválogatja a megfelelő mezőértékeket (ez a művelet a rendezettség miatt gyors lesz), és bázistáblának csak a kiválasztott, az ezeket az értékeket tartalmazó rekordjait olvassa be a lemezről.

Egy táblában bármennyi mező indexelhető. Ha több mezőt is indexelünk, akkor természetesen mindegyik számára különálló indextábla készül.

A fentiek alapján könnyen következtethetünk arra, hogy érdemes minden mezőt indexelni. Ez nem pontosan így van, ugyanis az indexelésnek negatívumai is vannak. Minden indextábla helyet foglal a lemezen, és ezzel **növeli az adatbázis méretét**. Ráadásul az indexek csak akkor működnek helyesen, ha a DBMS a bázistábla rekordjainak változásakor (új rekord, rekordtörlés, mezőérték változtatás) automatikusan frissíti az indextáblát is. Ha például új rekord kerül a bázistáblába, akkor be kell tölteni az indexet, beszúrni a bázistábla az új mezőértékét, majd újrarendezni és lemezre másolni az indextáblát. Ez mindig időt vesz igénybe, azért az indexelés némileg lassítja a rekordkezelő műveleteket.

Ha helyesen akarunk eljárni, akkor csak **azokat a mezőket indexeljük**,

- amelyek alapján **gyakran válogatunk ki rekordokat** egy nagyméretű bázistáblából, vagy

- biztosítani akarjuk, hogy a mező rekordonként **egyedi értéket tartalmazzon**, azaz alternáló kulcs legyen, vagy
- a **mezőt idegen kulcsként** akarjuk használni.

9.5.1 Indexelés típusai

Indexet létrehozhatunk egy vagy több mező értékeinek kombinációja alapján. Az indexelés lehet egyedi, és nem egyedi.

Az egyedi index használata gyorsabb, de csak olyan mezők esetében lehetséges, amelyek egyes értékei a bázistábla egy-egy rekordjában fordulnak csak elő, nem ismétlőnek, azaz valójában kulcsjelöltek.

Pontosan ezért az **egyedi indexelésű mezőkben** a DBMS **nem engedi meg az mezőértékek ismétlődését**. Az ilyen indexelést nem csak a kereséséi sebesség növelésére, hanem egy mező egyedi értékeinek biztosítására is használjuk.

9.5.2 Indexelés beállítása

Nem egyedi indexelés

A nem egyedi indexelés a meződefiníciók után, azoktól vesszővel elválasztva az alábbi formában adható meg:

```
INDEX [index_név] [index_típus] (indexelt_mező,...)
      [index_opciók]
```

Az könnyebb megértés kedvéért lássunk egy egyszerű példát, amely az összes választható opciót elhagyva állítja be a ***nagyker*** *tábla* ***nKapcsolat*** *mezőjének nem egyedi indexelését. Abból indulunk ki, hogy gyakran keressük a nagykereskedések rekordjait a kapcsolattartó neve alapján, de feltételezzük, hogy a különböző nagykerek munkatársainak lehet azonos a nevük.*

```
CREATE TABLE nagyker (
    idNagyker int(10) UNSIGNED NOT NULL
                      PRIMARY KEY AUTO_INCREMENT,
    nCegNev varchar(50)  DEFAULT NULL,
    nKapcsolat varchar(30) DEFAULT NULL,
    nTelefon varchar(30) DEFAULT NULL,
    INDEX idx1 (nKapcsolat)
 )ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

A példában csak az *index_név*, és az *indexelt_mező* opciókat használtuk ezért röviden most tekintsük át az összes lehetőség jelentését!

- *index_név*: Példánkban '**idx1**', az index neve. Ez tábla szinten egyedi karaktersorozat, amivel az indexet azonosíthatjuk, illetve hivatkozhatunk rá. Az index későbbi törlésekor, vagy módosításkor lehet rá szükség, ezért bár nem kötelező, célszerű megadni.
- *index_típus*: A kétféle indexelési típus a b-fa (BTREE) és a hashing (HASH) között választhatunk. A HASH csak a MEMORY és az NDB táblamotorok esetében használható, az InnoDB, és a többi motor mindig b-fa típusú indexet készít.
- *indexelt_mező*: Példánkban egyedül az indexelt mezőt adtuk meg. A zárójelek között egyszerű index esetén egy, összetett indexelés esetén több, vesszővel elválasztott mezőnevet sorolunk fel.
- *index_opciók*: az indexelés további, finomhangolására használható opciói.

Egyedi indexelés:

Az egyedi indexelés szintén a meződefiníciók után, vesszővel elválasztott, úgynevezett megszorítás definícióval adató meg:

```
[CONSTRAINT [szimb_név]] UNIQUE [INDEX|KEY]
       [index_név] [index_típus] (indexelt_mező,...)
       [index_opciók] ...
```

A megszorítások mindegyike a **CONSTRAINT** kulcsszóval kezdődik, amit egy adatbázis szintjén egyedi, úgynevezett szimbolikus név követ. Szükség esetén ezzel a névvel azonosítható a megszorítás. Az ezt követő **UNIQUE** kulcsszó jelzi, hogy **egyedi indexet** szeretnénk készíteni. Az **INDEX|KEY** kulcsszavak kompatibilitási okokból maradtak a nyelvben. Egymás szinonimái, de el is hagyhatók. Minden ezt követő opció jelentése megegyezik a nem egyedi indexelésnél leírtakkal.

Tegyük fel, hogy webes kereskedésünk adatbázisában egyedi indexeléssel szeretnénk biztosítani, hogy ne lehessen kétszer szerepeltetni ugyanazt a cégnevet. Az alábbi példában úgy módosítanunk a tábla modelljét, hogy az ***nNEv*** *mező értékei egyedi indexelésűek legyenek.*

```
CREATE TABLE nagyker (
    idNagyker int(10) UNSIGNED NOT NULL
                      PRIMARY KEY AUTO_INCREMENT,
    nCegNev varchar(50)  DEFAULT NULL,
    nKapcsolat varchar(30) DEFAULT NULL,
    nTelefon varchar(30) DEFAULT NULL,
    INDEX idx1 (nKapcsolat),
    CONSTRAINT szimNagykIdx UNIQUE idx2 (nCegNev)
 )ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

9.6 ELSŐDLEGES KULCS

Az indexeléssel az adatbázis hatékonyságát növelhetjük, az elsődleges kulcs beállítása azonban a relációs adatmodell alapvető integritási szabályának megvalósítása.

Mint tudjuk, a relációs modell minden relációjában kötelező, hogy legyen olyan, kulcsként explicit módon megjelölt mező, amelynek egyedi értékei alkalmasak a rekordok egyértelmű megkülönböztetésére, és amely nem tartalmazhat **NULL** értéket.

A kulcs tehát valójában egy egyedi indexelésű mező, amelynek mezőtulajdonságai között szerepel a **NOT NULL** érték. Az ilyen mezőket a **UNIQUE** indexelés és a **NOT NULL** tulajdonság kombinációjával is létrehozhatunk, de ha a mezőt valóban kulcsként akarjuk használni, akkor a **PRIMARY KEY** megszorítást kell használnunk.

Hallgatólagosan már a múlt leckében, és eddigi példáinkban is használtuk az elsőleges kulcs beállításnak egy lehetséges formáját. Eddig a **PRIMARY KEY** mezőtulajdonsággal jelöltük meg a kulcsot, ami mellett az **AUTO_INCREMENT** tulajdonságot is feltüntettük. Utóbbival a mező automatikus számozását biztosítottuk.

Az AUTO_INCREMENT tulajdonságot kulcsként megjelölt, egész szám típusú mezőkre használhatjuk. A tulajdonság hatására az adatbázis-

kezelő rendszer minden új rekordod fölvételekor automatikus sorszámot helyez el a mezőben. Egy táblában csak egy AUTO_INCREMENT mező lehet!

Elsődleges kulcsként az esetek döntő többségében előjel nélküli **INT** típusú mezőket használunk, és előírjuk, hogy a DBMS automatikus sorszámozással kezelje a mezőt.

```
CREATE TABLE nagyker (
    idNagyker int(10) UNSIGNED NOT NULL
                      PRIMARY KEY AUTO_INCREMENT,
    nCegNev varchar(50)  DEFAULT NULL,
    nKapcsolat varchar(30) DEFAULT NULL,
    nTelefon varchar(30) DEFAULT NULL,
    INDEX idx1 (nKapcsolat),
    UNIQUE idx2 (nCegNev)
 )ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Az azonosító automatikus értékbeállítása csak mezőtulajdonságként adható meg, azonban a kulcs megjelölésére megszorítást is alkalmazhatunk.

```
[CONSTRAINT [szimb_név]] PRIMARY KEY
    [index_típus]
    (indexelt_mező,...)
    [index_opciók]
```

A formális leírás hasonló az egyedi indexeléséhez, de itt nem lehet megadni az index nevét. Ennek az az oka, hogy az elsődleges kulcs neve mindig **PRIMARY**.

Az alábbi példában megszorítással állítjuk be a **nagyker** tábla elsődleges kulcsát.

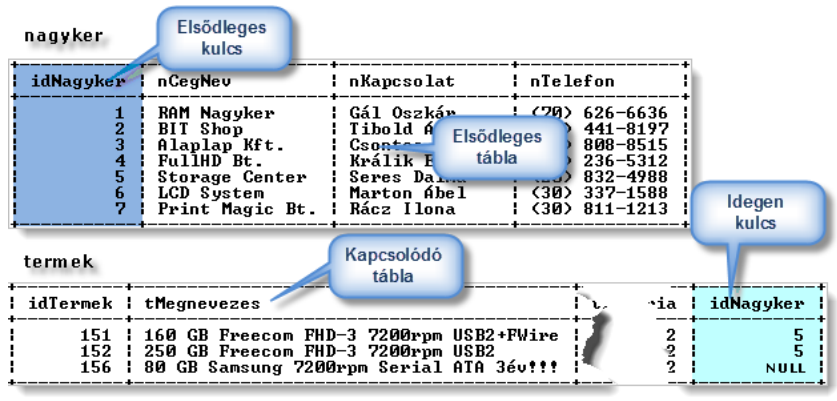
```
CREATE TABLE nagyker (
    idNagyker int(10) UNSIGNED NOT NULL
                      AUTO_INCREMENT,
    nCegNev varchar(50)  DEFAULT NULL,
    nKapcsolat varchar(30) DEFAULT NULL,
    nTelefon varchar(30) DEFAULT NULL,
    INDEX idx1 (nKapcsolat),
    CONSTRAINT  UNIQUE idx2 (nCegNev),
    CONSTRAINT  PRIMARY KEY (idNagyker)
 )ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

9.7 IDEGEN KULCSOK

Az egyedintegritás mellett a **hivatkozási integritás** volt a relációs adatmodell másik fontos integritási szabálya. Mint tudjuk, a kapcsolatokat idegen kulcsokkal írjuk le. A kapcsolódó táblában akkor számít helyesnek az idegen kulcs értéke, ha az az elsődleges tábla létező rekordjára hivatkozik, vagy NULL értékű, azaz nem hivatkozik egy rekordra sem.

*Vegyük példaként a **nagyker-termek** kapcsolatot. A **nagyker** az elsődleges, a **termek** pedig a kapcsolódó tábla, hiszen utóbbi idegen kulccsal hivatkozik az előbbire.*

*A **termek** tábla **idNagyker** idegen kulcsának helyes értékei a **nagyker** tábla egy-egy létező rekordjának azonosító értékét tartalmazzák.*



50. ábra nagyker-termek kapcsolat

A fenti ábra konzisztens állapotot mutat. A kapcsolódó **termek** *tábla két első rekordjának idegen kulcsában (***idNagyker***) az 5-ös érték található. Ezek helyes idegen kulcsok, hiszen mindkettő a létező* **Storage Center** *nevű cég rekordjára hivatkozik az elsődleges táblában (***nagyker***). A* **termek** *tábla harmadik rekordja szintén nem sérti a hivatkozási integritást, hiszen az* **idNagyker** *mező* **NULL** *értéket tárol. Ez azt jelzi, hogy ez a termék nem kapcsolódik egyik nagykereskedéshez sem, „csak úgy van". Utóbbi jelenség sajnos egyáltalán nem ritka a kereskedelmi gyakorlatban, egy esetleges adóellenőrzéskor azonban kínos pillanatokat eredményezhet. Tegyük fel, hogy a NULL értéket 2-re javítjuk. A konzisztens állapot megmarad, nem sértjük hivatkozási integritást, hiszen ismét létező nagykereskedésre hivatkozunk (***BIT Shop***). Ha azonban figyelmetlenségből 20-as értéket írnánk az idegen kulcsba, sérülne a hivatkozási integritás, az adatbázis inkonzisztenssé válna, ellentmondást tartalmazna, hiszen 20-as azonosítójú nagykereskedés nincsen a* **nagyker** *táblában.*

Látjuk, hogy a konzisztens állapot könnyen fölborítható. Ráadásul nemcsak hibás idegen kulcs megadásával, hanem ettől kevéssé nyilvánvaló módon is megsérthetjük a hivatkozási integritást.

Gondoljuk el, mi történne, ha a fenti ábrán látható állapot úgy változnak meg, hogy törölnénk a **Storage Center** *nevű nagykereskedés rekordját. Az idegen kulcsot tartalmazó táblát nem bántottuk, mégis sérül a hivatkozási integritás, hiszen az első két termék már nem létező nagykereskedésre hivatkozik. Hasonló hiba állna elő, ha nem törölnék a Storage Centert, csupán 5-ről 15-re változtatnánk azonosítóját.*

Lássuk, hogyan lehetne rávenni a MySQL-t arra, hogy ne engedje meg a hibás idegen kulcsok keletkezését, azaz ne legyen hajlandó végrehajtani olyan műveleteket, amelyek nyomán sérül a hivatkozási integritás.

9.7.1 A hivatkozási integritás megőrzése

Mielőtt a hivatkozási integritás megőrzésének előírásához szükséges nyelvi elemeket megismernénk, jegyezzük meg, hogy az idegen kulcsok létrehozása

minden táblamotor esetén lehetséges, de a hivatkozási integritás megőrzését csak az InnoDB táblamotor garantálja.

A hivatkozási integritás megőrzéséhez világosan tudatni kell az adatbáziskezelő rendszerrel, hogy **melyek egy kapcsolódó tábla idegen kulcsai**, és **melyik tábla mely mezőjére hivatkoznak**. Azt is közölnünk kell, hogy **mi történjen, ha az elsődleges táblában** végzett **művelet sérti a hivatkozási integritást**.

Példánkban két idegenkulcs is szerepel a ***termek*** *táblában. Az egyik a* ***nagyker*** *tábla* ***idNagyker*** *mezőjére, a másik a* ***kategoria*** *tábla* ***idKategoria*** *mezőjére hivatkozik.*

A hivatkozási integritás megőrzését a kapcsolódó tábla definíciójában elhelyezett megszorítással érhetjük el.

```
[CONSTRAINT [szimb_név]] FOREIGN KEY
    [index_név]
    (indexelt _mező,...)
    hivatkozás_definíció
```

A fenti formális leírás értelmezéséhez tudnunk kell, hogy a MySQL automatikusan indexeli az idegen kulcsokat, tehát amikor más táblákra hivatkozó mezőket hozunk létre, egyben indexet is készítünk. Ez az oka annak, hogy a megszorítás megfogalmazása nagyon hasonlít az elsődleges kulcs, vagy az egyedi index megadásához.

Az *indexelt _mező* helyén szerepelhetne az *idegen_kulcs* jelölés is, hiszen a kerek zárójelek között az idegen kulcs mező nevét kell megadni.

A lényeg a hivatkozás definíciójában lesz. Itt kell megneveznünk a hivatkozott mezőt, és itt kell jeleznünk, hogy mi történjen, ha sérül az integritás.

```
REFERENCES tábla_név (mező_név,...)
    [ON DELETE CASCADE|SET NULL|NO ACTION]
    [ON UPDATE CASCADE|SET NULL|NO ACTION]
```

A hivatkozás definíciójában szereplő jelölések jelentése az alábbi:

- *tábla_név*: az hivatkozott, elsődleges tábla neve.
- *mező_név*: a mező, amire az idegen kulcs hivatkozik. Általában a hivatkozott tábla elsődleges kulcsa, de mindenképpen kulcsjelölt.

- Az **ON DELETE**, és **ON UPDATE** opciók megadása nem kötelező. Azt jelölik, mi történjen, ha az elsődleges táblából törlünk egy hivatkozott rekordot, vagy megváltoztatjuk azonosítójának értékét.
  A **CASCADE** jelentése, az hogy a változtatás történjen meg a kapcsolódó táblában is: vagyis ha az elsődleges táblából töröltünk egy hivatkozott rekordot, akkor törlődjön a kapcsolódó tábla hivatkozó rekordja is.
  Ha módosítjuk a hivatkozott rekord azonosítóját, az idegen kulcs értéke automatikusan módosuljon az új elsődleges kulcs értékére.
  A **SET NULL** esetén az idegenkulcs **NULL** értékékű lesz.
  A **NO ACTION** az alapértelmezett beállítás. Ilyenkor az adatbáziskezelő rendszer egyszerűen megtagadja a hivatkozási integritást sértő művelet végrehajtását.

Mezők típuskompatibilitása

Ahhoz, hogy a MySQL-ben hivatkozó idegen kulcsokat és hivatkozott kulcsjelölteket kapcsoljunk össze, **a kapcsolódó mezőknek azonos domainbe kell tartozniuk**. Ez azt jelenti, hogy típusuknak és a domaint befolyásoló összes mezőtulajdonságaiknak (számoknál például UNSIGNED, szövegeknél karakterszám) is azonosnak kell lenni. Ezt az előírást nevezzük **típuskompatibilitás szabályának**.

Ezek után lássunk két példát a **nagyker-termek** és a **kategoria-termek** kapcsolat leírásán keresztül. Mivel mindkét kapcsolatban a **termek** a kapcsolódó tábla, az idegen kulcsokat itt kell elhelyezni.

Legfölül emlékeztetőként a **kategoria** és a **nagyker** táblák modelljei:


```
CREATE TABLE IF NOT EXISTS kategoria
```


```
(
```


```
idKategoria    INT UNSIGNED
```


```
                    NOT NULL
```


```
                    PRIMARY KEY
```

```
                    AUTO_INCREMENT,
```

```
 kNev  VARCHAR(30) COMMENT 'A kategória neve'
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE nagyker (
```

```
    idNagyker int(10) UNSIGNED NOT NULL
```

```
                        AUTO_INCREMENT,
```

```
    nCegNev varchar(50)  DEFAULT NULL,
```

```
    nKapcsolat varchar(30) DEFAULT NULL,
```

```
    nTelefon varchar(30) DEFAULT NULL,
```

```
    INDEX idx1 (nKapcsolat),
```

```
    CONSTRAINT  UNIQUE idx2 (nCegNev),
```

```
    CONSTRAINT  PRIMARY KEY (idNagyker)
```

```
 )ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE termek (
```

```
  idTermek    int(10)    UNSIGNED    NOT    NULL
AUTO_INCREMENT,
```

```
  tMegnevezes varchar(50) NOT NULL,
```

```
  tBeszerAr int(10) UNSIGNED DEFAULT NULL,
```

```
  tEladAr int(10) UNSIGNED DEFAULT NULL,
```

```
  tKeszlet int(10) UNSIGNED DEFAULT '0',
```

```
  tSzukseges int(10) UNSIGNED,
```

```
  idKategoria int(10) UNSIGNED NOT NULL,
```

```
  idNagyker int(10) UNSIGNED NOT NULL,
```

```
  CONSTRAINT PRIMARY KEY (idTermek),
```

```
  CONSTRAINT fk_1 FOREIGN KEY
```

```
      (idKategoria)
```

```
        REFERENCES kategoria (idKategoria)
        ON DELETE NO ACTION ON UPDATE NO
ACTION,
    CONSTRAINT fk_2 FOREIGN KEY
        (idNagyker)
        REFERENCES nagyker (idNagyker)
        ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

9.7.2 Kapcsolatok tulajdonságai

1:n kapcsolat

Az idegen kulcsokkal ábrázolt kapcsolatok számosságát az jelzi, hogy a kapcsolódó táblában ismétlődhet-e az idegen kulcs egy értéke. Ha az idegen kulcs nem egyedi indexelésű, akkor a kapcsolat 1:n.

1:1 kapcsolat

1:1 kapcsolatot úgy hozhatunk létre, hogy a kapcsolódó táblában egyedi indexeléssel (UNIQUE) biztosítjuk, hogy az idegen kulcs értékei ne ismétlődhessenek.

Parciális részvétel

Azok a rekordok, amelyekben az idegen kulcs **NULL** értéket tartalmaz, nem vesznek részt a kapcsolatban. Ha a kapcsolódó táblában megengedett az idegen kulcs **NULL** értéke, akkor a **tábla részvétele parciális**. Ha totális részvételt akarunk előírni, a **NOT NULL** mezőtulajdonsággal tiltanunk kell az idegen kulcs **NULL** értékét!

A fenti példában mindként idegen kulcsnál jeleztük, hogy a ***termek*** *tábla részvétele totális, azaz minden termék kötelezően kapcsolódik szállítóhoz, és kategóriához is.*

9.8 NÉZETEK A MYSQL-ADATBÁZISOKBAN

A relációs adatmodell ismertetésekor esett szó a relációs adatbázisok sajátos strukturális elemeiről, a származtatott relációkról, vagy más néven a **nézetekről**. Mint említettük, a nézetek egy vagy több egymáshoz kapcsolódó reláció, rekord és mező halmazából kiválasztott részhalmazok. Alkalmasak arra, hogy a különböző táblákban tárolt adatokat közös, **virtuális táblában** jelenítsük meg a segítségükkel. A következő fejezetben ismertetésre kerülő jogosultsági rendszerrel együtt alkalmazva a MySQL nézetei lehetővé teszik, hogy a különböző felhasználók számára ugyanannak az adatbázisnak más-más részeit tegyük láthatóvá. A nézetek készítését a **CREATE VIEW** parancs biztosítja, amelynek egyszerűsített változatát a következő formai leírás mutatja be:

```
CREATE
    [DEFINER = { felhasználó | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW nézet_neve [(oszlop_lista)]
    AS választó_lekérdezés
```

A *nézet_neve* opció a származtatott reláció neve, amellyel a felhasználók a DML- és DQL-mondatokban táblaként hivatkozhatnak a nézetre.

A *választó_lekérdezés* az a DQL-mondat, amely a bázistáblák rekordjainak részhalmazát állítja elő. Az *oszlop_lista* az eredményreláció egyes mezőinek felsorolására való. A nézet használója csak ezeket az oszlopokat láthatja majd. Az oszloplistát általában elhagyjuk, így a nézet használója előtt a választó lekérdezés eredményének minden mezője megjelenik. A **DEFINIER** záradék szabályozza, hogy DBMS melyik felhasználót tekinti a nézet létrehozójának. Az **SQL SECURITY** álltja be azt, hogy a nézet használója saját, vagy a **DEFINIER** jogosultságaival dolgozhasson a virtuális táblával.

Tegyük fel, hogy szeretnénk egy olyan virtuális táblát, amely csak a ***Storage Center*** *által szállított termékeket tartalmazza. A nézet az alábbi módon hozható létre:*

```
CREATE VIEW storage
```

```
AS

SELECT* FROM termek WHERE idNagyker=5;
```

Tananyagunkban a csupán a DDL parancsaival foglalkoztunk, azonban az itt látható egyszerű DQL-mondat biztosan nem akadályozza az olvasót a **storage** nevű nézet létrehozásának megértésében.

A választó lekérdezés azokat a rekordokat válogatja ki a **termek** táblából, amelyekben az **idNagyker** mező értéke egyenlő a **Storage Center** azonosítójával (5). A **SELECT** parancsot követő csillag (*****) azt jelenti, hogy a kiválasztott rekordok összes mezője kerüljön be az eredménybe.

Miután a fenti view-t létrehoztuk, a felhasználók táblaként hivatkozhatnak a ***storage*** *nézetre.*

```
mysql> select * from storage;
+-----------+-----------------------------------------+-------------+------------+
| idTermek  | tMegnevezes                             | idKategoria | idNagyker  |
+-----------+-----------------------------------------+-------------+------------+
|       151 | 160 GB Freecom FHD-3 7200rpm USB2+FWir  |           2 |          5 |
|       152 | 250 GB Freecom FHD-3 7200rpm USB2       |           2 |          5 |
|       215 | DVD+-RW MSI DR16-B2 dob. Double Layer   |           6 |          5 |
|       216 | DVD+-RW Freecom FS-50 Slim USB2 8x DL 2 |           6 |          5 |
|       220 | DVD+-RW Plextor PX-712SA Serial ATA !!! |           6 |          5 |
|       221 | DVD+-RW Plextor PX-716A DL OEM          |           6 |          5 |
+-----------+-----------------------------------------+-------------+------------+
```

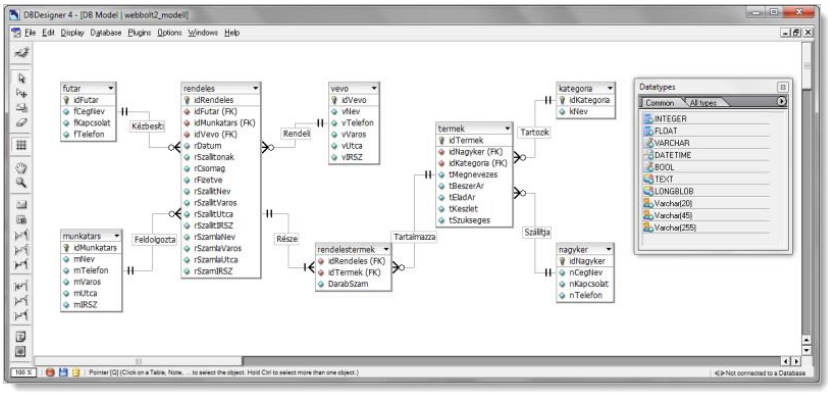
51. ábra STORAGE nézet használata

9.9 CASE-ESZKÖZÖK

Adatbázisok különböző szintű modellezésére számos alkalmazást fejlesztettek már. Ezek egy része nem csupán a grafikus reprezentáció elkészítésére, hanem a modell fizikai leképezésére is alkalmas. Az ilyen szoftvereket sorolják a **Computer-aided Software Engineering** kifejezésből rövidített **CASE Tools** kategóriába.

Mielőtt ilyen szoftverrel próbálunk dolgozni, föl kell elevenítenünk, hogy az ER-koncepcionális modell. Sokan, köztük Codd, a relációs adatmodell megalkotója is keveslik azokat a megszorításokat, amelyeket az ER kialakítása közben alkalmazhatunk. Nincs lehetőség például az attribútumok adattípusának leírására, a hivatkozási integritással kapcsolatos, a NULL, vagy alapértelmezett értékekre vonatkozó megszorítások megfogalmazására, vagy például a származtatott mezők számítási módjának megadására.

Éppen a megszorítási lehetőségek hiánya az oka annak, hogy a CASE-eszközök nem a koncepcionális ER-modell eredeti, Chen-féle változatát használják. Ehelyett a logikai szinthez jóval közelebb álló modellből indulnak ki. Elvárják a tulajdonságok adattípusának precíz meghatározását, sőt néhány a fizikai modellezéshez kapcsolódó megszorítás megfogalmazására is lehetőséget biztosítanak.



52. ábra Adatbázis terve a DVDesigner4 felületén

Az ilyen eszközök használatakor le kell mondanunk ugyan az ER néhány igazán szép lehetőségéről, például a nem atomi attribútumok használatáról, vagy a kapcsolatokhoz kötődő tulajdonságokról, cserébe viszont fantasztikus lehetőséget kapunk. Az elkészített modell alapján a CASE-eszköz automatikusan elkészíti a fizikai adatbázist, ami ezek után azonnal kész az adatok fogadására.

Ilyen CASE-eszköz a MySQL-adatbázisok tervezésére és létrehozására is alkalmas, ingyenesen használható DBDesigner4, illetve az ennek továbbfejlesztéseként felfogható MySQL Workbench.

http://www.mysql.com/downloads/workbench/

3. link A MySQL Workbench letöltése

9.10 ÖSSZEFOGLALÁS, KÉRDÉSEK

9.10.1 Összefoglalás

Mai leckénkkel az adatbázis tervezéssel és modellezéssel foglalkozó anyagrész végére értünk. Megtanultuk, hogyan valósíthatók meg a fizikai modellezés szintjén a logikai modellben használt strukturális elemek, az azokkal kapcsolatos integritási szabályok, és hogyan implementálható az adatbázis fizikai modellje.

Megismertük a **`CREATE TABLE`** parancs indexek, elsődleges kulcsok és idegen kulcsok megvalósítására használható nyelvi elemeit.

Megismertük az indexelés jelentőségét, elsajátítottuk az egyedi és nem egyedi indexek létrehozásának módszerét. Összehasonlítottuk az indexeket és az elsődleges kulcsot, megtanultuk, hogyan lehet automatikus számozású kulcsokat készíteni.

Leckénk befejező részben a táblák közötti kapcsolatok definiálásával, és a hivatkozási integritás megőrzésével foglalkoztunk.

Megtanultuk a típuskompatibilitás jelentőségét, az idegen kulcsok hivatkozásainak pontos leírását. Leckénk végére nemcsak a relációs adatmodell strukturális elemeinek leképezésére, hanem az egyed-, és hivatkozás integritás megőrzésének fizikai modellezésére is képessé váltunk

9.10.2 Önellenőrző kérdések

1. Miért célszerű az InnoDB táblamotor használata?
    - Azért, mert ez a táblamotor teszi lehetővé a tranzakciókezelést, és a hivatkozási integritás megőrzését is.
2. Mik az indexelés előnyei és hátrányai?
    - A indexelés hatására bázistáblában gyorsabb lesz az indexelt mező szerinti keresés és sorba rendezés, de nő az adatbázis mérete, és lassulnak a bázistábla rekordkezelő műveletei.
3. Miben hasonlít az elsődleges kulcs, és az egyedi indexelés?
    - Az elsődleges kulcs beállítása valójában egyenértékű egyedi indexeléssel, de míg az egyedi indexelés önmagában megengedi a NULL értéket, addig az PRIMARY KEY beállítás nem.

4. Mit jelent a típuskompatibilitás?
    - Azt, hogy az idegen kulcsnak azonos domainbe kell tartoznia a hivatkozott kulcsjelölttel.
5. Mi történik, ha beállítottuk egy idegen kulcs hivatkozását, de nem adtunk meg az ON DELETE opciót, majd törlünk egy hivatkozott rekordot az elsődleges táblából?
    - A MySQL nem hajtja végre a törést, hanem hibaüzenetet küld.

10. LECKE: ADATBIZTONSÁG KÉRDÉSEI, A MYSQL JOGOSULTSÁGI RENDSZERE

10.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

Az adatbázis-tervezés és -modellezés célja, hogy a modell alapján készíthető adatbázis eleget tegyen az adatbázis-kezeléssel szemben támasztott legalapvetőbb elvárásoknak. Adatbázisainknak biztosítanunk kell az adatok perzisztens, konzisztens és hatékony tárolását, gyors és egyszerű visszakereshetőségét.

Már a kőtáblák és Mózes története rávilágított azonban arra, hogy a felhasználó tevékenysége nyomán a legnagyobb körültekintés mellett megtervezett szerkezetben és a legbiztonságosabbnak vélt adathordozón tárolt adatok is megsérülhetnek, sőt, meg is semmisülhetnek. Az adatbiztonság megvalósításának fontos, de csak egyik összetevője a megfelelő tervezés és implementáció. Emellett azonban legalább ekkora hangsúlyt kell fektetni az adatok felhasználói tevékenységgel szembeni védelmére, és az óvintézkedések ellenére bekövetkező adatvesztés okozta károk elháríthatóságra.

Az adatok konzisztens és perzisztens tárolásában a tervezés után a felhasználói jogosultságok megfelelő szabályozása, a konzisztens állapotok rendszeres mentése, illetve visszaállíthatóságuk biztosítása játssza a legfontosabb szerepet.

Nem véletlen, hogy a modern adatbázis-kezelő rendszerek legfontosabb összetevői közé tartozik a jogosultsági, illetve a mentés/visszaállítási alrendszer.

Most következő leckénkben a MySQL korábban már említett jogosultságairól, a jogok kiosztásáról, és tárolásáról fogunk beszélni, a következő leckében pedig, a mentés/visszaállítási rendszer lehetőségeit ismerjük meg.

A 10. lecke megtanulása után, Ön ismerni fogja a MySQL kétlépéses jogosultság-ellenőrzését, a több szinten biztosítható jogosultságokat, és a jogok kiosztásának és megvonásának módját.

10.2 JOGOK ELLENŐRZÉSE

Az adatbázisok bonyolult, **sok objektumból** fölépülő, **több felhasználó** által **hálózatokon** keresztül hozzáférhető rendszerek, amelyekben **számos különböző művelet** elvégzésére van lehetőség.

A jogosultsági rendszernek ennek megfelelően **mind a négy tényezőre** ki kell térnie. Amikor egy jogot meghatározunk, világosan le kell fektetni, hogy az adatbázis mely **objektumára** vonatkozó, és milyen **művelet** elvégzésére biztosítottunk lehetőséget a **felhasználó** számára, sőt, arra is ki kell térnünk, hogy a művelet melyik **számítógépről** tesszük lehetővé.

A MySQL-lel végzett munka a következő lépésekben történik:

- A felhasználó elindítja a mysql klienst.
- Megadja felhasználói nevét és jelszavát.
- A kliens kapcsolatot épít fel a kiválasztott szerverrel és ellenőrzésre elküldi a felhasználó adatait, valamint annak hostcímét.
- A szerver megvizsgálja az adatokat és azonosítja a felhasználót.
- Ezt követően létrejön a felhasználó munkamenete, amely a kijelentkezésig tart.
- A munkamenet felhasználója SQL-mondatokat küldhet a DBMS-nek, amelyek a szerver által felügyelt különböző objektumokra vonatkozó műveleteket írnak le.
- A DBMS minden parancs esetén megvizsgálja, a műveletet, és azt, hogy melyik objektumra vonatkozik.
- Ezt követően ellenőrzi, hogy bejelentkezett felhasználó rendelkezik-e az objektumon a művelet elvégzését biztosító jogosultsággal.
- Ha igen, akkor elvégzi a műveletet, ellenkező esetben megtagadja a végrehajtást.
- A felhasználói munkamenet befejeződik, amikor a felhasználó bezárja a klienst.

A felhasználó által megadott művelet elvégezhetőségét többszintű jogosultsági rendszerrel és a jogok kétlépéses ellenőrzésével biztosítja a MySQL.

Az ellenőrzés első lépése a felhasználó azonosítása, a második pedig az egyes műveletekre vonatkozó jogok vizsgálata.

10.3 KÉTLÉPÉSES ELLENŐRZÉS

10.3.1 Felhasználók azonosítása

A jogosultsági rendszer adatait, a felhasználói tulajdonságokat, és a jogosultságokat a különálló **`mysql`** adatbázisban tárolják a MySQL adatbázis-kezelő rendszerek. Ez az adatbázis a szerver telepítésekor automatikusan létrejön, és folyamatosan tárolja a jogosultságok változásait.

A MySQL minden egyes felhasználóhoz **felhasználói fiókot**, **account**ot rendel, amit a **`mysql`** adatbázis **`user`** táblája tárol.

A több adatbázist is felügyelő adatbázis-kezelő rendszerekben *adatbázis*`.`*tábla* formátumú, úgynevezett **minősített hivatkozásokkal** hivatkozhatunk az egyes adatbázisok tábláira. A minősített hivatkozásban előbb az adatbázis, majd tőle ponttal elválasztva a hivatkozott tábla nevét adjuk meg.
Például:
`webbolt.termek`

A **`mysql.user`** tábla egy-egy felhasználói accountot tárol rekordonként. A tábla összetett kulcsát a **`User`** és **`Host`** mezők alkotják, amelyek egyben az **felhasználói fiók azonosítóját** is jelentik. A **`User`** mező a felhasználó nevét, a **`Host`** pedig a kapcsolódáshoz használható gép címét tartalmazza. A felhasználói fiókhoz tartozó jelszó a **`Password`** mezőben tárolódik.

A MySQL-ben egy felhasználó neve különböző gépcímekkel párosítva több account-ban is szerepelhet. A név és a hozzáféréshez használt gépcím ugyanis együtt azonosítja a fiókot. Ebből adódóan a **jogosultságok** sem felhasználói névhez, hanem **a teljes fiókhoz kapcsolódnak.**

Amikor a felhasználó elindítja a kliens programját, majd kapcsolódni próbál a szerverhez, a kliens elküldi a szervernek saját hostja címét, a felhasználó nevét és begépelt jelszavát. A MySQL ellenőrzi, hogy a **`mysql.user`** táblában tartalmaz-e olyan rekordot, amely a kapott felhasználói nevet és hostcímet tárolja. Ha igen, akkor megvizsgálja a jelszó helyességét is. Hibás felhasználói adatok esetén visszautasítja a kapcsolatot és hibaüzenetet küld:

```
mysql -u betolakodo
ERROR 1045 (28000):
A(z) 'betolakodo'@'localhost' felhasznalo szamara
tiltott eleres. (Hasznalja a jelszot: NEM)

vagy:

ERROR 2003 (HY000): A '10.4.2.2' host szamara nem
engedelyezett a kapcsolodas ehhez a MySQL szerverhez
```

Ha az adatok rendben vannak, létrejön a **munkamenet**, az úgynevezett session, ami felhasználó a kilépéséig tart.

A felhasználó belépésekor lezajló ellenőrzés a kétlépéses folyamat első fázisa.

10.3.2 Műveleti jogosultságok vizsgálata

A bejelentkezést követően a MySQL minden egyes SQL-mondat végrehajtása előtt ellenőrzi, hogy a munkamenethez kapcsolódó felhasználói fiók rendelkezik-e a művelet végrehajtásához szükséges joggal.

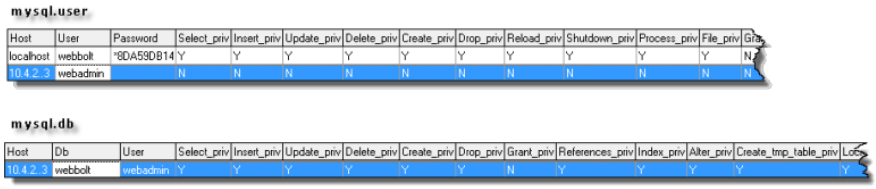
A jogosultságok vizsgálata az ellenőrzés második lépése.

10.4 TÖBBSZINTŰ JOGOSULTSÁGI RENDSZER

Az adatbázis-kezelő rendszerek szinte mindig több adatbázist tárolnak, amelyek külön-külön is több táblából épülnek fel. A táblák a tárolt egyedtípus attribútumainak megfelelően több mezőt tartalmaznak. A különböző strukturális elemek (szerver, adatbázis, tábla, mező) többszintű hierarchiába rendeződnek. A jogosultsági rendszer ezekhez a szintekhez, illetve a szintek objektumaihoz kötődik.

A jogok szabályozhatók a szerver, az adatbázisok, a táblák, sőt, akár a mezők szintjén is. Amikor egy adott szinten álló objektumra jogot biztosítunk egy fiók számára, akkor az engedély az objektumhoz kapcsolódó, alacsonyabb szintű objektumokra is érvényes lesz. Ha például a teljes szerverre, azaz minden adatbázisra kapunk **`CREATE`** jogot, akkor bármely, még a később létrehozott adatbázisokban is létrehozhatunk majd táblákat. Ha a **`CREATE`** jogot nem a szerver, hanem az adatbázisok szintjén, egy bizonyos adatbázisra kapjuk meg, akkor csak abban készíthetünk táblákat.

A különböző szintekhez kapcsolódó jogosultságok a **`mysql`** adatbázis **`user`**, **`host`**, **`db`**, **`tables_priv`**, **`columns_priv`**, és **`proc_priv`** tábláiban tárolódnak. Mindegyik tábla felhasználói fiókot és a szint egy objektumát azonosító, valamint a szinten beállítható jogokat leíró mezőket tartalmaz. A jog meglétét mezők 'Y', vagy 'N' betűi jelzik. A legfelső, globális jogosultságok szintjét a **`user`** tábla írja le.



mysql.user

Host	User	Password	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv	Shutdown_priv	Process_priv	File_priv	Gra
localhost	webbolt	*8DA59DB14	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N
10.4.2..3	webadmin		N	N	N	N	N	N	N	N	N	N	N

mysql.db

Host	Db	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Grant_priv	References_priv	Index_priv	Alter_priv	Create_tmp_table_priv	Lo
10.4.2..3	webbolt	webadmin	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y

53. ábra A mysql.user és mysql.db táblák részletei

A fenti példa szerint a **`10.4.2.3`** *hostról bejelentkező* **`webadmin`** *felhasználó nem rendelkezik a szerver egészére kiterjedő jogokkal, de a* **`webbolt`** *adatbázisban szinte mindent megtehet.*

10.4.1 Jogok

A MySQL-ben különböző jogok, úgynevezett privilégiumok kapcsolhatók a fiókokhoz, és a különböző objektumokhoz. A jogokat a GRANT paranccsal adhatja meg a root, vagy az erre jogosult más felhasználó.

Az alábbi táblázat a teljesség igénye nélkül foglalja össze a megadható jogosultságokat.

Jogosultság	Jogot leíró mezőnév	Szint
CREATE	Create_priv	adatbázis, tábla vagy index
DROP	Drop_priv	adatbázis, tábla vagy nézet
GRANT OPTION	Grant_priv	adatbázis, tábla vagy tárolt eljárás
REFERENCES	References_priv	adatbázis vagy tábla
EVENT	Event_priv	adatbázis
ALTER	Alter_priv	tábla
DELETE	Delete_priv	tábla
INDEX	Index_priv	tábla
INSERT	Insert_priv	tábla vagy mező
SELECT	Select_priv	tábla vagy mező
UPDATE	Update_priv	tábla vagy mező

Jogosultság	Jogot leíró mezőnév	Szint
CREATE TEMPORARY TABLES	Create_tmp_table_priv	tábla
LOCK TABLES	Lock_tables_priv	tábla
TRIGGER	Trigger_priv	tábla
CREATE VIEW	Create_view_priv	nézet
SHOW VIEW	Show_view_priv	nézet
ALTER ROUTINE	Alter_routine_priv	tárolt eljárás
CREATE ROUTINE	Create_routine_priv	tárolt eljárás
EXECUTE	Execute_priv	tárolt eljárás
FILE	File_priv	fájlkezelés a szerver hostján
CREATE USER	Create_user_priv	szerver
PROCESS	Process_priv	szerver
RELOAD	Reload_priv	szerver
REPLICATION CLIENT	Repl_client_priv	szerver
REPLICATION SLAVE	Repl_slave_priv	szerver
SHOW DATABASES	Show_db_priv	szerver
SHUTDOWN	Shutdown_priv	szerver
SUPER	Super_priv	szerver
ALL [PRIVILEGES]		szerver
USAGE		szerver

10.5 DDL-MŰVELETEKHEZ KÖTŐDŐ JOGOK

10.5.1 USAGE, SHOW DATABESES

A **`USAGE`** jog lehetővé teszi, hogy az account felhasználója munkamenetet alakítson ki, de önmagában nem engedi az adatok kezelését. Ez a jog csak globális szinten létezik, így kizárólag a **`mysql.user`** táblában tárolódik.

A **`SHOW DATABASES`** jog birtokában már listázhatók a szerveren tárolt adatbázisok is.

10.5.2 CREATE, ALTER, DROP, CREATE TEMPORARY TABLES

A **CREATE**, **ALTER**, **DROP** jogok rendre az adott objektum létrehozását, szerkezetének megváltoztatását, illetve az objektum törlését teszik lehetővé. A **CREATE TEMPORARY TABLES** jog birtokában a felhasználó ideiglenes táblákat hozhat létre.

10.5.3 INDEX

Táblák utólagos indexelésének joga. Akkor lehet rá szükség, ha egy felhasználó nem hozhatja létre, és nem is változtathatja meg a táblát, de megengedjük neki, hogy indexelje a meglévőt.

10.5.4 CREATE VIEW, SHOW VIEW

Nézetek létrehozását és a létrehozásukhoz használt SQL-mondat megtekintését biztosító jogok. A nézetek, mint tudjuk, egy vagy több tábla rekordjaiból és mezőiből képzett részhalmazok, származtatott táblák.

10.5.5 EVENT, TRIGGER

Események, illetve triggerek létrehozását és törlését lehetővé tévő jog.

10.5.6 CREATE ROUTINE, ALTER ROUTINE, EXECUTE

Tárolt eljárások létrehozását, megváltoztatását, és futtatását biztosító jogok.

10.6 JOGOSULTSÁGOK BIZTOSÍTÁSÁHOZ SZÜKSÉGES JOGOK

10.6.1 CREATE USER

A **CREATE USER** új felhasználók felvételére jogosít fel, de lehetővé teszi a **DROP USER**, **RENAME USER**, és a **REVOKE ALL PRIVILEGES** parancsok végrehajtását is. Ezek a felhasználó törlését, átnevezését, és jogainak teljes megvonását eredményezik.

10.6.2 GRANT OPTION

A **GRANT OPTION** jog birtokosai megadhatják, vagy megvonhatják más felhasználóktól azokat a jogokat, amelyekkel ők maguk is rendelkeznek.

10.6.3 ALL, ALL PRIVILEGES

Az adott objektumra lehetséges összes jog megadását jelenti.

Általában ezt használjuk, ha egy felhasználónak minden jogot meg akarunk adni egy adatbázisra.

```
GRANT ALL PRIVILEGES ON webbolt.* TO
'webboltadmin'@'localhost';
```

10.7 DQL- , DML-MŰVELETEKHEZ SZÜKSÉGES JOGOK

10.7.1 SELECT

A legelemibb műveleteket, a DQL-parancsok végrehajtását teszi lehetővé. A **SELECT** jog birtokában a felhasználó lekérdezheti a táblákban tárolt adatokat.

10.7.2 INSERT, UPDATE, DELETE

Új rekord felvételét, rekordok módosítását, illetve törlését biztosító jogok. Birtokukban használhatók a DML résznyelv parancsai, az **INSERT**, az **UPDATE** és a **DELETE**.

10.7.3 LOCK TABLE

A táblák zárolására használt LOCK TABLE parancs futtatását teszi lehetővé. A zárolás alatt álló táblát más felhasználók nem módosíthatják.

10.8 FELHASZNÁLÓK FÖLVÉTELE

A MySQL szerver egy új felhasználói fiókjának létrehozása, egyben a **USAGE** jog hozzárendelése a már ismerős **CREATE** paranccsal végezhető.

```
CREATE USER felhasználó_definíció
[,felhasználó_definíció …]

felhasználó_definíció:
felhasználói_név [IDENTIFIED BY [PASSWORD] 'jel-
szó']
```

A paranccsal több felhasználó is fölvehető. Minden felhasználói definíció felhasználói névből áll, amelyhez megadható a felhasználó jelszava.

A jelszó megadása opcionális, de a legalapvetőbb biztonsági intézkedések közé tartozik, hogy egyetlen felhasználó bejelentkezését sem engedélyezzük jelszó nélkül.

```
CREATE USER webboltadmin IDENTIFIED BY
'titkos';
```

Ahogyan az látható, a `CREATE USER` nem rendel hostot a fiókhoz. Ez azonban csak látszólag van így. A parancs speciális hostnevet, a '%' jelet rendeli fiókhoz, ami azt jelzi, hogy az így létrehozott felhasználó bárhonnan beléphet. Természetesen ezen kívül csak a `USAGE` joggal rendelkezik, tehát nem csinálhat semmit.

Ha pontosítani szeretnénk, hogy a létrehozott fiókhoz milyen host tartozik, az elektronikus levelezésnél megszokott formátumban adhatjuk meg a *név@host* párost.

```
CREATE USER webboltadmin@localhost
```

```
    IDENTIFIED BY 'titkos';
```

10.9 JOGOSULTSÁGOK BIZTOSÍTÁSA

A MySQL jogokra vonatkozó alapszabályai, hogy a felhasználó nem rendelkezik semmilyen joggal, amíg azt nem kapja meg. Azonban a

jogosultsági hierarchia valamely szintjén megadott jog átterjed az alacsonyabb szintek kapcsolódó objektumaira, mindaddig, amíg egy alárendelt szinten felül nem írjuk a jogot.

A jogosultságokat a **GRANT** paranccsal adhatja meg a **GRANT OPTION** joggal rendelkező felhasználó.

```
  GRANT
    jogosultság [(mezőfelsorolás)]
        [,jogosultság [(mezőfelsorolás)]] ...
    ON [objektumtípus] jogosultsági_szint
    TO felhasználói_név [IDENTIFIED BY [PASSWORD]
'jelszó']
      [,felhasználói_név [IDENTIFIED BY [PASSWORD]
'jelszó']]
      [REQUIRE {NONE | ssl_opció [[AND] ssl_opció]
...}]
      [WITH opciók ...]
```

A parancs általános formai leírása kissé összetett, számos opcionális elemet tartalmaz, ezért egy példán keresztül tesszük egyszerűbbé.

*Az alábbi példa globális **SHOW DATABASES** és **CREATE** jogot biztosít a **localhost**-ról bejelentkező, **webboltadmin** nevű felhasználó számára.*

```
GRANT SHOW DATABASES, CREATE
    ON *.*
    TO webboltadmin@localhost;
```

A **GRANT** parancs után a megfelelő jogosultságok neveit kell felsorolni, az **ON** kulcsszót pedig a jogosultsági szint követi. Itt általában minősített hivatkozást használunk, amelyben csillaggal (*) hivatkozhatunk az adott szint összes objektumára. A *.* minden adatbázis minden tábláját jelenti, tehát valójában globális szintet jelez. A **TO** kulcsszó után jelezzük, hogy a jogokat melyik fiókhoz rendeljük hozzá, azaz ki kapja azokat.

Amennyiben a felhasználó még nem létezik, a **GRANT** előbb létrehozza, majd beállítja jogait. Ebben az esetben mindenképpen használjuk az [**IDENTIFIED BY [PASSWORD]** *'jelszó'*] opciót!

10.10 JOGOK LISTÁZÁSA ÉS MEGVONÁSA

A MySQL szerverek biztonságos használatához elengedhetetlen a felhasználók jogosultságainak biztosítása, de legalább ilyen fontos a jogok áttekintése és szükség esetén visszavonása.

A jogok megjelenítésében hasznos lehet a **SHOW GRANTS** parancs, amely kilistázza azokat az SQL-mondatokat, amelyekkel egy felhasználó éppen érvényes jogai megadhatók lennének.

```
SHOW GRANTS FOR fiók;
```

A korábban megadott jogok visszavonása a **REVOKE** paranccsal történik.

```
REVOKE
    jogosultság [(mezőfelsorolás)]
      [,jogosultság [(mezőfelsorolás)]] ...
    ON [objektumtípus] jogosultsági_szint
    FROM felhasználó [,felhasználó] ...
```

Például:

```
REVOKE ALL PRIVILEGES, GRANT OPTION
    FROM webboltadmin@localhost;
```

A jogaitól megfosztott felhasználó változatlanul kapcsolódhat a szerverhez, ugyanis globális **USAGE** joga megmarad. Ha véglegesen el akarunk távolítani egy felhasználót, akkor a **DROP USER** paranccsal törölhetjük fiókját.

```
DROP USER webboltadmin@localhost;
```

10.11 ÖSSZEFOGLALÁS, KÉRDÉSEK

10.11.1 Összefoglalás

Mai leckénkben az adatbiztonság fontos problémájára, a felhasználók azonosítására és jogosultságaik szabályozására kerestünk megoldásokat.

Megtanultuk, hogy a MySQL kétlépéses ellenőrzéssel vizsgálja a felhasználó jogosultságokat. Első lépésben a felhasználói fiókot, második lépésben pedig az éppen kért művelet jogosságát vizsgálja.

A felhasználók azonosítása nem csupán név alapján történik. A **`mysql.user`** tábla névvel és **`host`** címmel azonosított felhasználói fiókokat tárol. A jogosultságok ezekhez a fiókokhoz rendelhetők.

A jogok több szinten szabályozhatók, így nem csupán a **`mysql.user`** táblában rögzített globális jogok, hanem az objektumszintek jogainak összessége határozza meg egy felhasználó jogosultságait.

A felhasználók és jogaik kezelésére DCL-parancsokat használunk. A **`CREATE USER`** paranccsal új fiókot alakíthatunk ki, a **`GRANT`** paranccsal pedig szintekre és objektumokra specializáltan biztosíthatjuk a jogosultságokat.

A jogok a **`SHOW GRANTS FOR`**… kezdetű mondatokkal listázhatók, és a **`REVOKE`** paranccsal vonhatók meg. A felhasználói fiók végleges eltávolítását a **`DROP USER`** paranccsal végezhetjük el.

10.11.2 Önellenőrző kérdések

1. Mit értünk az alatt, hogy a MySQL kétlépéses jogosultságellenőrzést végez?
    - Azt, hogy belépéskor ellenőrzi a USAGE jogot, majd minden művelet elvégzése előtt ellenőrzi a kérdéses parancshoz kapcsolódó jogot.
2. Mi a lényege a többszintű jogosultsági rendszernek?
    - Az, hogy a jogosultságok teljes rendszer-, adatbázis-, tábla- és mezőszinten lévő objektumokra szabályozhatók. A jogok lefelé terjednek a hierarchiában, tehát ha egy felhasználó megkap valamilyen jogot egy adatbázisra, ezzel akkor minden táblájára is megkapja ugyanazokat a privilégiumokat. A jogok ugyanakkor szintenként felülbírálhatók.
3. Hogy lehetséges, hogy ugyanaz a felhasználó ugyanarra az objektumra eltérő jogokat birtokol?
    - Úgy, hogy a MySQL nemcsak felhasználói név, hanem név és host cím alapján azonosítja a fiókokat. Ha úgy tetszik, egy felhasználónak több felhaszná-

lói fiókja is lehet. Elképzelhető, hogy egyik fiókjához más jogok vannak hozzárendelve, mint a másik fiókhoz.

4. Hogyan adna meg minden jogot a 10.4.2.3 hostról bejelentkező tester nevű felhasználónak a webbolt adatbázis összes táblájára?

- GRANT ALL PRIVILEGES
  ON webbolt.*
  TO tester@10.4.2.3
  WITH GRANT OPTION;

5. Hogyan lehet véglegesen törölni egy felhasználót?

- A REVOKE paranccsal.

11. LECKE: TOVÁBBI ADMINISZTRÁTORI FELADATOK

11.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

11.2 KONZISZTENS ÁLLAPOTOK MENTÉSE

Az adatbázis-kezelő rendszerek felhasználóit a szerver kifogástalan működése esetén vajmi kevéssé érdeklik a biztonsági mentések. Csupán azzal foglalkoznak, hogy mindig elérjék és a lehető legnagyobb sebességgel kezelhessék az adataikat. Amikor azonban bekövetkezik az adatvesztés, minden szem az adminisztrátorra szegeződik. Ha a legutolsó konzisztens állapot visszaállítható, a felhasználók megnyugszanak, de ha nincs megfelelő mentés, a károk keletkezéséért részben vagy kizárólag az adminisztrátort teszik felelőssé. Az adatok sérülését vagy teljes elvesztését számos körülmény okozhatja. Szoftveres hiba, vírusfertőzés, a hardver (például a merevlemez) meghibásodása, áramszünet, emberi figyelmetlenség, legrosszabb esetben a hardver elhelyezésére szolgáló épületben esett kár.

Ha netán kétségeink támadnának a tekintetben, hogy saját adatainkkal előfordulhat-e ilyen esemény, nyugtasson meg minket az a bizonyosság, hogy az adatvesztés előbb vagy utóbb mindenképpen bekövetkezik.

Utólag már hiába temetjük meggyötört arcunkat a billentyűzeten megkérgesedett tenyerünkbe. A konzisztens állapot visszaállításának lehetőségéről, az adatok mentéséről még az első adatvesztés előtt kell gondoskodnunk.

Az adatbázis-szerverek különböző hardver- és szoftverkörnyezetben működnek, terhelésük, használatuk módja eltérő, és a biztonsági mentésekre fordítható anyagi erőforrások sem mindenhol azonosak. Éppen ezért számos különböző mentési és visszaállítási technika alakult ki. Mai leckénkben áttekintjük a biztonsági mentések különböző formáit. A felsorolt mentési technikák mindegyike megvalósítható a MySQL adatbázis-kezelő rendszerekkel is, azok lebonyolítása azonban általában kiemelt jogosultságokat igényel. Leckénkben ezért, az akár korlátozott, csak néhány adatbázisra, esetleg táblára vonatkozó jogok mellett is használható mentési technikával, a **`mysqldump`** segédprogrammal megvalósítható, logikai mentéssel foglalkozunk részletesebben.

Leckénk elolvasása után Ön képes lesz saját adatbázisainak, vagy azok kiválasztott tábláinak biztonsági mentésére, és az adatok helyreállítására.

11.3 TELJES ÉS INKREMENTÁLIS MENTÉS

A teljes és inkrementális mentések abban különböznek egymástól, hogy az adatbázis-kezelő rendszerrel kezelt adatbázisok adatainak egészét, vagy csak azoknak az utolsó teljes mentés óta megváltozott részét mentjük-e.

11.3.1 Teljes mentés

A teljes mentés esetén értelemszerűen **minden** releváns **adatról biztonsági mentést készítünk**. Az adatbázis-kezelő rendszerek esetében teljes mentésről beszélünk, ha a program által kezelt összes adatállományt, az adatbázis fájlokat, és a konfigurációs állományokat lemásoljuk.

A teljes mentés tökéletes visszaállíthatóságot garantál, azonban elkészítése időigényes, és jelentős háttértár-kapacitást igényel. Az adatvesztés utáni visszaállítás szintén sok időt vesz igénybe. Ennek ellenére adatainkat csak akkor tudhatjuk biztonságban, ha bizonyos időközönként elkészítjük a teljes mentést.

11.3.2 Inkrementális mentés

Az **inkrementális mentés** esetében csak az utolsó mentés óta **megváltozott adatokról** készítünk **másolatot**. Az ilyen mentéseket mindenképpen egy teljes mentésnek kell megelőznie. Azt követően azonban gyakran végezhetünk inkrementális mentéseket, hiszen általában kevés adat változik meg, így a másolat gyorsan elkészül, és kis tárigényű lesz. Visszaállításkor előbb a teljes mentésnek, majd az egyes inkrementális mentéseknek megfelelő állapotokat kell helyreállítanunk.

Az inkrementális mentésnek két altípusát, a **kumulatív**, és a **differenciális** mentést különböztetjük meg. A **kumulatív** típus esetén az egymást követő mentésekbe mindig **minden bekerül, ami az utolsó teljes mentés óta változott**. Így a legutolsó kumulatív mentés mérete és elkészítési ideje egyre nagyobb lesz, viszont visszaállításkor (a teljes mentésen kívül) csak erre lesz szükség.

A **differenciális mentéskor** a másolatba **csak a legutolsó** teljes, vagy inkrementális **mentés óta, megváltozott adatok** kerülnek be. Az egyes mentések mérete ezért viszonylag kicsi lesz, azok gyorsan elkészülnek, viszont visszaállításkor mindre szükség van. A differenciális mentés pozitívuma még, hogy alkalmazásával különböző időpontokig „visszagörgethető” az adatszerkezet állapota.

11.3.3 Online, offline mentés

Az adatbázisok mentésekor a mentésre kerülő adatok kinyerésére, a másolat elkészítésére felhasználhatjuk magát az adatbázis-szervert. Ilyenkor online, vagy hot (forró) mentésről beszélünk. Általában lehetőség van arra is, hogy a másolatokat a szerver megkerülésével készítsük el. Ezt nevezzük offline, vagy cold (hideg) mentésnek. Az online mentéskor a szervernek futnia kell, és a hozzáférés egy kliens útján történik. Az offline mentéskor a szervernek nem kell feltétlenül működnie, sőt, általában le is állítjuk.

11.3.4 Lokális mentés, távoli mentés

A **lokális** mentéseket a **szervert futtató hoston** végezzük el, és az eredmény, azaz a másolat is ezen a gépen keletkezik. Ez nagy sebességet kölcsönöz, de fizikai hozzáférést feltételez a szervert hardveréhez.

A **távoli**, vagy remote mentés alkalmával a **mentést távoli gépről indítjuk** el, és a mentés is oda kerül. Ez a módszer, a hálózati adatátvitel miatt lassúbb, de nem igényel fizikai hozzáférést. Mivel a másolat más gépre kerül, a távoli mentés eredménye a szerver fizikai károsodása esetén is megmarad.

Természetesen léteznek ennek a két típusnak a kombinációi is. Végezhetjük a mentést a lokális gépen, a másolatot pedig elhelyezhetjük a távoli hoston, de indíthatjuk backupot a távoli gépről, miközben a másolat a szervergépen marad.

11.3.5 Fizikai és logikai mentés

Adatbázis-szerverek adatainak biztonsági mentése lehet fizikai és lehet logikai mentés. **Fizikai mentéskor** az **adatbázisok adatait fizikailag tartalmazó könyvtárakat, állományokat másoljuk le**. Ez a mentés csak fájlszintű műveleteket igényel, gyors, egyszerű, és nemcsak az adatbázisok, hanem a konfigurációs állományok biztonsági másolására is alkalmas. Visszaállításkor egyszerűen a helyükre kell másolni az előzőleg mentett fájlokat.

A fizikai mentés hátránya, hogy a mentés idejére az adatbázis-kezelő rendszert le kell állítani, ami időszakosan lehetetlenné teszi az adatok elérhetőségét. További problémát jelent, hogy a mentett állományok különböző platformok közötti átvitele nehézségekbe ütközhet.

Logikai mentéskor a DMBS olyan **DDL- és DML-mondatokat** generál, amelyekkel az **adatbázis(ok) pillanatnyi állapota** és **adattartalma** lenne **előállítható**. A logikai mentés eredményeként kialakuló szövegállomány, az úgynevezett **dump**, DROP, CREATE, és INSERT parancsokat tartalmaz. **Helyreállításkor** a

dumpban lévő **mondatok kötegelt végrehajtására** utasítjuk a MySQL szervert. Ilyenkor, a művelet nyomán helyreáll a mentett adatszerkezet és tartalom. A logikai mentés óriási előnye, hogy a dump SQL-mondatokat tartalmazó, szerkeszthető szövegfájl, így egyes részei külön is fölhasználhatók.

Tananyagunkban terjedelmi okokból nem térünk ki a további mentési technikák – mint például az inkrementális mentés – MySQL-beli megvalósításaira, de érdemes tudnunk, hogy a MySQL-ben a logikai mentés az inkrementális mentés alapja. Az inkrementális mentés speciális, automatizált logikai mentések sorozata, amelyek eredménye naplófájlban tárolódik. A napló a visszaállítás egészen finoman hangolható végrehajtását teszi lehetővé.

11.4 ADATBÁZISOK LOGIKAI MENTÉSE

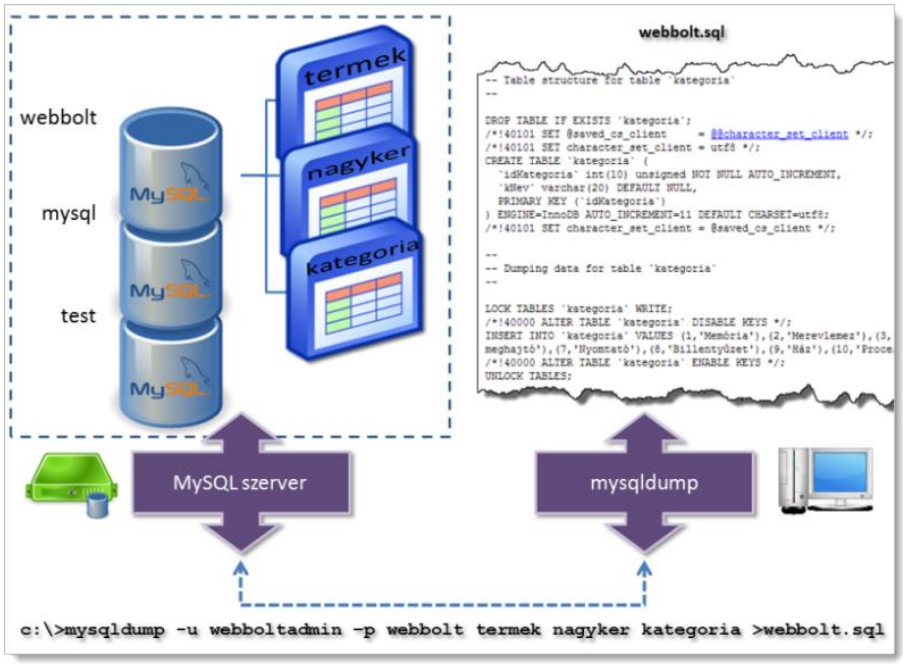
A logikai mentésről már tudjuk, hogy alkalmazásakor a lemezen tárolt „nyers” adatok másolása helyett az **adatszerkezet és a tartalom kialakítására alkalmas, SQL-mondatokat tartalmazó szövegfájlt, dumpot** hozunk létre.

11.5 A MYSQLDUMP ALKALMAZÁS

A logikai mentés lebonyolítása általában a MySQL egy speciális kliensprogramjával, a **mysqldump** segédprogrammal történik. Az eddig használt parancssori klienssel szemben a mysqldump **nem rendelkezik interaktív felülettel**. **Vezérlésére** a parancssorban megadott **paraméterekkel** lehetséges.

A mysqldump működését a MySQL konfigurációs állományában beállítható paraméterekkel is szabályozhatjuk, de mivel tananyagunkban nem foglalkoztunk a szerverkonfigurációval, leckénkben csak a parancssori paramétereket ismertetjük.

Az alábbi ábrán a **`webbolt`** adatbázis **`termek`**, **`nagyker`** és **`kategoria`** tábláiról készít logikai mentést a **`mysqldump`**. A mentéskor kapott webbolt.sql SQL-mondatainak végrehajtásával a adatbázis-kezelő rendszer újra létrehozná az a három táblát.



54. ábra A webbolt logikai mentése

A **mysqldump** használata az alábbi általános formában történik:

```
  mysqldump [opciók] adatbázis [tábla [tábla…]]
  mysqldump [opciók] --databases adatbázis
[adatbázis…]
  mysqldump [opciók] --all-databases
```

A különböző opciókkal kapcsolatfölépítés, a mentendő adatobjektumok, és a dump tulajdonságai szabályozhatók.

> A parancs kimente alaphelyzetben a képernyőn jelenik meg, így vajmi keveset kezdhetünk az eredménnyel. A dumpot ezért általában szövegállományba irányítjuk. Az előző példában a **webbolt.sql** nevű állományba küldtük a dumpot.

11.5.1 A mysqldump paraméterei

A program paramétereit a parancssori kliensnél megismert módon adhatjuk meg. A rövid, egybetűs forma előtt egy, a hosszú, szöveges paraméterek

előtt két kötőjel áll. (-u, --user). A parancs négy nagyobb csoportba osztható, jelentős számú paraméterrel rendelkezik. Ezek mind megjeleníthetők a **help** opcióval (**mysqldump --help**).

A paramétercsoportok az alábbiak:

- **kapcsolat**: A parancssori kliens azonos paramétereinek megfelelően a kapcsolat felépítéséhez szükségesek (**-u**, **-p**, **-h**…).
- **forrás**: Ezekkel az opciókkal adható meg, hogy mit akarunk menteni.
- **kimenet**: A dump részleteit szabályozó paraméterek.
- **egyéb**: Ide tartozik például a **--defaults-file** amivel a parancs konfigurációs állományának helyét adhatjuk meg.

11.5.2 Kapcsolat felépítése

A szerverrel való kapcsolat felépítéséhez az **–u** paraméter után adjuk meg a felhasználónevet, a **–p** paraméter pedig jelzi a jelszóhasználat szándékát jelzi. A **–h** opció után a szerver címe írható be, a **–P** (nagy P!) után pedig a portszámot kell elhelyezni.

A **–h** opciónak jó hasznát vehetjük távoli gépeken tárolt adatbázisok mentésekor. Segítségével távoli gépről a lokális gépre menthetünk adatbázisokat.

11.5.3 Forrás megadása

A forrás az az adatbázis, adatbázisok, vagy táblák, amelyekről a mentést készítjük. Az alábbi forma egyetlen, a megadott nevű adatbázist, de annak minden tábláját menti.

```
mysqldump [opciók] adatbázis
```

mysqldump –u webboltadmin –p webbolt

Ez a forma egy adatbázis megadott tábláiról készít dumpot.

```
mysqldump [opciók] adatbázis [táblák]
```

mysqldump -u webboltadmin -p webbolt termek kategoria

A következő formában több adatbázist is menthetünk, de nincs lehetőség táblák megadására. Minden felsorolt adatbázis összes táblája benne lesz a mentésben.

```
  mysqldump [opciók] --databases adatbázis
[adatbázis...]
```

A **mysqldump** alábbi változata a szerver összes adatbázisát dumpolja.

```
  mysqldump [opciók] --all-databases
```

11.5.4 Kimenet szabályozása

A dump tartalmazza a mentett adatszerkezetek ismételt létrehozásához szükséges CREATE-parancsokat, a mentéskor meglévő rekordok beszúrását biztosító INSERT-mondatokat. A dump esetleges visszaállításakor hiba keletkezhet, ha a szerveren már van olyan tábla, ami a dumpban is szerepel. Ezért már a mentéskor gondoskodhatunk arról, hogy bekerüljenek-e a dumpba olyan kiegészítő parancsok, amelyek kiküszöbölik az ilyen ütközéseket. Ha a táblákat létrehozó CREATE parancsok előtt szerepel az azonos nevű táblát törlő DROP is, akkor biztosan nem keletkezik hiba a visszaállításkor. A kimenetet szabályozó paraméterekkel a kiegészítő parancsok beillesztését szabályozhatjuk:

- **--add-drop-table**: Hatására a táblákat létrehozó **CREATE** parancsok elé, a meglévő változatot törlő **DROP** parancs kerül a dumpba.
- **--add-locks**: A táblák tartalmát beszúró **INSERT** parancsok **LOCK TABLES** és **UNLOCK TABLES** parancsok közé kerülnek. Így garantálható, hogy a visszaállításkor a felhasználók ne férjenek hozzá a táblákhoz, amíg minden rekordjuk a helyére nem kerül.
- **--create-options**: A dumpba bekerülnek a táblák szerkezetét fölépítő **CREATE** parancsok. Ha az opció kikapcsolt, csak **INSERT**-mondatok lesznek a mentésben. Ilyenkor a backup nem alkalmas a táblaszerkezet visszaállításra.
- **--lock-tables**: Ez az opció biztosítja, hogy az adatbázisok táblái olvasásra legyenek zárolva a mentés alatt. Feltétlenül használjuk, különben nem biztosítható az adatbázisok konzisztenciája!
- **--no-create-info**: a **--create-options** ellentéte. Hatására csak INSERT-ek kerülnek a dumpba.
- **--no-data**: Nem kerülnek a dumpba a rekordokat beszúró **INSERT** parancsok.

- **--tab**: Az opció után egy mappa útvonalát kell megadni. Hatására program létrehozza a dumpot, de a megadott mappában további állományokat helyez el. Különálló SQL-fájlokban ide kerül minden mentett tábla CREATE-je, továbbá a tábla adatait tartalmazó, tabulátorokkal szeparált CSV is. A --tab opció csak egy adatbázis mentésekor használható.
- **--compatible**: Más rendszereknek megfelelő SQL-nyelvjárásban történik a mentés. A paraméter értéke lehet ansi, mysql323, mysql40, postgresql, oracle, mssql, db2, maxdb.

11.6 LOGIKAI MENTÉS VISSZAÁLLÍTÁSA

Ha a dumpot megfelelően készítettük el, akkor alkalmas lesz a mentett adatszerkezetek és rekordjaik mentéskor aktuális állapotának helyreállítására. Ehhez semmi mást nem kell tenni, mint a mysql parancsori kliensben kötegelt módon végrehajtani a dump SQL mondatait.

A kötegelt végrehajtás a klines **\.** direktívájával lehetséges, ami után a dump útvonalát kell feltűntetni.

Például: **mysql> \. \backups\webbolt.sql**

A végrehajtás sikere természetesen függ a munkamenetünk pillanatnyi állapotától, és a dump tartalmától.

*Ha a parancssori kliens indítása után nem választottunk ki adatbázist, és a dump sem a **USE** paranccsal, hanem azonnal **CREATE TABLE**-lel kezdődik, akkor – kiválasztott adatbázis híján – kötegelt végrehajtás hibaüzenettel leáll.*

Ezért a dump végrehajtása előtt mindig meg kell vizsgálni annak tartalmát, a munkamenetünkben olyan környezetet előállítani, amelyben a dump hibamentesen futtatható.

11.7 ÖSSZEFOGLALÁS, KÉRDÉSEK

11.7.1 Összefoglalás

Utolsó leckénkben az adatok perzisztens és konzisztens tárolásának egy fontos kérdéséről a konzisztens állapotok mentéséről és visszaállításáról tanultunk. Megismertük az adatbázis-kezelésben – de az informatikában általában is

– elterjedt mentési technikákat, majd részletesen foglalkoztunk a MySQL-adatbázisok logikai mentésére használható **mysqldump** alkalmazással. Megismertük a **mysqldump** fontosabb attribútumait, amelyekkel a kliens és a szerver közötti kapcsolat, a mentésre szánt adatszerkezetek kijelölése, a dump tartalmának szabályozása, és néhány egyéb lehetőség állítható be. Leckénk végén a **mysqldumppal** készült mentések tartalmának visszaállítására láttunk példát.

11.7.2 Önellenőrző kérdések

1. Mi a különbség a teljes és az inkrementális mentés között?
   - A teljes mentéskor az adatbázis-kezelő rendszer felügyelete alá tartozó összes adatbázis minden táblájáról menjük. Az inkrementális mentéskor csak az utolsó teljes, vagy inkrementális mentés állapotához képest történt változtatásokat rögzítjük. Utóbbi jóval gyorsabb, és gyakoribb lehet, visszaállításkor pedig sokkal pontosabban tudjuk kiválasztani a visszaállítandó konzisztens állapotot.
2. Miért kell valamilyen állományba irányítani a mysqldump eredményét?
   - Azért, mert a mysqldump alapértelmezésként a képernyőre írja a dumpot.
3. Hogyan készítene dumpot a webbolt adatbázis vevo táblájáról? A mentésnek a \backups\ vevo.sql nevű állományba kell kerülnie!
   - mysqldump –u webboltadmin –p webbolt vevo>\backups\vevo.sql
4. Az a feladata, hogy a teljes webbolt adatbázist másolja át egy Oracle szerverre. Hogyan készítené el a dumpot?
   - mysqldump –u webboltadmin –p –compatible oracle webbolt>\backups\ora.sql
5. Szeretné visszaállítani a vevo.sql tartalmát. Írja le, milyen lépésekben hajtaná végre a feladatot! A dump így kezdődik:
   DROP TABLE vevo;
   CREATE TABLE vevo …
   - mysql –u webboltadmin –p
     mysql>USE webbolt;
     mysql>\. \backups\vevo.sql

12. ÖSSZEFOGLALÁS

12.1 TARTALMI ÖSSZEFOGLALÁS

A biztonsági mentésekkel foglalkozó leckével tananyagunk végéhez értünk. Összefoglalásként idézzük fel az egyes anyagrészeket, és az azokban megszerezhető legfontosabb ismereteket!

12.1.1 Bevezetés

Első leckénk a tananyag fölépítését, a leckék szerkezetét, mutatta be. Az olvasó megismerhette a szövegben található formátumokat, tanulási tanácsokat kapott, és áttekinthette a tananyag tematikáját.

12.1.2 Az adatbázisok elemei, az adatbázis-tervezés szintjei

A második lecke az adatbázisok szerkezetének és fölépítésének megértéséhez szükséges egyed, egyedtípus, tulajdonság, tulajdonságtípus, kapcsolat, és kapcsolattípus fogalmakat ismertette. Az alapfogalmak után Ön ebben a leckében olvashatott az adatbázisok modellezésének különböző absztrakciós szintjeiről, a koncepcionális, a logikai és a fizikai modellezésről.

12.1.3 Koncepcionális modellezés, ER-modell

Ebben a tananyagrészben az egyik széles körben elterjedt koncepcionális modellezési technikát, a Peter Chen által kifejlesztett ER-modellt, és ábrázolási módját, az ER-diagramot ismerhette meg. A leckében a koncepcionális modellezés olyan érdekes lehetőségeiről olvashatott, mint például a gyenge egyed, a többértékű mező, a kapcsolat számossága és a kötelezőség, vagy a specializáció.

12.1.4 A relációs adatmodell

A negyedik lecke a modellezés középső, logikai szintjét mutatta be. A hierarchikus, a hálós, és az objektumorientált modell alapfogalmai után részletesen elemeztük a relációs adatmodell strukturális elemeit és integritási szabályait. Megtanultuk a reláció, a mező, a rekord és mezőérték fogalmakat, és összekapcsoltuk őket a koncepcionális modellezés strukturális elemeivel.

12.1.5 A redundancia következményei és csökkentése

A redundancia a relációs adatbázisok hatékony működésének egyik legfőbb akadálya. Az 5. leckében azokkal a tervezési technikákkal foglakoztunk, amelyek az adatbázisok fölösleges adatismétlődéseinek megszűntetését, a változtatási, bővítési és törlési anomáliák kiküszöbölését tették lehetővé. A lecke a funkcionális függések elemzése után bemutatta a normalizálás technikáját, majd ismertette az ER-modell relációs modellre konvertálásának szabályait.

12.1.6 Adatbázis-kezelés és adatbázis-kezelő rendszerek

A modellezés fizikai szintjéhez közeledve már külön kellett választanunk az adatmodell és az adatbázis fogalmát. Meg kellett ismerkednünk az adatbázis-kezelő rendszerek legfontosabb jellemzőivel. A 6. lecke az adatbázis-rendszerek elemeivel, köztük a DBMS-ek funkcióival és a többszintű kliens-szerver architektúrával foglakozott.

12.1.7 Az SQL nyelv és nyelvjárásai, a MySQL

Ebben a leckében a relációs adatbázisok kezelésre és a relációs adatbázis-kezelő rendszerek irányítására használható szabványos nyelvvel, az SQL-lel ismerkedtünk meg. Az SQL története és résznyelvei után az SQL-mondatok formai leírásával foglalkoztunk, majd áttértünk az ingyenesen használható, de professzionális relációs adatbázis-kezelő rendszer, a MySQL megismerésére.

12.1.8 Az adatbázis és a táblák létrehozása

A relációs adatbázisok fizikai modellezésre a 8. fejezetben tértünk rá. Ez a lecke mutatta be az SQL DDL résznyelvének legfontosabb parancsát, a CREATE-et, amellyel adatbázisok, és táblák egyaránt létrehozhatók. A lecke a tábladefiniálás alapjait, a mezők leírását ismertette.

12.1.9 Indexelés, kapcsolatok, táblatulajdonságok

Az indexelés és a kapcsolatok fizikai modellezésének megismerése lehetővé tette, hogy ne csak a táblák szerkezetét, de a logikai adatmodellben megfogalmazott egyed- és hivatkozási integritási szabályokat is implementálni tudjuk. A 9. lecke az indexek, az elsődleges kulcsok, és az idegen kulcsok létrehozását mutatta be.

12.1.10 Adatbiztonság kérdései, a MySql jogosultsági rendszere

A fizikai modellezés után az adatbázis-kezelés fontos feladatának, a perzisztens és konzisztens tárolás több-felhasználós környezetben történő megvalósításának előfeltételét, a MySQL jogosultsági rendszerét ismertük meg. Tanultunk a kétlépéses ellenőrzésről és a többszintű jogrendszerről. megismertük a jogok biztosításának, precíz szabályozásának, és megvonásának parancsait.

12.1.11 További adminisztrátori feladatok

A további adminisztrátori feladatok leckében a mentés-visszaállítás kérdéseivel foglalkoztunk. Megismerkedtünk a legjellemzőbb mentési technikák jellemzőivel, majd a MySQL adatbázisok mentésére használt egyik leggyakoribb módszer, a **mysqldump** alkalmazással végezhető logikai mentéssel.

Megtanultuk, hogyan menthetjük szövegfájlba az adatbázis egy részének, vagy akár több adatbázis egészének létrehozására alkalmas DDL- és DML-parancsokat, majd megvizsgáltuk a dumpok visszaállításának lehetőségeit.

12.1.12 Összefoglalás

Jelen utolsó leckénkben áttekintettük a tananyag fontosabb fogalmait, összegeztük a megszerzett ismereteket.

12.2 ZÁRÁS

Tananyagunk végén megköszönjük az olvasó érdeklődését, figyelmét és kitartását. Bízunk benne, hogy a leckékben szerzett tudás képessé teszi Önt a hatékonyan, biztonságosan működő relációs adatbázisok modellezésére, és kivitelezésére. Tananyagunkban nem foglakoztunk az adatbázis-kezelő rendszerek tényleges adatkezelő műveleteinek az SQL DML, és DQL résznyelveinek bemutatásával, azonban biztosak vagyunk abban, hogy a megszerzett ismeretek birtokában az olvasó már könnyedén veszi majd ezt a következő akadályt. A relációs adatbázisok adatainak tárolásával, módosításával, törlésével, a relációkban tárolt adatok kinyerésével tankönyvünk következő részében ismerkedhet meg.

12.3 FOGALMAK

12.3.1 Bevezetés

12.3.2 Az adatbázisok elemei, az adatbázis tervezés szintjei

Egyed, tulajdonság, egyedtípus, tulajdonságtípus, kapcsolat, kapcsolattípus, adatbázis-modellezés, koncepcionális szint, logikai szint, fizikai szint.

12.3.3 Koncepcionális modellezés, ER-modell

ER, Entity-Relationship, erős egyed, gyenge egyed, parciális kulcs, összetett attribútum, többértékű attribútum, származtatott attribútum, fokszám, számosság, totális- és parciális részvétel, számosság jelölések, rekurzív kapcsolat, EER, alosztály, főosztály.

12.3.4 A relációs adatmodell

Hierarchikus adatmodell, hálós adatmodell, objektumorientált adatmodell, relációs adatmodell, reláció, attribútum, domain, rekord, kulcs, kulcsjelölt, alternáló kulcs, összetett kulcs, idegen kulcs, származtatott reláció, snapshot, view, egyed integritás, hivatkozási integritás.

12.3.5 A redundancia következményei és csökkentése

Redundancia, anomáliák, függőség, teljes függőség, részleges függőség, tranzitív függőség, normalizálás, normálforma, dekompozíció, UNF,1NF, 2NF, 3NF, ER leképezési szabályok.

12.3.6 Adatbázis-kezelés, és adatbázis-kezelő rendszerek

Adatmodell, adatbázis, ANSI-SPARC architektúra, adatbázis-rendszer, adatfüggetlenség, kliens-szerver architektúra.

12.3.7 Az SQL nyelv és nyelvjárásai, a MySQL

Fizikai modell, SQL, nyelvjárások, DDL, DML, DCL, DQL, MySQL szerver, mysql kliens.

12.3.8 Az adatbázis és a táblák létrehozása

SHOW, USE, DDL, CREATE, DROP, adattípusok, ALTER.

12.3.9 Indexelés, kapcsolatok, táblatulajdonságok

Táblamotor, SHOW ENGINES, indexelés, egyedi index, elsődleges kulcs, AUTO_INCREMENET, idegen kulcs, hivatkozási integritás, típus kompatibilitás, nézetek, CASE-eszközök.

12.3.10 Adatbiztonság kérdései, a MySql jogosultsági rendszere

Kétlépéses ellenőrzés, azonosítás, jogosultságok, többszintű jogosultsági rendszer, GRANT, SHOW GRANTS, REVOKE, DROP USER.

12.3.11 További adminisztrátori feladatok

Mentés, teljes mentés, inkrementális mentés, lokális mentés, távoli mentés, fizikai mentés logikai mentés, mysqdump.

13. KIEGÉSZÍTÉSEK

13.1 IRODALOMJEGYZÉK

13.1.1 Hivatkozások

Könyv

HALASSY, Béla.: *Az adatbázis-tervezés alapjai és titkai*. Budapest, IDG Magyarországi Lapkiadó Kft.,1994

TÍMÁR, Lajos., VÍGH, Katalin., SZIGETI, Judit.: *Így még könnyebb az adatmodellezés!* Veszprém, Veszprémi Egyetemi Kiadó, 1997.

TÍMÁR, Lajos.: *Építsünk könnyen és lassan adatmodellt*: Veszprém, Veszprémi Egyetem, 1999

ZAIDAH, Ibrahim: *Database system*. Open University Malaysia, 2010.

Elektronikus dokumentumok / források

DOMINICH, Sándor: alcím. Kiadás helye, Kiadó, Kiadás éve. [online dokumentum] [2012.06.12] <URL: http://www.dcs.vein.hu/CIR/cikkek/Database_Theory.pdf>

KOVÁCS László: *Adatbázis rendszerek I.* Miskolc, Miskolci Egyetem. [online dokumentum] [2012.június 1.] <URL: http://www..sztaki.hu>

Adatbázis rendszerek [online dokumentum] [2012.06.03] <URL: http://www.kobakbt.hu/jegyzet/AdatbazisElmelet/index.html>