A D A T B Á Z I S O K

Előadási jegyzet (BSc)

Készítette: dr. Katona Endre

Szegedi Tudományegyetem
Informatikai Tanszékcsoport
2013.

Ez a jegyzet az adatbázis-tankönyvek szokásos felépítését követi:

– Az 1.-5. fejezetek az adatbázis-tervezés kérdéskörét tárgyalják (egyed-kapcsolat modell, relációs modell, normalizálás).
– A 6.-8. fejezetek az SQL nyelvet és annak alkalmazásait tekintik át.
– A 9.-10. fejezetek konkrét adatbázis-kezelő rendszerekről szólnak.
– Az utolsó, adatbiztonságról szóló fejezet már a nagy adatbázis-alkalmazások világába nyújt bepillantást.

A mintapéldák tábla-, mező- és változónevei – a könnyebb olvashatóság érdekében – ékezetes betűkkel szerepelnek, konkrét programozási környezetben azonban ez esetleg nem megengedett vagy zavarokat okozhat, tehát kerülendő.

Az apró betűs szövegrészek kevésbé fontos részleteket tartalmaznak, amelyek a tananyag mélyebb megértéséhez ajánlottak.

A jegyzetben talált esetleges hibákat kérem jelezzék a
katona@inf.u–szeged.hu címre.

Tartalom

1. Bevezetés

Az első számítógépeket matematikai feladatok megoldására készítették, de már az 1960-as évek elejétől a számítógépes alkalmazások nagyobbik részét az adatfeldolgozás tette ki. Kezdetben egyedi programok készültek az egyes vállalatoknál a munkaügyi, termelési, stb. adatok nyilvántartására. A tömeges alkalmazási igény azonban kikényszerítette az adatformátumok szabványosítását, és általános célú adatbázis-kezelő szoftverek kifejlesztését.

> Adatok *gépi kezelés*ére többféle eszköz is alkalmas lehet:
> *1. Szövegszerkesztő program.* Tegyük fel például, hogy egy vállalat dolgozóinak önéletrajzát tároljuk egy szövegfájlon. Ebben a fájlban rá lehet keresni adott névre, lakcímre, lehet csoportosítani vállalati osztályok szerint (vázlatszint). Ugyanakkor probléma lekérni például azon dolgozók listáját, akik 1960 és 1970 között születtek.
> *2. Hypertext (web).* A hivatkozások (linkek) segítségével fájlon belül és fájlok között is komplex kapcsolatok alakíthatók ki (lásd még a HTML és XML egyéb lehetőségeit).
> *3. Táblázatkezelő program.* Itt a fontosabb életrajzi adatok (név, lakcím, születési dátum, iskolai végzettség) már elkülönítve tárolhatók, és számos lekérdezési lehetőség van. Viszont sokféle adat közötti bonyolult kapcsolatrendszert, nagy adathalmazok hatékony és biztonságos kezelését nem támogatják a táblázatkezelők.
> *4. Adatbázis-kezelő rendszer.* A nyilvántartás valamilyen *adatmodell*re épül, amely komplex kapcsolatrendszer kézbentartását is lehetővé teszi. Az adatbázis-kezelő rendszerek kimondottan nagy adatmennyiség hatékony és biztonságos kezelését támogatják.

Adatok típusai:
a) Egyszerű (atomi) adat: szám, string, dátum, logikai érték.
b) Összetett adat: egyszerű adatokból képezhető. Változatai:
– halmaz: egynemű elemek halmaza. Példa: egy vállalat osztályai.
– lista: egynemű elemek rendezett sorozata. Példa: könyv szerzői.
– struktúra: különféle elemek rendezett sorozata. Példa: lakcím = (helység, utca, házszám).
– a fentiek kombinációi.
c) NULL: definiálatlan adat. (Nem azonos a nulla értékkel!)

Elnevezések:
Adatbázis (= DB = database): adott formátum és rendszer szerint tárolt adatok együttese.
Adatbázis-kezelő rendszer (= DBMS = Database Management System): az adatbázist kezelő szoftver.
Rekord (= feljegyzés): az adatbázis alapvető adategysége. Általában struktúra felépítésű.

A DBMS fő feladatai:
- adatstruktúra (adatbázisséma) definiálása,
- adatok aktualizálása (új felvétel, törlés, módosítás),
- lekérdezési lehetőségek,
- fejlesztő környezet biztosítása célalkalmazások létrehozásához.

Néhány ismertebb DBMS:

- *xBase rendszerek* (dBase, FoxPro, Clipper): elavult, de még sok alkalmazás működik.
- *Access* (Microsoft): könnyen kezelhető grafikus felület, kisebb alkalmazásokhoz.
- *MySQL:* nyílt forráskódú adatbázis-szerver, közepes méretű (pl. webes) alkalmazásokhoz.
- *Oracle:* nagy teljesítményű rendszer, nagy adatbázisok, sok felhasználó, különleges biztonsági követelmények esetén ajánlott.

Egy *adatbázis-alkalmazás*nál az alábbi szinteket különböztethetjük meg:

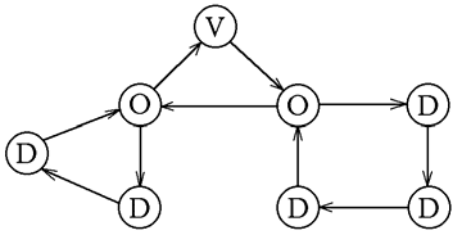
Felhasználói felület
Célalkalmazásként készített program
Adatmodell (logikai adatstuktúra)
DBMS
Fizikai adatstruktúra

1.1. Adatmodellek áttekintése

Adatbázisséma: az adatbázis struktúrájának leírása. Erre különféle adatmodellek használatosak.

Hierarchikus modell: a rekordok fastruktúra-szerű hierarchiába rendezettek (például vállalat, főosztályok, osztályok, dolgozók). A feldolgozás fabejáró és egyéb fastruktúra kezelő algoritmusok segítségével történik. A hierarchikus modellnek ma már csak történeti jelentősége van.

Hálós modell (1961): a rekordok pointerekkel kapcsolódnak egymáshoz. A pointerek ciklikusan körbefutnak egy összetartozó rekordcsoporton, egy ilyen csoportot *set*nek neveznek. Egy set mindig egy "szülő" és több "gyermek" rekordot tartalmaz (például set lehet egy vállalati osztály és a dolgozói, lásd 1. ábra.) A hálós modell ma már szintén csak történeti jelentőséggel bír.



1. ábra. Vállalati osztályok és dolgozók nyilvántartása hálós modellben (V: vállalat, O: osztály, D: dolgozó)

Relációs modell (1970): az adatok kétdimenziós táblákban tárolódnak, a rekordok közötti kapcsolatot pointerek helyett szintén táblázatok valósítják meg. A relációs modellre épülő adatbáziskezelőket RDBMS-nek (Relational DBMS) nevezzük. Szabványos leíró/lekérdező nyelvük az SQL. A relációs modell jelenleg a legszélesebb körben használatos.

Objektumorientált modell (1990-es évek). Az objektumorientált programozási nyelvek (C++, Smalltalk) eszközrendszerével definiálja az adatbázis struktúráját. Leíró nyelve az ODL, lekérdező nyelve az OQL. Az objektumorientált modellre épülő adatbázis-kezelő rendszereket OODBMS-nek nevezzük (Object Oriented DBMS). Ezek fejlesztő nyelve általában C++ vagy Smalltalk. Az OODBMS rendszerek a gyakorlatban nem terjedtek el.

Objektum-relációs modell: a relációs modell bővítése objektumorientált lehetőségekkel, az erre épülő rendszereket ORDBMS-nek nevezzük (Object Relational DBMS). Ezek széles körben használatosak.

2. Egyed-kapcsolat modell

Grafikus leíró eszköz, diagram segítségével szemléletesen adja meg az adatbázis struktúráját. Az adatbázis implementálásához a diagramot transzformálni kell valamilyen adatmodellre, ill. annak megfelelő nyelvi leírásra (pl. SQL).

1. Példa. Tegyük fel, hogy egy *könyvtár kölcsönzési nyilvántartását* szeretnénk adatbázissal megoldani. Ehhez nyilvántartást kell vezetni
- a könyvekről,
- az olvasókról,
- a kikölcsönzési és visszahozási időpontokról.

A modell megalkotásához néhány alapfogalmat meg kell ismernünk.

*Egyed*nek vagy *entitás*nak nevezünk egy, a valós világban létező dolgot, amit tulajdonságokkal akarunk leírni. Esetünkben egyed lehet egy könyv a könyvtárban, illetve egy adott olvasó. Általánosított fogalmakat használva beszélhetünk "könyv" egyedről és "olvasó" egyedről is.

*Tulajdonság*nak vagy *attribútum*nak nevezzük az egyed egy jellemzőjét. Például a könyv, mint egyed legfontosabb tulajdonságai a cime, és a szerző neve.

Az attribútumokat úgy célszerű megválasztani, hogy azok egyértelműen meghatározzák az egyedet. Mivel adott szerző adott című könyve több kiadásban is megjelenhet, sőt adott kiadásból is több példány lehet a könyvtárban, így minden könyvhöz egy egyedi azonosítót, *könyvszámot* (könyvtári számot) célszerű felvenni. Ekkor a "könyv" egyed tulajdonságai: *könyvszám, szerző, cím*. (További tulajdonságoktól, mint kiadó, kiadási év, stb. esetünkben eltekintünk.) Hasonló meggondolások alapján az "olvasó" egyedhez *olvasószám, név, lakcím* tulajdonságokat rendelhetünk.

Egy egyed attribútumainak azt a minimális részhalmazát, amely egyértelműen meghatározza az egyedet, *kulcs*nak nevezzük és *aláhúzás*sal jelöljük. Esetünkben a „könyv” egyed kulcsa a *könyvszám*, az „olvasó” egyedé az *olvasószám*.

Könyvtári nyilvántartásunk azonban ezzel még nincs kész. A "könyv" és "olvasó" egyedek között ugyanis egy sajátos *kapcsolat* léphet fel, amelyet *kölcsönzés*nek nevezünk. Ezen kapcsolathoz a kivétel és visszahozás időpontját rendelhetjük tulajdonságként.

A valós világ jelenségeit egyedekkel, tulajdonságokkal és kapcsolatokkal leíró modellt *egyed-kapcsolat modell*nek, az ezt ábrázoló diagramot *egyed-kapcsolat diagram*nak nevezik. (Rövidítve az *E-K modell* és *E-K diagram*, illetve az angol *entity-relationship model* elnevezés alapján az *E-R modell* és az *E-R diagram* elnevezések használatosak.) Megjegyezzük, hogy hasonló modellezési technikát használ az SSADM rendszerszervezési módszertan is.

Az egyed-kapcsolat diagramoknak sajátos jelölésrendszerük van:
- az egyedeket téglalappal,
- az attribútumokat ellipszissel,
- a kapcsolatokat rombusszal

szokták jelölni. A 2. ábra a fentiekben tárgyalt könyvtári nyilvántartás E-K diagramját ábrázolja. A tervezés kezdeti szakaszában, illetve bonyolult E-K diagramok esetén az attribútumok ábrázolását el szokták hagyni.

Az eddig leírtaknál kissé pontatlanul fogalmaztunk, ugyanis meg kell különböztetni *egyedpéldány*t, *egyedtípus*t és *egyedhalmaz*t. Példánkban az egyedpéldány egy adott könyvet, az egyedtípus a *könyv* fogalmat jelenti. Egy valós adatbázisban minden egyedtípusnak egy konkrét egyedhalmaz (egyedpéldányok halmaza) felel meg. A kissé nehézkes terminológia elkerülésére az egyedpéldány, egyedtípus és egyedhalmaz helyett egyszerűen *egyed*et mondunk, ha ez nem értelemzavaró.

Hasonlóan beszélhetünk *tulajdonságpéldány*ról, amely egy egyedpéldány adott tulajdonságát jelenti (például adott könyv szerzőjének nevét), és *tulajdonságtípus*ról, amely adott egyedtípus adott tulajdonságát, mint fogalmat jelöli (például könyvek esetén a "szerző" fogalmat).

Ugyanígy meg lehet különböztetni *kapcsolatpéldány*t, amely két egyedpéldány közötti konkrét kapcsolatot jelent (például X olvasó kikölcsönözte Y könyvet), *kapcsolattípus*t és *kapcsolathalmaz*t, ez utóbbi a két egyedtípus közötti kapcsolatok összességét jelenti.

2. ábra: Könyvtári nyilvántartás E-K diagramja.

Fontos az egyedtípus pontos (informális) meghatározása. Például, egy egyetemi oktatási adatbázisnál a *kurzus* egyedtípus többféleképp értelmezhető:

(i) Több féléven keresztül tartó kurzust egy egyednek tekintünk.

(ii) Az összetartozó előadást és gyakorlatot egy kurzusnak tekintjük.

(iii) Adott helyen és időpontban tartott foglalkozást tekintünk kurzusnak. Ha több hallgatói csoport van, akkor mindegyik csoport gyakorlati órája külön egyedpéldányt jelent.

2.1. Kapcsolatok típusai

A kapcsolatok típusai a következők:

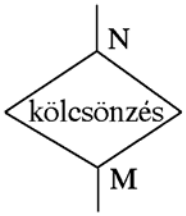
a). *Két egyed közötti (*más néven *bináris) kapcsolat*, mint a könyvtári példa esetében. Ennek három altípusa lehetséges (E és F jelöli a két egyedtípust):

- *1:1 kapcsolat*, amikor minden E-egyedhez csak legfeljebb egy F-egyed tartozhat, és fordítva.

- *1:N kapcsolat (egy-a-sokhoz kapcsolat)*, amikor egy E-egyedhez több F-egyed tartozhat, de ez fordítva nem igaz, vagyis egy F-egyedhez csak legfeljebb egy E-egyed tartozhat.

- *N:M kapcsolat (sok-a-sokhoz kapcsolat)*, amikor mindkét fajta egyedhez tetszőleges számú másik fajta egyed tartozhat.

b). *Kettőnél több egyed közötti (*más néven *sokágú) kapcsolat*. Ez a típus ritkábban lép fel, szükség esetén visszavezethető bináris kapcsolatokra.



3. ábra: Kapcsolat típusának jelölése felirattal

4. ábra. Kapcsolat típusának jelölése nyíllal az "1"-oldalon (rendre 1:1, N:1, N:M kapcsolat)

A könyvtári nyilvántartás mindhárom típusra példával szolgálhat.

1. változat: Tételezzük fel, hogy a könyvtáros két feltételezéssel él:

a). Egy olvasónak egyszerre csak egy könyvet hajlandó kiadni.

b). Csak azt kívánja nyilvántartani, hogy egy adott könyv éppen kinél van, azt nem, hogy korábban ki(k)nél volt. (Ekkor valójában fölöslegessé válik a "visszahozás" tulajdonság, hisz a könyv visszahozásakor a könyv-olvasó kapcsolat megszűnik.)

A fenti feltételezések mellett a könyv és olvasó egyedek között *1:1 kapcsolat* lép fel, hiszen egy könyv egyszerre csak egy olvasónál lehet, illetve egy olvasó egyszerre csak egy könyvet vihet ki.

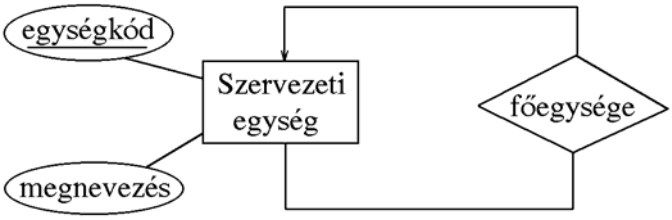
2. változat: Most tételezzük fel, hogy a könyvtáros eltekint az *a)*. feltételtől, és egy olvasónak egyszerre több könyvet is hajlandó kiadni. Ekkor a könyv és olvasó egyedek között *N:1 kapcsolat* lép fel, ugyanis egy olvasónál egyszerre több könyv lehet, viszont egy könyv egyszerre csak egy olvasónál tartózkodhat.

3. változat: Tegyük fel, hogy a könyvtáros eltekint a *b).* feltételtől is, és azt is nyilván akarja tartani, hogy egy adott könyv korábban mely olvasóknál mettől meddig volt kint. Ekkor már egy könyv több könyv-olvasó kapcsolatban is részt vehet, ezért a két egyed között *N:M kapcsolat* áll elő.

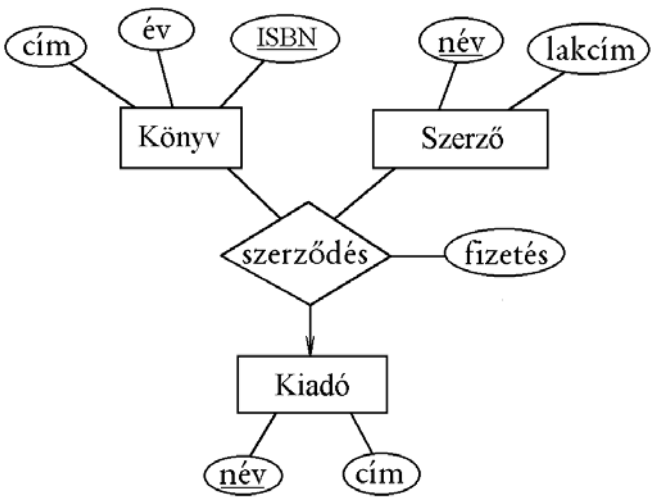
Látjuk, hogy a kapcsolat típusa lényeges az E-K modell szempontjából, ezért azt az E-K diagramon a 3. ábra vagy 4. ábra szerint jelölni szokták.

Egy egyedtípus *teljesen részt vesz* egy kapcsolatban, ha minden egyedpéldány kapcsolatban áll valamely másik egyeddel. Ha ezt hangsúlyozni akarjuk, akkor az egyed és a kapcsolat közötti *kettős vonalat* húzunk. A teljes részvétel általában nem teljesül, például a könyvtári nyilvántartás 1. és 2. változatánál rendszerint nincs minden könyv kikölcsönözve, és nincs minden olvasónál könyv. A 3. változatnál viszont megkövetelhetjük, hogy egy olvasót csak akkor veszünk nyilvántartásba, ha valamikor legalább egy könyvet kölcsönzött, ekkor az Olvasó egyed teljesen részt vesz a kapcsolatban.

2. Példa. Előfordul, hogy egy egyedtípus önmagával áll kapcsolatban. A 5. ábra egy hierarchikus felépítésű intézmény szervezeti egységeit modellezi (például egyetemi karok, tanszékcsoportok, tanszékek). Itt 1:N kapcsolatról van szó, ahol egy kapcsolatpéldány azt jelenti, hogy X egységnek Y egység a főegysége. Megjegyzendő, hogy ez a modell nem zárja ki a körkörös hivatkozásokat.



5. ábra. Hierarchikus felépítésű intézmény szervezeti egységeinek modellezése

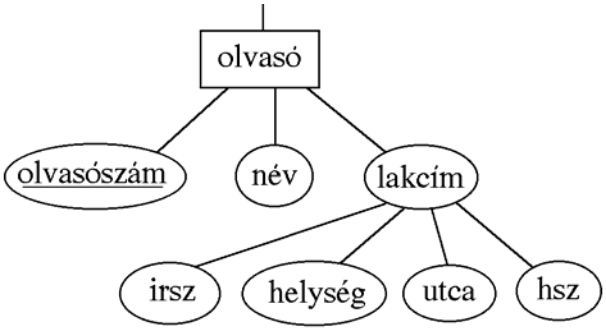


6. ábra. Példa sokágú kapcsolatra

3. Példa. A 6. ábra sokágú kapcsolatra ad példát. A kiadóra mutató nyíl azt jelenti, hogy adott (könyv, szerző) pár legfeljebb egy kiadóval állhat kapcsolatban. Hasonló állítás nem igaz a (kiadó, szerző) és (könyv, kiadó) párokra, mivel egy könyvnek több szerzője lehet.

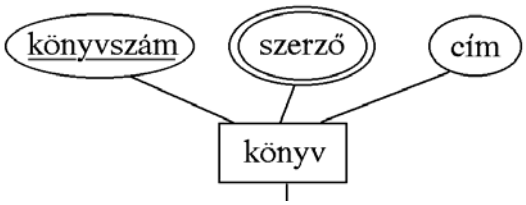
2.2. Összetett és többértékű attribútumok

Összetett attribútum (struktúra): maga is attribútumokkal rendelkezik. Például a *lakcím* attribútumhoz az *irányítószám, helység*, *utca*, *házszám* részattribútumok tartoznak. Jelölése: *attribútumhoz kapcsolódó attribútumok*.



7. ábra. Példa összetett attribútumra

Többértékű attribútum: aktuális értéke halmaz vagy lista lehet. Ha például egy könyvnek több szerzője van, és azok sorrendjét nem tartjuk fontosnak, akkor halmazként, ha fontosnak tartjuk, akkor listaként adhatjuk meg a neveket. A többértékű attribútum jele *kettős ellipszis*.



8. ábra. Példa többértékű attribútumra

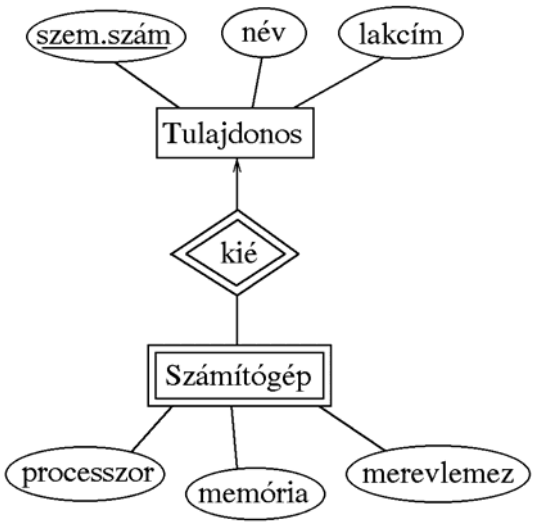
2.3. Gyenge entitások

Gyenge entitás: az attribútumai nem határozzák meg egyértelműen, csak a kapcsolatai révén lesz meghatározott. *Jele:* kettős téglalap.

Meghatározó kapcsolat: gyenge entitást határoz meg. *Jele:* kettős rombusz.

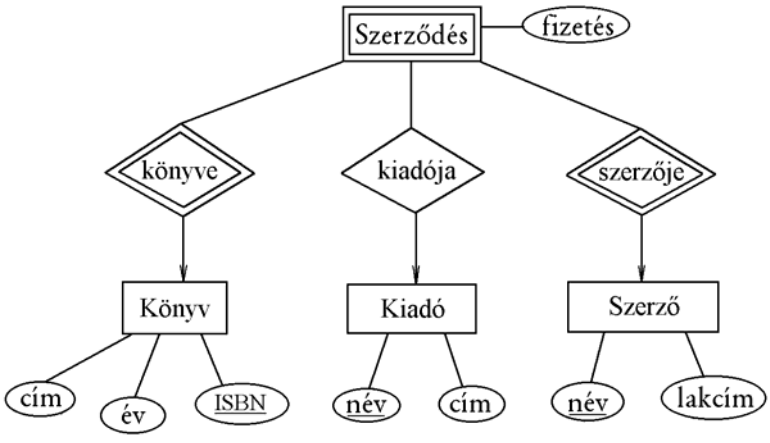
4. Példa. Egy számítógép szerviz nem bajlódik azzal, hogy egyedi azonosítót rendeljen a javított gépekhez, hanem azokat a tulajdonosaik szerint tartja nyilván (9. ábra). Itt a *számítógép* gyenge entitás, mivel a műszaki paraméterek nem határozzák meg egyértelműen a

gépet. Ha előfordulhat, hogy egy tulajdonosnak több, azonos paraméterekkel rendelkező gépe van, akkor a *számítógép* egyedhez egy *sorszám* attribútum felvétele is szükséges a megkülönböztetésre. Ez azonban könnyebben kezelhető, hisz itt csak adott tulajdonos gépeit kell egymástól megkülönböztetni, nem az összes gépet.



9. ábra. Példa gyenge entitásra: számítógép szerviz nyilvántartása

N:M típusú és sokágú kapcsolat mindig helyettesíthető gyenge entitással és több bináris kapcsolattal (10. ábra).



10. ábra. Sokágú kapcsolat (6. ábra) helyettesítése gyenge egyeddel és bináris kapcsolatokkal

2.4. Specializáló kapcsolatok

Ha valamely általános egyednek bizonyos altípusait külön szeretnénk modellezni, akkor a főtípus és az altípusok viszonyát specializáló kapcsolattal írhatjuk le.

Jelölés: háromszög, amelynek csúcsa a főtípus felé mutat. A háromszögbe angolul "is a", magyarul "az egy" szöveget szoktak írni, ezzel is hangsúlyozva a kapcsolat jellegét.

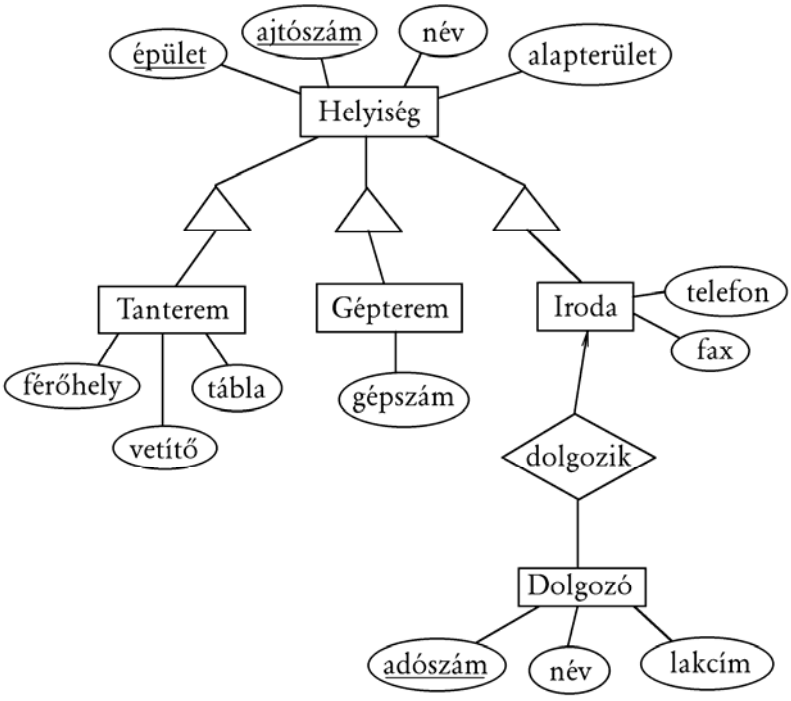
5. Példa. A 11. ábrán egy oktatási intézmény helyiségeit nyilvántartó diagram látható. Az egyes helyiségeket a tartalmazó épület azonosítójával és az azon belüli ajtószámmal azonosítjuk, további attribútumok a helyiség neve és alapterülete.

A *helyiség* egyed altípusai a *tanterem* (attribútumok: az ülőhelyek száma, a tábla és vetítő típusa), a *számítógépterem* (attribútum: a gépek száma) és az *iroda* (attribútumai az irodában működő telefon és fax száma, és kapcsolatban áll az irodában dolgozó személyekkel).

Látjuk, hogy az altípusoknak lehetnek saját attribútumai és kapcsolatai, ugyanakkor *öröklik* a főtípus attribútumait és esetleges kapcsolatait is. Például a *tanterem* teljes attribútumhalmaza: épület, ajtószám, név, alapterület, férőhely, vetítő, tábla.

A specializáló kapcsolat az egyedek *többszörös előfordulás*át eredményezi. Ha ugyanis egyedhalmazokat képzelünk a főtípus és altípusok helyére, akkor egy egyedpéldány több egyedhalmazban is szerepel: például egy konkrét előadóterem egyaránt része a *Helyiség* és *Tanterem* egyedhalmazoknak. A specializáló kapcsolat lényegében 1:1 kapcsolatot jelent egy főtípus és egy altípus között, de sajátos módon nem különböző egyedeket, hanem ugyanazon egyed két előfordulását kapcsolja össze. Az altípus mindig teljesen részt vesz ebben a kapcsolatban, míg a főtípus általában nem.

Egy egyed egyszerre kettőnél több egyedhalmazban is előfordulhat, egy számítógépes oktatóterem például tanterem és gépterem egyszerre. Végül az is lehet, hogy egy egyed csak a főtípushoz tartozik (például folyosó, mosdó, raktár, stb.)



11. ábra. Oktatási intézmény helyiség nyilvántartása

3. A relációs adatmodell

3.1. A relációs adatmodell fogalma

A relációs adatmodellt 1970-ben definiálta E. F. Codd amerikai kutató, de gyakorlati alkalmazása csak az 1980-as években vált általánossá. Lényege, hogy az egyedeket, tulajdonságokat és kapcsolatokat egyaránt táblázatok, úgynevezett *adattáblák* segítségével adja meg.

Az adattábla (vagy egyszerűen csak *tábla*) sorokból és oszlopokból áll. Egy sorát *rekord*nak nevezzük, amely annyi *mező*ből áll, ahány oszlopa van a táblának.

6. Definíció. *Attribútum*nak nevezünk egy tulajdonságot, amelyet a megnevezésével azonosítunk, és *értéktartományt* rendelünk hozzá. A *Z* attribútum értéktartományát a domain szó rövidítésével jelöljük: dom(*Z*).

Korlátozás: a relációs adatmodellnél az értéktartomány *csak atomi értékekből állhat,* vagyis elemei nem lehetnek struktúrák, halmazok, stb.

Az értéktartomány megadása rendszerint *típus* és *hossz* megadását jelenti, például a *könyvszám* attribútum értéktartománya a legfeljebb 4-jegyű decimális számok halmaza lehet. A gyakorlatban az attribútumnévhez általában informális leírást (kódolási utasítást) kell mellékelni, amely az attribútum megadását pontosítja (például a *szerző* attribútumot több szerző esetén hogyan kell megadni, a *könyvszám* egyes számjegyei utalhatnak a könyv jellegére, stb.).

7. Definíció. *Relációséma*nak nevezünk egy attribútumhalmazt, amelyhez azonosító nevet rendelünk. (Ahol nem értelemzavaró, relációséma helyett egyszerűen csak *sémát* mondunk.)

Jelölések:

- A relációsémát $R(A_1,...,A_n)$ módon szokás jelölni, ahol $A_1,...,A_n$ attribútumok, R pedig a séma neve.
- Használjuk még az R(A) jelölést is, ahol A az $\{A_1,...,A_n\}$ attribútumhalmaz.
- Az R séma A_i attribútumát $R.A_i$-vel jelöljük, ha különböző sémák azonos nevű attribútumait kell megkülönböztetni.

Megállapodás. A továbbiakban mindvégig, ha valamely Z attribútum(rész)halmazról beszélünk, akkor feltételezzük, hogy Z nem üres. Ha üres halmaz is megengedett, erre külön felhívjuk a figyelmet.

8. Példa. A könyvek nyilvántartására szolgáló relációséma
Könyv (könyvszám, szerző, cím), ahol az egyes attribútumok értéktartománya:
dom(könyvszám) = 4-jegyű decimális számok halmaza,
dom(szerző) = legfeljebb 30 hosszú karaktersorozatok halmaza,
dom(cím) = legfeljebb 50 hosszú karaktersorozatok halmaza.

9. Definíció. *Reláció* az $R(A_1,...,A_n)$ séma felett: $T \subseteq dom(A_1) \times ... \times dom(A_n)$. Vagyis, T elemei $(a_1,...,a_n)$ alakúak, ahol $a_i \in dom(A_i)$ $(i=1,...,n)$.

A reláció megjelenési formája az *adattábla*, amelynek oszlopai az $A_1,...,A_n$ attribútumoknak, sorai pedig T egyes elemeinek felelnek meg.

10. Példa. Tekintsük a Könyv (könyvszám, szerző, cím) sémát! Ekkor a dom(könyvszám) X dom(szerző) X dom(cím) halmaz az összes lehetséges *(könyvszám, szerző, cím)* hármast tartalmazza. Ezek közül kiválasztjuk azokat, amelyek a könyvtárban lévő könyveknek felelnek meg, ez lesz a *T* halmaz. Például *T* a következő lehet:

(1121, Sályi, Adatbázisok)
(3655, Radó, Világatlasz)
(2276, Karinthy, Így írtok ti)
(1782, Jókai, Aranyember)

Az adattábla fejlécében a relációsémát szokták megadni, amely azonban matematikai értelemben nem része a táblának. Példánk esetében:

Könyvszám	*Szerző*	*Cím*
1121	Sályi	Adatbázisok
3655	Radó	Világatlasz
2276	Karinthy	Így írtok ti
1782	Jókai	Aranyember

A matematikában halmazok Descartes-szorzatának részhalmazát általában relációnak nevezik. Mivel az adattáblát relációként definiáltuk, innen kapta nevét a relációs adatmodell.

Ahogy az E-K modellnél megkülönböztettünk egyedtípust és egyedpéldányt, a relációs modellnél is beszélhetünk *relációtípus*ról, amely a relációsémának felel meg, és *relációpéldány*ról, amely az adattáblának felel meg.

Általános esetben a sémára és táblára külön jelölést használunk (például R séma feletti T tábla), de konkrét példák esetén a kettőt azonosan jelöljük (például Könyv séma és Könyv tábla).

Mivel a definíció szerint a T reláció egy halmaz, így a relációs modellben *a tábla minden sora különböző*, és a sorokra *semmilyen rendezettséget nem tételez fel*. Valójában az adatok gépi tárolása mindig valamilyen sorrendben történik, és a konkrét adatbázis-kezelő rendszerek általában megengednek azonos sorokat is. Az elméleti modell és a gyakorlati alkalmazás ezen eltéréseire mindig ügyelni kell.

A relációs modell valójában a tábla oszlopaira sem határoz meg sorrendet. Mivel a reláció fenti definíciója akaratlanul is kiköti az oszlopok sorrendjét, így egy másik definíció is használatos:

Tekintsük a $D = dom(A_1) \cup ... \cup dom(A_n)$ egyesített értéktartományt és az $A = \{A_1,...,A_n\}$ attribútumhalmazt. *Reláció*nak nevezünk egy $T = \{t_1,...,t_k\}$ halmazt, ahol $t_i: A \rightarrow D$ leképezés, amelynél minden j-re $t_i(A_j) \in dom(A_j)$ teljesül.

Több adattábla együttesen alkotja a *relációs adatbázist*, amely egy teljes jelenségkör leírására alkalmas. A könyvtári nyilvántartás egy lehetséges megvalósítását a 12. ábra mutatja: itt a Könyv táblában adjuk meg az adott könyvet kikölcsönző olvasó számát és a kivétel dátumát. Ha egy könyvet éppen nem kölcsönöztek ki, akkor a megfelelő mezők NULL értékűek (a 12. ábrán egyszerűen üresen hagytuk ezeket).

A Könyv tábla:

Könyvszám	*Szerző*	*Cím*	*Olvasószám*	*Kivétel*
1121	Sályi	Adatbázisok		
3655	Radó	Világatlasz	122	2012.07.12
2276	Karinthy	Így írtok ti		
1782	Jókai	Aranyember	355	2012.09.23

Az Olvasó tábla:

Olvasószám	*Név*	*Lakcím*
122	Kiss István	Szeged, Virág u. 10.
612	Nagy Ágnes	Szentes, Petőfi út 38.
355	Tóth András	Budapest, Jég u. 3.

12. ábra: A könyvtári nyilvántartás 1. ill. 2. változatát megvalósító adatbázis

A 12. ábrán jól látható, hogy az *olvasószám* attribútum mindkét táblában szerepel, ezzel kapcsolatot létesít a táblák között. Ez rávilágít a következőre:

A relációs adatmodell lényege, hogy a különböző relációsémák azonos attribútumokat tartalmazhatnak, ezáltal kerülnek kapcsolatba egymással, és így a különálló adattáblák együttese egy szervesen összefüggő adatbázist alkot.

3.2. Kulcsok

11. Definíció. *Szuperkulcs*nak nevezünk egy attribútumhalmazt, ha egyértelműen azonosítja a tábla sorait. Pontosabban: egy $R(A_1,...,A_n)$ relációséma esetén az $A = \{A_1,...,A_n\}$ attribútumhalmaz egy K részhalmaza *szuperkulcs*, ha bármely R feletti T tábla bármely két sora K-n különbözik.

Formálisan: bármely $t_i \in T$ és $t_j \in T$ esetén $t_i \neq t_j \Rightarrow t_i(K) \neq t_j(K)$. Szemléletesen: ha a táblán a K-n kívüli oszlopokat letakarjuk, akkor is minden sor különböző marad.

12. Példa. A Könyv (könyvszám, szerző, cím) sémában a {könyvszám} szuperkulcs, de szuperkulcs például a {könyvszám, szerző} vagy a {könyvszám, cím} attribútumhalmaz is.

Megjegyzendő, hogy a teljes *A* attribútumhalmaz mindig szuperkulcs, hiszen definíció szerint a tábla minden sora különböző.

13. definíció. Az *A* attribútumhalmaz *K* részhalmazát *kulcs*nak nevezzük, ha minimális szuperkulcs, vagyis egyetlen valódi részhalmaza sem szuperkulcs. Ha *K* egyetlen attribútumból áll, akkor *egyszerű*, egyébként *összetett* kulcsról beszélünk.

Ha egy relációsémának több kulcsa is van, egyet kiválasztunk közülük, ez lesz az *elsődleges kulcs*. (Ha csak egy kulcs van, akkor szükségképpen az lesz az elsődleges kulcs.) Egy relációsémában tehát mindig csak egy elsődleges kulcs lehet.

Jelölés: az elsődleges kulcsot alkotó attribútumokat aláhúzással szokás jelölni.

Megjegyzés: A kulcs nem a tábla tulajdonsága, hanem egy *feltétel előírása* a relációsémára: annak megkövetelése, hogy a sémához tartozó táblában (annak bármely

időpontbeli állapotában) nem lehet két azonos kulcsú sor. A kulcs meghatározása az attribútumok *jelentésének* vizsgálatával lehetséges, és nem egy adott tábla vizsgálatával. Például a 12. ábrán látható Könyv tábla esetén a *cím* vagy *szerző* attribútumok is minden sorban különbözők, de tudjuk, hogy ez nem garantálható a Könyv tábla mindenkori állapotára.

14. Példa. Az alábbi tábla gépkocsik mozgásának menetlevél-szerű nyilvántartását tartalmazza:

Fuvar (gkvez, rendszám, indul, érkezik)

Itt négy összetett kulcs van: {gkvez, indul}, {gkvez, érkezik}, {rendszám, indul}, {rendszám, érkezik}. Ezek közül önkényesen kiválasztunk egyet, ez lesz az elsődleges kulcs:

Fuvar (gkvez, <u>rendszám</u>, <u>indul</u>, érkezik)

15. definíció. Egy relációséma attribútumainak valamely részhalmaza *külső kulcs (*másnéven *idegen kulcs,* angolul *foreign key)*, ha egy másik séma elsődleges kulcsára hivatkozik. A külső kulcs értéke a hivatkozott táblában előforduló kulcsérték vagy NULL lehet.

> Formálisan: legyenek $R_1(A)$, $R_2(B)$ relációsémák. Az $L (\subseteq A\}$ külső kulcs az R_1-ben R_2-re vonatkozóan, ha
> - R_2 elsődleges kulcsa K, és dom(K) = dom(L),
> - bármely R_1, R_2 feletti T_1, T_2 táblák esetén L értéke T_1 bármely sorában T_2-ben előforduló K-érték vagy NULL.

Jelölés: a külső kulcsot dőlt betűvel, vagy a hivatkozott kulcsra mutató nyíllal jelöljük.

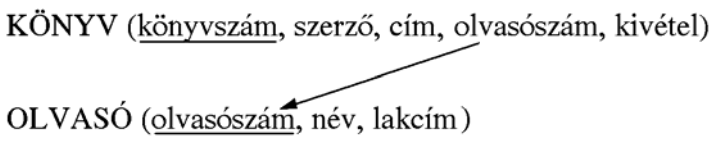
A kulcshoz hasonlóan a külső kulcs is *feltétel előírása* a sémákra, és nem az aktuális táblák tulajdonsága. A külső kulcs feltételek biztosítják az ún. *hivatkozási integritást* az adatbázisban.

16. Definíció. Ha egy adatbázis valamennyi táblájának sémáját felírjuk a kulcsok és külső kulcsok jelölésével együtt, akkor *relációs adatbázissémát* kapunk.

17. Példa. A könyvtári nyilvántartás relációs adatbázissémája:

Könyv (<u>könyvszám</u>, szerző, cím, *olvasószám*, kivétel)
Olvasó (<u>olvasószám</u>, név, lakcím)

vagy más jelölésmóddal:



3.3. Indexek

Az index nem része a relációs modellnek, hanem kiegészítő adatstruktúra, amelyet egy táblához lehet generálni. Fő céljai:

- *Keresések gyorsítása*. Ha például adott olvasószámnak megfelelő rekordot keressük, ehhez ne kelljen valamennyi rekordot végignézni.
- *Rendezés*. Listázáskor illetve feldolgozáskor gyakran szeretnénk valamilyen szempont szerint rendezve kezelni a rekordokat (például olvasó neve szerint ábécé rendben), függetlenül a fizikai adattárolás sorrendjétől.

Az indexet a tábla attribútumainak valamely *L* részhalmazához generáljuk, ezt *indexkulcs*nak nevezzük. Megegyezhet a tényleges kulccsal, de bármi más is lehet. Az index segítségével a tábla sorai *L* szerinti rendezésben kezelhetők.

Az indexet is táblaként lehet elképzelni, amelynek első oszlopa az indexkulcsot, a második a megfelelő rekord fizikai sorszámát (a gyakorlatban inkább a rekord fizikai címét a merevlemezen) tartalmazza (13. ábra).

Könyvszám	*Szerző*	*Cím*	*Olvasószám*	*Kivétel*
1121	Sályi	Adatbázisok		
2276	Karinthy	Így írtok ti		
3655	Radó	Világatlasz	122	2012.07.12
1782	Jókai	Aranyember	355	2012.09.23

Szerző	*Index*
Jókai	4
Karinthy	2
Radó	3
Sályi	1

Cím	*Index*
Adatbázisok	1
Aranyember	4
Így irtok ti	2
Világatlasz	3

13. ábra. A Könyv táblához létrehozott szerző szerinti ill. cím szerinti indextábla

Az index konkrét megvalósítása DBMS-enként változik. Az indextábla általában úgynevezett *B-fa* (B = balanced = kiegyensúlyozott) struktúrában kerül tárolásra, amely a bináris keresőfa általánosítása. Tulajdonságai:

- egy csomópontnak kettőnél több gyermeke lehet,
- minden módosítás után kiegyensúlyozott marad (a keresés mélysége bármely ágon ugyanannyi).

A B-fát általában mágneslemezen tárolják (kivéve a gyökér csomópontot, amely tartósan a memóriában lehet). Egy csomópont *egy lemezblokkot* foglal el, ezért akár száz gyermekre mutató pointert is tartalmazhat. A keresés ritkán mélyebb 3 szintnél. Mivel a keresés idejében a lemezolvasás a meghatározó, így a gyakorlatban konstans keresési idővel számolhatunk.

Index létrehozása viszonylag lassú, hiszen ekkor végig kell menni a teljes táblán. A folyamatot úgy képzelhetjük el, hogy az i-edik rekordhoz egy (z_i, i) párt generálnak, ahol z_i az

L indexkulcs értéke az adott rekordban, i pedig a rekord fizikai sorszáma, és ezt a (z_i, i) párt fűzik fel a fára.

Index használata.

- Az elkészült indexben L adott értékéhez (például a 2276 könyvszámhoz) gyorsan előkereshető a megfelelő fizikai rekord sorszáma.
- A tábla rendezett listázásához a B-fát kell bejárni.
- Ha a táblába új rekordot veszünk fel, ez mindig a tábla végére kerül, egyidejűleg a (z_i,i) pár beszúrásra kerül az indexbe.
- Ha rekordot törlünk a táblából, a megfelelő indexbejegyzés törlődik, de a táblában a rekord helye üresen marad, így a rekordok fizikai címei nem változnak meg.

Egy táblához *egyszerre több index* is létrehozható, például a könyveket indexelhetjük könyvszám, szerző és cím szerint is. A rekordokat a képernyőn mindig aszerint látjuk rendezve, hogy a lekérdezésnél melyik mező szerinti rendezettséget kérjük. Ilyenkor automatikusan a megfelelő index lép működésbe, miközben a rekordok fizikai sorrendje mindvégig változatlan marad.

3.4. E-K diagramból relációs adatbázisséma készítése

Egyedek leképezése

Szabály: az E-K modell minden egyedéhez felírunk egy relációsémát, amelynek neve az egyed neve, attribútumai az egyed attribútumai, elsődleges kulcsa az egyed kulcs-attribútuma(i). A séma feletti adattábla minden egyes sora egy egyedpéldánynak felel meg.

18. Példa. A 2. ábra szerinti könyvtári nyilvántartás esetén a könyveket egy Könyv táblában tarthatjuk nyilván, amely az alábbi séma szerint épül fel:

Könyv (<u>könyvszám</u>, szerző, cím)

Az olvasók nyilvántartására egy Olvasó nevű tábla szolgálhat, amelynek sémája:

Olvasó (<u>olvasószám</u>, név, lakcím)

Gyenge entitások leképezése

Szabály: a gyenge entitás relációsémáját bővíteni kell a meghatározó kapcsolat(ok)ban szereplő egyed(ek) kulcsával.

19. Példa. A 9. ábra szerinti számítógép nyilvántartás adatbázissémája a következő:

Tulajdonos (<u>személyiszám</u>, név, lakcím)
Számítógép (<u>processzor</u>, <u>memória</u>, <u>merevlemez</u>, <u>*személyiszám*</u>)

Ha egy tulajdonosnak több, azonos gépe lehet, akkor ezeket egy *sorszám* attribútummal különböztetjük meg:

Tulajdonos (<u>személyiszám</u>, név, lakcím)
Számítógép (processzor, memória, merevlemez, <u>*személyiszám*</u>, <u>sorszám</u>)

20. Példa. A 10. ábrán látható *Szerződés* egyed leképezése:

Szerződés (fizetés, <u>*ISBN, szerzőnév*</u>)

Összetett attribútumok leképezése

Tegyük fel, hogy az Olvasó táblában a *lakcím* attribútumot (helység, utca, házszám) struktúraként szeretnénk kezelni. Relációs adatmodellben erre egyetlen lehetőség van: az

Olvasó (olvasószám, név, lakcím)

séma helyett a

Olvasó (olvasószám, név, helység, utca, házszám)

sémára térünk át, a megfelelő tábla a következő:

Olvasószám	*Név*	*Helység*	*Utca*	*Házszám*
122	Kiss István	Szeged	Virág u.	10
612	Nagy Ágnes	Szentes	Petőfi út	38
355	Tóth András	Budapest	Jég u.	3

Többértékű attribútumok leképezése

Kérdés, hogy többszerzős könyveket hogyan tartsunk nyilván az adatbázisban. Példaként a Könyv táblát vizsgáljuk, amelynél a 1121 számú könyvnek valójában két szerzője van: Sályi János és Szelezsán János. Alább sorra vesszük a lehetőségeket.

1. Megadás egyértékű attribútumként. A szerző megadására szolgáló szövegmezőben felsoroljuk a szerzőket.

Hátrányok:

- a szerzőket külön-külön nem tudjuk kezelni.
- sok szerző esetleg nem fér el a megadott mezőben

2. Sorok többszörözése. A Könyv táblában egy könyvhöz annyi sort veszünk fel, ahány szerzője van:

Könyvszám	*Szerző*	*Cím*
1121	Sályi	Adatbázisok
1121	Szelezsán	Adatbázisok
3655	Radó	Világatlasz
2276	Karinthy	Így írtok ti
1782	Jókai	Aranyember

A megfelelő relációséma: Könyv (<u>könyvszám</u>, <u>szerző</u>, cím)

A fenti megoldás hátránya, hogy a többszerzős könyvek címét több példányban kell megadni. Ez redundanciát jelent, tehát ez nem jó megoldás.

3. Új tábla felvétele. A Könyv (könyvszám, szerző, cím) sémát az alábbi két sémával helyettesítjük:

Könyv (<u>könyvszám</u>, cím)
Szerző (<u>*könyvszám*</u>, <u>szerző</u>)

A megfelelő adattáblák a következők:

Könyvszám	*Cím*
1121	Adatbázisok
3655	Világatlasz
2276	Így írtok ti
1782	Aranyember

Könyvszám	*Szerző*
1121	Sályi
1121	Szelezsán
3655	Radó
2276	Karinthy
1782	Jókai

Bár ez a megvalósítás bonyolultabbnak tűnik, később látni fogjuk, hogy ez a korrekt megoldás. Ha a szerzők sorrendje fontos, akkor a Szerző táblát egy *sorszám* mezővel kell bővíteni (emlékeztetünk rá, hogy a relációs adatmodell nem definiálja a rekordok sorrendjét):

Könyv (könyvszám, cím)
Szerző (*könyvszám*, sorszám, szerző)

Kapcsolatok leképezése

Általános szabály:

1. Vegyünk fel a kapcsolathoz egy új sémát, amelynek neve a kapcsolat neve, attribútumai pedig a kapcsolódó entitások kulcs attribútumai és a kapcsolat saját attribútumai.

> Formálisan, ha az összekapcsolt egyedeknek az $R_1(K_1 \cup B_1)$, ..., $R_n(K_n \cup B_n)$ sémák felelnek meg (K_i a kulcs, B_i a további attribútumok halmaza), akkor a kapcsolatnak egy $R(K_1 \cup ... \cup K_n \cup B)$ sémát feleltetünk meg, ahol B a kapcsolat saját attribútumai. R-ben K_i külső kulcs hivatkozás az R_i sémára. Az R feletti adattábla minden egyes sora egy kapcsolat-példánynak felel meg.

2. Ha ezen séma kulcsa megegyezik valamely kapcsolódó egyed kulcsával, akkor a kapcsolat sémája és az egyed sémája összevonható (az attribútumok unióját képezve).

21. Példa. A 2. ábrán szereplő "kölcsönzés" kapcsolat esetén az alábbi sémát kapjuk:

Kölcsön (könyvszám, olvasószám, kivétel, visszahozás)

Kérdés, hogy mi lesz a kulcs ebben a táblában. Ehhez a kapcsolat típusát kell megvizsgálni. Nézzük meg sorra az előzőekben tárgyalt három változatot!

1. változat: Ha egy olvasónak egyszerre csak egy könyvet adnak ki, akkor a kölcsönzés 1:1 kapcsolatot jelent. Ilyenkor a Kölcsön sémában a *könyvszám* és az *olvasószám* egyaránt kulcs. Továbbá, a *visszahozás* attribútumra nincs szükségünk, mivel a könyv visszahozásával a könyv-olvasó kapcsolat megszűnik. Tehát, a

Kölcsön (könyvszám, olvasószám, kivétel)

vagy a

Kölcsön (könyvszám, olvasószám, kivétel)

sémát vehetjük fel a kapcsolathoz. Az első változat kulcsa a Könyv sémáéval, a másodiké az Olvasó sémáéval egyezik meg. A Kölcsön sémát az azonos kulcsú sémába olvasztva a

Könyv (könyvszám, szerző, cím, *olvasószám*, kivétel)
Olvasó (olvasószám, név, lakcím)

vagy a

Könyv (könyvszám, szerző, cím)
Olvasó (olvasószám, név, lakcím, *könyvszám*, kivétel)

adatbázissémákat kapjuk. A megfelelő táblák a 12. és 14. ábrán láthatók. Ha egy könyvet éppen senki sem kölcsönzött ki, illetve ha egy olvasónál éppen nincs könyv, akkor a megfelelő mezők üresen maradnak (azaz NULL értékűek).

A Könyv tábla:

Könyvszám	*Szerző*	*Cím*
1121	Sályi	Adatbázisok
3655	Radó	Világatlasz
2276	Karinthy	Így írtok ti
1782	Jókai	Aranyember

Az Olvasó tábla:

Olvasószám	*Név*	*Lakcím*	*Könyvszám*	*Kivétel*
122	Kiss István	Szeged, Virág u. 10.	3655	2012.07.12
612	Nagy Ágnes	Szentes, Petőfi út 38.		
355	Tóth András	Budapest, Jég u. 3.	1782	2012.09.23

14. ábra. Könyvtári nyilvántartás abban az esetben, ha egy olvasó egyszerre csak egy könyvet kölcsönözhet ki

2. változat: Ha egy olvasó több könyvet is kikölcsönözhet, akkor a könyv-olvasó kapcsolat N:1 típusú. Ekkor a Kölcsön sémában csak a *könyvszám* lehet kulcs, ezért a Kölcsön sémát csak a Könyv sémába olvaszthatjuk:

Könyv (<u>könyvszám</u>, szerző, cím, *olvasószám*, kivétel)
Olvasó (<u>olvasószám</u>, név, lakcím)

A megfelelő táblák a 12. ábrán láthatók, azzal a különbséggel, hogy most több könyvnél is szerepelhet ugyanazon olvasó száma. A 14. ábra szerinti lehetőség, vagyis hogy az Olvasó táblát bővítjük *könyvszám* és *kivétel* oszloppal, már nem járható. Ugyanis egy olvasóhoz több könyvszámot kellene beírnunk, ami ellentmond a relációs adatmodell alapelvének: az adattábla egy mezőjébe csak *atomi értéket* lehet beírni.

3. változat: Ha az egyes könyvek korábbi kölcsönzéseit is nyilvántartjuk, akkor nem csak egy olvasóhoz tartozhat több könyv, hanem egy könyvhöz is több olvasó (N:M kapcsolat), sőt adott olvasó adott könyvet egymás után többször is kikölcsönözhet. Ezért a Kölcsön sémában

{könyvszám, kivétel}

vagy

{könyvszám, visszahozás}

a kulcs, a Kölcsön táblát most sem a Könyv, sem az Olvasó táblába nem tudjuk beolvasztani. Az adatbázisséma ezért a következő:

Könyv (<u>könyvszám</u>, szerző, cím)
Olvasó (<u>olvasószám</u>, név, lakcím)
Kölcsön (<u>*könyvszám*</u>, *olvasószám*, <u>kivétel</u>, visszahozás)

A Könyv tábla:

Könyvszám	*Szerző*	*Cím*
1121	Sályi	Adatbázisok
3655	Radó	Világatlasz
2276	Karinthy	Így írtok ti
1782	Jókai	Aranyember

Az Olvasó tábla:

Olvasószám	*Név*	*Lakcím*
122	Kiss István	Szeged, Virág u. 10.
612	Nagy Ágnes	Szentes, Petőfi út 38.
355	Tóth András	Budapest, Jég u. 3.

A Kölcsön tábla:

Könyvszám	*Olvasószám*	*Kivétel*	*Visszahozás*
1121	355	2005.11.02	
1121	612	2003.11.14	2004.01.03
1121	122	2005.02.22	2005.04.17
3655	122	2005.07.12	
2276	612	2004.03.16	2004.04.02
1782	355	2005.09.23	

15. ábra: A könyvtári adatbázis 3. változata

A fentiek alapján az *alábbi szabályok* fogalmazhatók meg két egyed közötti kapcsolatok leképezésére relációs modellbe:

a) 1:1 kapcsolat esetén kiválasztjuk a kapcsolatban résztvevő két entitás egyikét (bármelyiket), és annak sémájába új attribútumként felvesszük a másik entitás kulcs attribútumait, valamint a kapcsolat attribútumait.

b) 1:N kapcsolat esetén az „N” oldali entitás sémájába új attribútumként felvesszük a másik entitás kulcs attribútumait, valamint a kapcsolat attribútumait.

c) N:M kapcsolat esetén új sémát veszünk fel, amelynek attribútumai

- a kapcsolódó entitások kulcs attribútumai,
- a kapcsolat saját attribútumai.

> *Megjegyzés.* Előfordul, hogy 1:1 illetve 1:N kapcsolat esetén sem érdemes a kapcsolat sémáját beolvasztani a megfelelő egyed sémájába. Ha például a Könyv táblát bővítjük *olvasószám* és *kivétel* oszloppal, de a könyveknek csak elenyészően kis százaléka van adott pillanatban kikölcsönözve, akkor *olvasószám* és *kivétel* attribútumok értéke majdnem minden sorban NULL lesz. Ez a redundancia megszűnik, ha a kölcsönzéseket egy külön Kölcsön (*könyvszám*, *olvasószám*, kivétel) táblában tartjuk nyilván.

22. Példa. A 5. ábra szerinti szervezeti egység nyilvántartás önmagával kapcsolatban álló egyedet tartalmaz. Lényegében itt is a fenti *b)* szabályt alkalmazhatjuk, vagyis az

Egység (egységkód, megnevezés)

sémát kell bővíteni *egységkód* attribútummal. Mivel egy sémában nem szerepelhet két azonos attribútumnév, ezért az új attribútumot *főegységkód*nak nevezzük:

Egység (egységkód, megnevezés, *főegységkód*)

ahol *főegység* a fölérendelt szervezeti egység kódja.

23. Példa. Az általános szabály alapján felírhatjuk a 6. ábra szerinti E-K modell relációs adatbázissémáját:

Könyv (cím, év, ISBN)
Szerző (név, lakcím)
Kiadó (név, cím)
Szerződés (*ISBN*, *szerzőnév*, *kiadónév*, fizetés)

Az azonos nevek ütközésének elkerülésére a Szerződés sémában módosított attribútumneveket alkalmaztunk. Mivel a Szerződés kapcsolatban a könyv és a szerző már meghatározza a kiadót (lásd a nyilat a 6. ábrán), ezért a kiadónév már nem része a kulcsnak. Ha a 10. ábra szerinti szétbontott változat sémáját írjuk fel, akkor is a fenti adatbázissémához jutunk.

Specializáló kapcsolatok leképezése

A relációs megvalósítási lehetőségeket a 11. ábra szerinti E-K modellen mutatjuk be.

1. Minden altípushoz külön tábla felvétele, egy egyed csak egy táblában szerepel. Az altípusok öröklik a főtípus attribútumait.

Helyiség (épület, ajtószám, név, alapterület)
Tanterem (épület, ajtószám, név, alapterület, férőhely, tábla, vetítő)
Gépterem (épület, ajtószám, név, alapterület, gépszám)
Iroda (épület, ajtószám, név, alapterület, telefon, fax)
Dolgozó (adószám, név, lakcím, *épület, ajtószám*)

Hátrányok:
- Kereséskor gyakran több táblát kell vizsgálni (ha például a Központi épület 211. sz. terem alapterületét keressük).
- Kombinált altípus (például számítógépes tanterem) csak új altípus felvételével kezelhető.

2. Minden altípushoz külön tábla felvétele, egy egyed több táblában is szerepelhet. A főtípus táblájában minden egyed szerepel, és annyi altípusában ahánynak megfelel. Az altípusok a főtípustól csak a kulcs-attribútumokat öröklik.

Helyiség (épület, ajtószám, név, alapterület)
Tanterem (*épület, ajtószám*, férőhely, tábla, vetítő)
Gépterem (*épület, ajtószám*, gépszám)
Iroda (*épület, ajtószám*, telefon, fax)
Dolgozó (adószám, név, lakcím, *épület, ajtószám*)

Hátrány: Itt is előfordulhat, hogy több táblában kell keresni (például ha a tantermek nevére és férőhelyére vagyunk kíváncsiak).

3. Egy közös tábla felvétele, az attribútumok uniójával. Az aktuálisan értékkel nem rendelkező attribútumok NULL értékűek.

Helyiség (épület, ajtószám, név, alapterület, férőhely, tábla, vetítő, gépszám, telefon, fax)
Dolgozó (adószám, név, lakcím, *épület, ajtószám*)

Hátrányok:

- Az ilyen egyesített táblában általában sok NULL attribútumérték szerepel.
- Elveszíthetjük a típusinformációt (például ha a gépteremnél a gépszám nem ismert és ezért NULL, akkor a gépterem lényegében az egyéb helyiségek kategóriájába kerül). Ez a hátrány kiküszöbölhető egy *típus* attribútum felvételével.

4. Relációs algebra

A relációs algebra adattáblákon végzett műveletek rendszere, amely az adatbázis lekérdezés matematikai alapját képezi.

4.1. Halmazműveletek

Itt az adattáblát (relációt) sorok halmazaként kezeljük.

24. Definíció. Két táblát *kompatibilis*nek nevezünk, ha sémáik megegyeznek, vagy csak az attrinútumok elnevezésében különböznek.

Pontosabban: Az $R_1(A_1,...,A_n)$ és $R_2(B_1,...,B_m)$ relációsémák *kompatibilis*ek, ha $n = m$ és $dom(A_i) = dom(B_i)$ minden i-re. Két táblát kompatibilisnek nevezünk, ha sémáik kompatibilisek.

A halmazműveleteket csak kompatibilis táblákon értelmezzük.

Unió

A T_1 és T_2 kompatibilis táblák halmazelméleti egyesítése a $T = T_1 \cup T_2$ tábla lesz, amelynek sémája szintén kompatibilis T_1 ill. T_2 sémájával. A művelet végrehajtása:

- a két tábla egymás után írása,
- ismétlődő sorok kiszűrése.

25. Példa. Legyen két könyvtár állománya az alábbi táblákban tárolva:

Könyv1 tábla:

szerzőnév	*cím*
Sályi	Adatbázisok
Jókai	Aranyember
Radó	Világatlasz

Könyv2 tábla:

szerző	*könyvcím*
Jókai	Aranyember
Karinthy	Így írtok ti

ahol dom(szerzőnév) = dom(szerző) és dom(cím) = dom(könyvcím), tehát a táblák kompatibilisek. Ekkor a két tábla uniója az alábbi (az attribútumneveket szabadon választhatjuk meg):

Könyv1 U Könyv2 tábla:

szerző	*cím*
Sályi	Adatbázisok
Jókai	Aranyember
Radó	Világatlasz
Karinthy	Így írtok ti

Metszet (Intersection)

Két kompatibilis tábla halmazelméleti metszete azokat a sorokat tartalmazza, amelyek mindkét táblában előfordulnak: $T = T_1 \cap T_2$.

26. Példa. Az előbbi példát tekintve, a két könyvtár állományának metszete:

Könyv1 ∩ Könyv2 tábla:

szerző	*cím*
Jókai	Aranyember

Különbség (Difference)

A T_1 és T_2 kompatibilis táblák halmazelméleti különbsége azokat a sorokat tartalmazza, amelyek T_1-ben szerepelnek, de T_2-ben nem: $T = T_1 - T_2$.

27. Példa. Az előbbi példát tekintve, a két könyvtár állományának különbsége:

Könyv1 – Könyv2 tábla:

szerző	*cím*
Sályi	Adatbázisok
Radó	Világatlasz

Tulajdonságok: az unió és metszet kommutatív, a különbség nem.

4.2. Redukciós műveletek

Projekció (vetítés)

Adott oszlopok kiválasztása a táblából. Az új tábla sémája a megfelelő attribútumok kiválasztásával adódik.

Jelölése: $\pi_{\text{attribútumlista}}(\text{tábla})$

28. Példa: A Könyv1 = $\pi_{\text{szerző,cím}}(\text{Könyv})$ tábla:

Szerző	*Cím*
Sályi	Adatbázisok
Radó	Világatlasz
Karinthy	Így írtok ti
Jókai	Aranyember

Ha az attribútumlista nem tartalmazza a kulcsot, akkor a rekordok száma csökkenhet. Például, ha két könyv szerzője és címe megegyezik (ugyanazon könyv különböző példányai), akkor a Könyv1 táblában összevonásra kerülnek.

Szelekció (kiválasztás)

Adott feltételnek eleget tevő sorok kiválasztása a táblából. A feltétel általában attribútumokból és konstansokból felépülő logikai kifejezés. Az eredménytábla sémája megegyezik (vagy kompatibilis) az eredetivel.

Jelölés: $\sigma_{\text{feltétel}}(\text{tábla})$

29. Példa: Tekintsük az alábbi Könyv táblát:

Könyvszám	*Szerző*	*Cím*	*Olvasószám*	*Kivétel*
1121	Sályi	Adatbázisok	612	2012.03.15
3655	Radó	Világatlasz	122	2013.07.12
2276	Karinthy	Így írtok ti		
1782	Jókai	Aranyember	355	2012.09.23

Ekkor a $\sigma_{\text{kivétel}<2013.01.01}$(Könyv) tábla:

Könyvszám	*Szerző*	*Cím*	*Olvasószám*	*Kivétel*
1121	Sályi	Adatbázisok	612	2012.03.15
1782	Jókai	Aranyember	355	2012.09.23

Vegyük észre: az a sor, ahol *kivétel* értéke NULL, nem kerül kiválasztásra.

A szelekció kommutatív: $\sigma_{f1}(\sigma_{f2}(\text{tábla})) = \sigma_{f2}(\sigma_{f1}(\text{tábla})) = \sigma_{(f1\ \text{AND}\ f2)}(\text{tábla})$

4.3. Kombinációs műveletek

Descartes-szorzat

Tetszőleges T_1 és T_2 táblák $T = T_1 \times T_2$ Descartes-szorzatát úgy kapjuk, hogy T_1 minden sorát párosítjuk T_2 minden sorával.

> Pontosabban: Legyen $R_1(A_1,...,A_n)$, $R_2(B_1,...,B_m)$ két tetszőleges relációséma, és $T_1 \subseteq \text{dom}(A_1) \times ... \times \text{dom}(A_n)$, $T_2 \subseteq \text{dom}(B_1) \times ... \times \text{dom}(B_m)$ táblák R_1, R_2 felett. A $T = T_1 \times T_2$ Descartes-szorzat az $R(A_1,...,A_n,B_1,...,B_m)$ séma feletti $T \subseteq \text{dom}(A_1) \times ... \times \text{dom}(A_n) \times \text{dom}(B_1) \times ... \times \text{dom}(B_m)$ tábla, amelyet úgy kapunk, hogy T_1 minden sorát párosítjuk T_2 minden sorával.
>
> Ha R_1 és R_2 attribútumai között azonos nevűek vannak, akkor R-ben az eredeti séma nevével különböztetjük meg őket (például $R_1.A_i$, $R_2.A_i$).

30. Példa:

T_1: A_1	A_2	A_3
a	b	c
b	d	e
f	c	b

T_2: B_1	B_2	B_3
b	d	e
a	d	b

$T_1 \times T_2$: A_1	A_2	A_3	B_1	B_2	B_3
a	b	c	b	d	e
a	b	c	a	d	b
b	d	e	b	d	e
b	d	e	a	d	b
f	c	b	b	d	e
f	c	b	a	d	b

Ha T_1 és T_2 sorainak száma r_1 ill. r_2, oszlopainak száma c_1 és c_2, akkor a T táblában $r_1 * r_2$ sor és $c_1 + c_2$ oszlop van.

Ha két tábla Descartes-szorzatát képezzük, akkor projekcióval visszakaphatók az eredeti táblák: $\pi_{A1,...An}(T) = T_1$ és $\pi_{B1,...,Bm}(T) = T_2$.

A Descartes-szorzat műveletet nem szokták alkalmazni a gyakorlatban, hiszen az adathalmaz redundanciáját növeli, az összekapcsolási műveletek definiálásánál azonban szükségünk lesz rá.

Természetes összekapcsolás (Natural join)

A relációs modell lényegéhez tartozik, hogy két tábla között a megegyező attribútumok létesítenek kapcsolatot. Általában, tekintsük az A és B attribútumhalmazok feletti $R_1(A)$ és $R_2(B)$ sémákat, ahol $X = A \cap B$ nem üres. Az R_1 és R_2 feletti T_1 és T_2 táblák természetes összekapcsolása egy R(A U B) feletti T tábla, amelyet a következőképp definiálunk:

$$T = \pi_{A \cup B}(\sigma_{R1.X=R2.X}(T_1 \times T_2))$$

Vagyis, a két tábla Descartes-szorzatából kiválasztjuk azokat a sorokat, amelyek az $R_1.X$ és $R_2.X$ attribútumokon megegyeznek, majd a projekcióval a duplán szereplő X-beli attribútumokat csak egy példányban tartjuk meg (az A U B halmazelméleti unió, vagyis benne az X elemei csak egyszeresen szerepelnek).

Jelölés: $T = T_1 * T_2$

31. Példa. A gyakorlatban általában külső kulcs alapján végeznek természetes összekapcsolást. Tekintsük a könyvtári nyilvántartás adatbázissémáját:

Könyv (<u>könyvszám</u>, szerző, cím, *olvasószám*, kivétel)

Olvasó (<u>olvasószám</u>, név, lakcím)

Ha most a kikölcsönzött könyvek listáját szeretnénk megkapni, de az olvasószám mellett az olvasó nevének és lakcímének a feltüntetésével, akkor ez a

Kolv = Könyv * Olvasó

természetes összekapcsolás művelettel végezhető el, ahol az eredményül kapott tábla a 12. ábra szerinti adatbázis esetén

K.szám	*Szerző*	*Cím*	*O.szám*	*Kivétel*	*Név*	*Lakcím*
3655	Radó	Világatlasz	122	2012.07.12	Kiss István	Szeged, Virág u.10
1782	Jókai	Aranyember	355	2012.09.23	Tóth András	Budapest, Jég u.3.

Megjegyzés: ha $T=T_1*T_2$, akkor T-ből projekcióval általában nem állítható elő T_1 ill. T_2. Például, a fenti Kolv tábla csak a kikölcsönzött könyveket tartalmazza, mivel a ki nem kölcsönzötteknél a Könyv táblában az olvasószám értéke NULL.

Külső összekapcsolás (Outer join)

A természetes összekapcsolás veszélye, hogy általában a kapcsolt táblák nem minden sora szerepel az eredménytáblában. Ha egy sor nem párosítható a másik tábla egyetlen sorával sem, akkor *lógó sor*nak nevezzük.

Ha például Könyv táblában téves olvasószám szerepel, akkor a fenti Kolv táblában az adott könyv nem fog szerepelni. További természetes igény lehet, hogy a Kolv táblában ne csak a kikölcsönzött könyveket, hanem az összes könyvet lássuk.

A fentiek miatt használatos a *külső összekapcsolás* (outer join) művelet, amely az összekapcsolt két tábla egyikénél vagy mindkettőnél valamennyi rekord megőrzését garantálja. Jelölésére az Oracle rendszer (+) konvencióját használjuk:

Bal oldali külső összekapcsolás: T_1 (+)* T_2. Azt jelenti, hogy az eredménytáblában T_1 azon sorai is szerepelnek, amelyek T_2 egyetlen sorával sem párosíthatók. Ezen sorokban a T_2-beli attribútumok értéke NULL.

Jobb oldali külső összekapcsolás: T_1 *(+) T_2. Hasonlóan a T_2 táblára.

Teljes külső összekapcsolás: T_1 (+)*(+) T_2. Itt mindkét tábla nem párosított rekordjai megőrződnek.

32. Példa. A Kolv1 = Könyv (+)* Olvasó tábla már a 12. ábra adatbázisának összes könyvét tartalmazza:

K.szám	*Szerző*	*Cím*	*O.szám*	*Kivétel*	*Név*	*Lakcím*
1121	Sályi	Adatbázisok				
3655	Radó	Világatlasz	122	2012.07.12	Kiss István	Szeged, Virág u.10
2276	Karinthy	Így írtok ti				
1782	Jókai	Aranyember	355	2012.09.23	Tóth András	Budapest, Jég u.3.

A Kolv2 = Könyv *(+) Olvasó táblában minden olvasó szerepel:

K.szám	*Szerző*	*Cím*	*O.szám*	*Kivétel*	*Név*	*Lakcím*
3655	Radó	Világatlasz	122	2012.07.12	Kiss István	Szeged, Virág u.10
			612		Nagy Ágnes	Szentes, Petőfi út 38
1782	Jókai	Aranyember	355	2012.09.23	Tóth András	Budapest, Jég u.3.

Végül a Kolv3 = Könyv (+)*(+) Olvasó tábla minden könyvet és olvasót tartalmaz:

K.szám	*Szerző*	*Cím*	*O.szám*	*Kivétel*	*Név*	*Lakcím*
1121	Sályi	Adatbázisok				
3655	Radó	Világatlasz	122	2012.07.12	Kiss István	Szeged, Virág u.10
2276	Karinthy	Így írtok ti				
			612		Nagy Ágnes	Szentes, Petőfi út 38
1782	Jókai	Aranyember	355	2012.09.23	Tóth András	Budapest, Jég u.3.

Külső összekapcsolás esetén már projekcióval visszakaphatók az eredeti táblák: bal oldali külső összekapcsolásnál $\pi_A(T) = T_1$, hasonlóan a többi esetre.

Théta-összekapcsolás (Theta-join)

Itt a táblák Descartes-szorzatából tetszőleges feltétel szerint választunk ki sorokat:

$T = \sigma_{feltétel}(T_1 \times T_2)$

Jelölése: $T = T_1 *_{feltétel} T_2$

33. Példa. Tegyük fel, hogy adott áruféleséget több raktár tárol, a raktározott mennyiséget egy

Raktár (<u>raktárkód</u>, mennyiség)

táblában, a vevők igényeit pedig egy

Vevő (<u>vevőkód</u>, igény)

táblában tartjuk nyilván. Az eladási ajánlatok egy

Ajánlat (raktárkód, mennyiség, vevőkód, igény)

táblába generálhatók az alábbi theta-join művelettel:

Ajánlat = Raktár $*_{\text{igény} \leq \text{mennyiség}}$ Vevő

4.4. Multihalmazok

Multihalmazon olyan halmazt értünk, amely ismétlődő elemeket is tartalmazhat (például {1, 3, 4} halmaz, de {1, 3, 1, 4} már multihalmaz). Ha a relációt multihalmaznak tekintjük, akkor ezzel az adattáblában azonos sorokat is megengedünk. A relációs algebra műveletei multihalmazokra is értelmezhetők, ennek részleteire itt nem térünk ki.

Az adatbázis-kezelő rendszerek általában multihalmazokkal dolgoznak, és csak külön kérésre végzik el az azonos sorok kiszűrését. Ennek okai a következők:

- Az adattábla fizikai tárolása természetes módon megengedi az azonos sorokat.
- Egyes relációs műveletek (például unió, projekció) lényegesen gyorsabbak, ha nem kell kiszűrni az azonos sorokat.
- Egyes esetekben a multihalmaz szolgáltat korrekt eredményt. Például, ha a Dolgozó (név, adószám, lakcím, fizetés) táblára a Dolg1 = $\pi_{\text{név,fizetés}}$(Dolgozó) projekciót végezzük, akkor feltehetően nem kívánjuk, hogy két azonos nevű és fizetésű személy összeolvadásra kerüljön.

A gyakorlatban tehát minden adatbázis-műveletnél el kell dönteni, hogy a relációs modell szerint halmazokkal, vagy (az RDBMS számára természetesebb) multihalmazokkal kívánunk dolgozni, és ennek megfelelően kell a műveleteket végrehajtani.

5. A relációs adatbázis normalizálása

Ha az egyed-kapcsolat modellt helyesen írjuk fel, akkor általában optimális (redundanciamentes) relációs adatbázis sémát kapunk. Semmi garancia nincs azonban arra, hogy az E-K modell optimális, ezért szükség van a relációsémák formális vizsgálatára, amely a redundanciákat detektálja és az optimalizálást lehetővé teszi (normalizálás). Ezen kérdéskör elméleti megalapozásával és gyakorlati módszereivel foglalkozik ez a fejezet.

5.1. Redundáns adattáblák

Tekintsük egy vállalat dolgozóit nyilvántartó

Dolgozó (név, adószám, cím, osztálykód, osztálynév, *vezAdószám*)

sémát, ahol *vezAdószám* a vállalati osztály vezetőjének adószámát jelenti. A megfelelő tábla a 16. ábrán látható.

Előny: egyetlen táblában a dolgozók és osztályok adatai is nyilvántartva.

Hátrány: redundancia, mivel *osztálynév, vezAdószám* több helyen szerepel.

Név	*Adószám*	*Cím*	*Osztálykód*	*Osztálynév*	*VezAdószám*
Kovács	1111	Pécs, Vár u.5.	2	Tervezési	8888
Tóth	2222	Tata, Tó u.2.	1	Munkaügyi	3333
Kovács	3333	Vác, Róka u.1.	1	Munkaügyi	3333
Török	8888	Pécs, Sas u.8.	2	Tervezési	8888
Kiss	4444	Pápa, Kő tér 2.	3	Kutatási	4444
Takács	5555	Győr, Pap u. 7.	1	Munkaügyi	3333
Fekete	6666	Pécs, Hegy u.5.	3	Kutatási	4444
Nagy	7777	Pécs, Cső u.25.	3	Kutatási	4444

16. ábra. Dolgozók nyilvántartását tartalmazó redundáns tábla

A redundancia aktualizálási anomáliákat okozhat:

(i) Módosítás esetén:

- Ha egy osztály neve vagy vezetője megváltozik, több helyen kell a módosítást elvégezni, ami hibákhoz vezethet.

(ii) Új felvétel esetén:

- Új dolgozó felvételénél előfordulhat, hogy az osztálynevet máshogy adják meg (például *Tervezési* helyett *tervezési* vagy *Tervező*).
- Ha új osztály létesül, amelynek még nincsenek alkalmazottai, akkor ennek adatait csak úgy tudnánk felvenni, ha a *név, adószám, cím* mezőkhöz NULL értéket rendelnénk (ami nem megengedett, mert *adószám* kulcs).

(iii) Törlés esetén:

- Ha egy osztály valamennyi dolgozóját töröljük, akkor az osztályra vonatkozó információk is elvesznek.

Megoldás: a relációséma felbontása két sémára (dekompozíció):
Dolg (név, adószám, cím, *osztálykód*)
Oszt (osztálykód, osztálynév, *vezAdószám*)

A szétválasztott táblák a 17. ábrán láthatók.

Név	*Adószám*	*Cím*	*Osztálykód*
Kovács	1111	Pécs, Vár u.5.	2
Tóth	2222	Tata, Tó u.2.	1
Kovács	3333	Vác, Róka u.1.	1
Török	8888	Pécs, Sas u.8.	2
Kiss	4444	Pápa, Kő tér 2.	3
Takács	5555	Győr, Pap u. 7.	1
Fekete	6666	Pécs, Hegy u.5.	3
Nagy	7777	Pécs, Cső u.25.	3

Osztálykód	*Osztálynév*	*VezAdószám*
1	Munkaügyi	3333
2	Tervezési	8888
3	Kutatási	4444

17. ábra. Redundancia megszüntetése a tábla felbontásával

Megjegyzés: Ha helyesen felírt E-K modellből indulunk ki, amely a *Dolgozó* és *Osztály* entitások között két kapcsolatot (*dolgozik* és *vezeti*) tartalmaz, akkor eleve a fenti két táblához jutunk.

A továbbiakban a relációséma formális vizsgálatával választ adunk a következő kérdésekre:
- mikor van redundancia egy táblában,
- hogyan kell ezt a tábla felbontásával megszüntetni.

5.2. Funkcionális függőség

34. definíció. Legyen $R(A_1,...,A_n)$ egy relációséma, és P, Q az $\{A_1,...,A_n\}$ attribútumhalmaz részhalmazai. P-től *funkcionálisan függ* Q (jelölésben $P \rightarrow Q$), ha bármely R feletti T tábla esetén valahányszor két sor megegyezik P-n, akkor megegyezik Q-n is, vagyis bármely $t_i \in T$ és $t_j \in T$ esetén

$$t_i(P) = t_j(P) \Rightarrow t_i(Q) = t_j(Q)$$

Elnevezések:
- A $P \rightarrow Q$ függést *triviális*nak nevezzük, ha $Q \subseteq P$, ellenkező esetben *nem triviális.*
- A $P \rightarrow Q$ függést *teljesen nemtriviális*nak nevezzük, ha $Q \cap P = 0$.

A gyakorlatban általában teljesen nemtriviális függőségeket adunk meg.

35. Példa. A korábban vizsgált
Dolgozó (Adószám, Név, Cím, Osztálykód, Osztálynév, VezAdószám)

tábla lényeges függőségei:
f_1: {Adószám} → {Név, Cím, Osztálykód}
f_2: {Osztálykód} → {Osztálynév, VezAdószám}

Példa további függőségekre, amelyek valójában a fentiekből következnek:
f_3: {Adószám} → {Osztálynév}
f_4: {Cím, Osztálykód} → {VezAdószám}

36. Példa. Egy számla tételeit tartalmazó
Számla (cikkszám, megnevezés, egységár, mennyiség, összeg)
tábla esetén az alábbi függőségeket állapíthatjuk meg:
{cikkszám} → {megnevezés, egységár}
{egységár, mennyiség} → {összeg}

Megjegyzések:

- A függőség nem az aktuális tábla, hanem a séma tulajdonsága. Ha az attribútumhalmazra megállapítunk egy funkcionális függőséget, akkor ez tulajdonképpen egy feltételt jelent az adattáblára nézve. Ha pl. Adószám → Cím funkcionális függőség fennáll, akkor egy személyhez több lakcímet nem tudunk tárolni.

- A "funkcionális" kifejezés arra utal, hogy ha P → Q fennáll, akkor létezik egy dom(P) → dom(Q) függvény, amely P minden konkrét értékéhez egyértelműen meghatározza Q értékét. Ez a függvény általában csak elméletileg létezik, pl. Adószám → Cím függés esetén nem tudunk olyan algoritmust adni, amely az adószámból a lakcímet előállítaná. A Számla tábla esetén azonban az {egységár, mennyiség} → {összeg} függőség már számítható, mivel egységár*mennyiség = összeg teljesül.

37. Állítás. Egy K ($\subseteq$ A) attribútumhalmaz akkor és csak akkor szuperkulcs, ha K→A.
Bizonyítás: a kulcs és a funkcionális függés definíciója alapján nyilvánvaló.

38. Definíció. Relációséma és adattábla fogalma függőség alapján:
*Relációséma*nak nevezünk egy R = (A, F) párt, ahol A = {A_1,...,A_n} attribútumhalmaz, és F = {f_1,...,f_m} az A-n definiált funkcionális függőségek egy halmaza (f_i: $P_i \to Q_i$, i=1,...,m). A függőségi halmaz olyan követelményrendszert definiál, amit eddig csak az attribútumok informális leírásával adhattunk meg.
Adattábla (reláció) R felett: T $\subseteq$ dom(A_1) X ... X dom(A_n), amely eleget tesz az F-beli függőségeknek.

Jelölés: R = (A, F) helyett továbbra is használjuk az egyszerűbb R(A) jelölést, ha a függőségeket nem kívánjuk hangsúlyozni.

39. Példa. A Dolgozó sémához tartozó függőségi halmaz F_D = {f_1, f_2}. Az f_3 és f_4 függőségeket nem szükséges hozzávenni, mert érezhetően következményei f_1 és f_2-nek.

Kérdés, hogy adott függőségekből levezethetők-e újabb függőségek. Erre vonatkozó, könnyen bizonyítható alapszabályok az *Armstrong-axiómák*:

A1. Reflexivitás: Ha Y $\subseteq$ X, akkor X→Y.
Bizonyítás: $t_i(X) = t_j(X) \Rightarrow t_i(Y) = t_j(Y)$ triv.

A2. Bővítés: Ha X→Y, akkor X U Z → Y U Z.
Bizonyítás: $t_i(X \cup Z) = t_j(X \cup Z) \Rightarrow t_i(X) = t_j(X)$ és $t_i(Z) = t_j(Z) \Rightarrow t_i(Y) = t_j(Y)$ és $t_i(Z) = t_j(Z) \Rightarrow t_i(Y \cup Z) = t_j(Y \cup Z)$.

A3. Tranzitivitás: Ha $X \to Y$ és $Y \to Z$, akkor $X \to Z$.
Bizonyítás: $t_i(X) = t_j(X) \Rightarrow t_i(Y) = t_j(Y) \Rightarrow t_i(Z) = t_j(Z)$.

40. Definíció. Az $R=(A, F)$ feletti $f_1,..., f_n$ függőségekből **következik** az f függőség, ha nem lehet olyan T táblát megadni R felett, amelyre $f_1,..., f_n$ teljesül, de f nem.

41. Állítás. Az Armstrong-axiómák segítségével egy adott függőségi halmazból következő bármely függőség formálisan levezethető. (Levezetésen az axiómák véges sokszori alkalmazását értjük a formális logika szabályai szerint.)
Bizonyítás: itt nem tárgyaljuk.

A funkcionális függés definíciója alapján könnyen beláthatók az alábbi szabályok:
Szétvágási szabály: ha $X \to \{B_1,...,B_k\}$, akkor $X \to B_1$, ..., $X \to B_k$ ($B_i \in A$ attribútum, i=1,...,k).
Egyesítési szabály: ha $X \to B_1$, ..., $X \to B_k$, akkor $X \to \{B_1,...,B_k\}$
De vigyázat! A fentiek fordítottja már nem igaz, vagyis ha $\{B_1,...,B_k\} \to X$, ebből nem következik, hogy $B_1 \to X$, ..., $B_k \to X$.

A szétvágási szabály bizonyítása Armstrong-axiómákkal: reflexivitás miatt $\{B_1,...,B_k\} \to B_i$, tranzitivitásból $X \to B_i$.
Az egyesítési szabály bizonyításához belátjuk, hogy $X \to Y$ és $X \to Z$ akkor $X \to (Y \cup Z)$. Ugyanis a bővítés miatt $(X \cup X) \to (Y \cup X)$ és $(X \cup Y) \to (Z \cup Y)$, innen tranzitivitással $X \to (Y \cup Z)$.

Kulcsok meghatározása

Kérdés: ha adott R = (A, F), a függéshalmaz vizsgálatával meg tudjuk-e határozni a kulcsokat?

42. Definíció. Egy X attribútumhalmaz *lezártja* az F függőségi halmaz szerint $X^+ = \{A_i \mid X \to A_i\}$, vagyis az összes X-től függő attribútumokból áll. Pontosabban: X^+ azon A_i attribútumokból áll, amelyekre az $X \to A_i$ függőség F-ből levezethető.

Algoritmus X^+ számítására. Az $X = X^{(0)} \subset X^{(1)} \subset ... \subset X^{(n)} = X^+$ halmazsorozatot képezzük. $X^{(i)}$-ből $X^{(i+1)}$ előállítása: keressünk olyan F-beli $P \to Q$ függőséget, amelyre $P \subseteq X^{(i)}$, de Q már nem része $X^{(i)}$-nek! Ha találunk ilyet, akkor $X^{(i+1)} := X^{(i)} \cup Q$, ha nem, akkor $X^{(i)} = X^+$, vagyis elértük a lezártat. Mivel A véges halmaz, így az eljárás véges sok lépésben véget ér.
Könnyen belátható, hogy a fenti módon generált X^+ halmaz bármely eleme függ X-től. Annak bizonyításától, hogy X^+ az összes X-től függő elemet tartalmazza, itt eltekintünk.

43. Példa. Tekintsük az R=(Z,F) sémát, ahol Z = {A, B, C, D, E}, és F tartalmazza az alábbi függőségeket:

```
{C} → {A}
{B} → {C,D}
{D,E} → {C}
```

Határozzuk meg a $\{B\}^+$ halmazt!

```
X(0) = {B}                          függőségek: {B} → {C,D}
X(1) = {B} U {C,D} = {B,C,D}        függőségek: {B} → {C,D}
```

$X^{(2)} = \{B,C,D\} \cup \{A,C,D\} = \{A,B,C,D\}$ függőségek: $\{C\} \rightarrow \{A\}$, $\{B\} \rightarrow \{C,D\}$, $\{C\} \rightarrow \{A\}$

$X^{(3)} = X^{(2)}$, tehát $\{B\}^+ = \{A,B,C,D\}$

44. Állítás. Egy K attribútumhalmaz akkor és csak akkor szuperkulcs, ha $K^+=A$.
Bizonyítás: belátható, hogy $K \rightarrow A$ akkor és csak akkor teljesül, ha $K^+=A$.

Kulcs meghatározása. Először legyen K=A, ez mindig szuperkulcs. K-ból sorra elhagyunk attribútumokat, és mindig ellenőrizzük $K^+=A$ teljesül-e.

A fenti R=(Z, F) séma esetén jól látható, hogy {B, E} szuperkulcs. Most vizsgáljuk meg, hogy Z-ből B-t illetve E-t elhagyva szuperkulcsot kapunk-e:
$\{A, C, D, E\}^+ = \{A, C, D, E\}$
$\{A, B, C, D\}^+ = \{A, B, C, D\}$
Egyik esetben sem kaptunk szuperkulcsot, amiből az következik, hogy minden kulcsnak tartalmaznia kell B-t és E-t, vagyis az egyetlen kulcs {B,E}.

45. Definíció. Az F *függéshalmaz lezártja* az összes F-ből levezethető függést tartalmazza. Jelölése F^+.

46. Definíció. Az F^+ egy részhalmazát bázisnak nevezzük, ha belőle F valamennyi függése levezethető.

47. Állítás. $F^+ = \{X \rightarrow Y \mid Y \subseteq X^+\}$, vagyis F^+ pontosan azokból az $X \rightarrow Y$ függőségekből áll, amelyekre Y részhalmaza X^+-nak.
Bizonyítás. Belátható, hogy $Y \subseteq X^+$ akkor és csak akkor teljesül, ha $X \rightarrow Y$.

Algoritmus F^+ meghatározására:
1. Vegyük az A attribútumhalmaz összes részhalmazát.
2. Minden X részhalmazhoz állítsuk elő X^+ -t.
3. Valamennyi $Y \subseteq X^+$ -ra az $X \rightarrow Y$ függőséget felvesszük F^+-ba.

5.3. Felbontás (dekompozíció)

A redundáns Dolgozó táblát a 17. ábra szerint bontottuk szét. Most megvizsgáljuk, hogy egy felbontás mikor helyes és mikor nem.

48. Definíció. Legyen R(A) egy relációséma, és $X,Y \subset A$ úgy, hogy $X \cup Y = A$. Ekkor az R(A) séma *felbontása* X, Y szerint az $R_1(X)$ és $R_2(Y)$ sémákat eredményezi. Az R séma feletti T tábla felbontása projekcióval történik: $T_1=\pi_X(T)$ és $T_2=\pi_Y(T)$.

49. Definíció. Egy felbontást *hűséges*nek nevezünk, ha bármely R feletti T tábla esetén $T=T_1*T_2$. Vagyis, a felbontás után adódó táblákból természetes összekapcsolással az eredeti táblát kapjuk vissza.

Könnyen belátható, hogy tetszőleges felbontás esetén $T \subseteq T_1*T_2$ teljesül. A hűségesség tehát azt jelenti, hogy az összekapcsolás nem állít elő fölös sorokat.

Hűséges felbontásra a 17. ábrán láthattunk példát.

A *hűséges* helyett a *veszteségmentes* (lossless) kifejezés is használatos, amely valójában nem sorok elvesztésére, hanem információvesztésre utal.

50. Példa. Nem hűséges felbontást kapunk, ha a Dolgozó táblát a *VezAdószám* mentén bontjuk fel:
Dolg (Név, Adószám, Cím, VezAdószám)
Oszt (Osztálykód, Osztálynév, VezAdószám)

Ugyanis a Dolgozó definiálásakor nem kötöttünk ki VezAdószám → Osztálykód függést, ezzel megengedtük, hogy egy személy több osztálynak is vezetője legyen. Ha például Takács dolgozó az 1-es osztályon dolgozik, de ennek vezetője azonos az 5-ös osztály vezetőjével, akkor a Dolg*Oszt táblában Takács kétszer fog szerepelni: egyszer az 1-es, egyszer az 5-ös osztály dolgozójaként (18. ábra).

A Dolgozó tábla:

Név	*Adószám*	*Cím*	*Osztálykód*	*Osztálynév*	*VezAdószám*
Takács	5555	Győr, Pap u. 7.	1	Munkaügyi	3333
Rácz	9999	Vác, Domb u. 1.	5	Pénzügyi	3333

A Dolg és Oszt táblák:

Név	*Adószám*	*Cím*	*VezAdószám*
Takács	5555	Győr, Pap u. 7.	3333
Rácz	9999	Vác, Domb u. 1.	3333

Osztálykód	*Osztálynév*	*VezAdószám*
1	Munkaügyi	3333
5	Pénzügyi	3333

Az egyesített Dolg*Oszt tábla:

Név	*Adószám*	*Cím*	*Osztálykód*	*Osztálynév*	*VezAdószám*
Takács	5555	Győr, Pap u. 7.	1	Munkaügyi	3333
Takács	5555	Győr, Pap u. 7.	5	Pénzügyi	3333
Rácz	9999	Vác, Domb u. 1.	1	Munkaügyi	3333
Rácz	9999	Vác, Domb u. 1.	5	Pénzügyi	3333

18. ábra. Nem hűséges felbontás következménye

A gyakorlatban rendszerint az alábbi tétel alapján végzünk dekompozíciót:

51. Heath tétele. Ha az R(A) sémánál A = B U C U D, ahol B, C és D diszjunkt attribútum-részhalmazok és C → D, akkor az R_1(B U C), R_2(C U D) felbontás hűséges.

Bizonyítás: Legyen T egy tetszőleges R feletti tábla, T_1 és T_2 a megfelelő szétbontott táblák. $T \subseteq T_1 * T_2$ nyilvánvaló, ezért csak azt kell megmutatni, hogy $T_1 * T_2 \subseteq T$. Legyen $t \in T_1 * T_2$. Ekkor kell hogy legyen olyan $t_1 \in T_1$ és $t_2 \in T_2$, amelyek egyesítéseként t előállt, vagyis $t_1(C) = t_2(C)$. Kell, hogy legyenek továbbá olyan u_1, u_2 sorok T-ben, amelyekből projekcióval t_1, t_2 származtatható, vagyis $u_1(B \cup C) = t_1$ és $u_2(C \cup D) = t_2$. Mivel $u_1(C) = u_2(C)$, így a C → D függőség miatt $u_1(D) = u_2(D)$. Tehát a $u_1 = t$, vagyis t szerepel T-ben.

52. Példa. A Dolgozó (név, adószám, cím, osztálykód, osztálynév, vezAdószám) tábla esetén az {osztálykód} → {osztálynév, vezAdószám} függőség teljesül. Ezért ha
B = {név, adószám, cím}, C = {osztálykód}, D = {osztálynév, vezAdószám}, akkor a
Dolg (név, adószám, cím, osztálykód)
Oszt (osztálykód, osztálynév, vezAdószám)
felbontás Heath tétele alapján hűséges lesz.

53. Példa. Tekintsük az R(e, f, g, h) relációsémát, ahol {e, f} → g. Ekkor a B, C, D attribútum részhalmazokat válasszuk úgy, hogy B = h, C = {e, f}, D = g. Mivel C → D, így Heath tétele alapján az R_1(e, f, h), R_2(e, f, g) felbontás hűséges.

A függőségeket is figyelembe véve, egy R=(A,F) relációséma felbontása X, Y szerint R_1=(X,F_1) és R_2=(Y,F_2), ahol F_1 úgy választandó meg, hogy F_1^+ az F^+ azon részhalmazával legyen egyenlő, amely csak X-beli attribútumokat tartalmaz, F_2 hasonlóan.

Egy R=(A, F) séma R_1=(X, F_1), R_2=(Y, F_2) felbontását *függőségőrző*nek nevezzük, ha F_1 U F_2 az eredeti F bázisát adják. Egy hűséges dekompozíció nem feltétlenül függőségőrző. Ha például a vállalat azzal a szokatlan feltétellel élne, hogy minden dolgozó a hozzá legközelebb lakó osztályvezetőhöz kell hogy tartozzon, akkor a Dolgozó táblában Cím → VezAdószám függés lép fel. A dekompozíció során ez a függőség elvész, de ez nem változtat azon a tényen, hogy - a hűségesség miatt - a Dolg és Oszt táblákból természetes join művelettel mindig visszaállítható az eredeti Dolgozó tábla.

Felbontás kulcs mentén

Legyenek K, A, B attribútumhalmazok. Ha K (szuper)kulcs, akkor az R(K, A, B) séma felbontása az R_1(K, A) és R_2(K, B) sémákra hűséges.

Bizonyítás: K → B miatt Heath tételéből következik.

54. Példa: Dolgozó (azonosító, név, cím, osztálykód) lehetséges felbontása:
Dolg1 (azonosító, név, cím) és Dolg2 (azonosító, osztálykód), vagy
Dolg1 (azonosító, név, cím) és Dolg2 (azonosító, név, osztálykód).

Megjegyzés: kulcs mentén mindig lehet felbontani, de ennek általában nincs értelme, mert nem szüntetünk meg vele redundanciát.

Egyesítés kulcs mentén

Ha két séma kulcsa megegyezik, akkor a sémák egyesíthetők, vagyis az R_1(K, A) és R_2(K, B) sémák helyettesíthetők az R(K, A, B) sémával.

55. Példa: Egy vállalatnál a Bérosztály a fizetéseket a
DolgBér (azonosító, név, cím, fizetés) táblában, az Ellátási osztály a munkaruhákat a
DolgRuha (azonosító, név, kiadásdátum) táblában tartja nyilván.
Az egyesített vállalati adatbázis: Dolgozó (azonosító, név, cím, fizetés, kiadásdátum)
Az egyesítés kétoldali külső összekapcsolással célszerű.

Megjegyzés: ha az azonosító DolgBér-ben adószám, DolgRuha-ban személyi szám, akkor az egyesítés nem lehetséges.

5.4. Normálformák

1. normálforma (1NF)

56. Definíció. Egy relációséma 1NF-ben van, ha az attribútumok értéktartománya csak egyszerű (atomi) adatokból áll (nem tartalmaz például listát vagy struktúrát).

Mivel az 1NF feltétel teljesülését már a relációs modell definíciójánál kikötöttük, ezért az 1NF-re hozást lényegében az E-K modellről relációs modellre történő átalakításnál elvégeztük (összetett és többértékű attribútumok leképezése).

2. normálforma (2NF)

57. Definíció. Legyen R(A) relációséma, $X,Y \subseteq A$, és $X \rightarrow Y$. Azt mondjuk, hogy X-től *teljesen függ* Y, ha X-ből bármely attribútumot elhagyva a függőség már nem teljesül, vagyis bármely $X_1 \subset X$ esetén $X_1 \rightarrow Y$ már nem igaz.

Megjegyzés: Ha K kulcs, akkor A teljesen függ K-tól.

58. Definíció. Egy attribútumot *elsődleges attribútum*nak nevezünk, ha szerepel a relációséma valamely kulcsában, ellenkező esetben *másodlagos attribútum*. Vagyis, ha a séma kulcsai $K_1,...,K_r$, akkor $K = K_1 \cup ... \cup K_r$ az elsődleges attribútumok halmaza, A–K a másodlagos attribútumok halmaza.

59. Definíció: Egy relációséma 2NF-ben van, ha minden másodlagos attribútum teljesen függ bármely kulcstól.

Következmények:

- Ha minden kulcs egy attribútumból áll, akkor a séma 2NF-ben van. Ilyen például a Dolgozó (név, <u>adószám</u>, cím, osztálykód, osztálynév, vezAdószám) tábla.
- Ha a sémában nincs másodlagos attribútum, akkor 2NF-ben van. Ilyen például a Fuvar (gkvez, rendszám, indul, érkezik) tábla, mivel a következő kulcsok vannak: {gkvez, indul}, {gkvez, érkezik}, {rendszám, indul}, {rendszám, érkezik}.

A séma akkor nincs 2NF-ben, ha egy kulcs részhalmazától függ (egy vagy több) másodlagos attribútum.

> Ha a séma nincs 2NF-ben, akkor a táblában redundancia léphet fel. Tegyük fel ugyanis, hogy valamely K kulcs L részhalmazától függ a másodlagos attribútumok egy B halmaza ($L \rightarrow B$). Ekkor a táblában több olyan sor lehet, amelyek L-en megegyeznek, így ezek szükségképpen B-n is megegyeznek, ami a B-értékek redundáns tárolását eredményezi (lásd az alábbi példát).

2NF-re hozás: a sémát felbontjuk Heath tétele szerint, a normálformát sértő függőség mentén.

> Ha valamely K kulcsra $L \subset K$ és $L \rightarrow B$ (itt B legyen az összes L-től függő attribútum halmaza), akkor a sémát felbontjuk az $L \rightarrow B$ függőség szerint. Legyen $C = A - (L \cup B)$, ekkor az R(A) sémát az $R_1(C \cup L)$ és $R_2(L \cup B)$ sémákkal helyettesítjük. Heath tétele alapján a felbontás hűséges.

60. Példa. Tegyük fel, hogy egy vállalat dolgozói különféle projekteken dolgoznak meghatározott heti óraszámban. Ezt a

DolgProj (adószám, név, projektkód, óra, projektnév, projekthely)

sémával tartjuk nyilván, a megfelelő tábla a 19. ábrán látható.

Adószám	*Név*	*Projektkód*	*Óra*	*Projektnév*	*Projekthely*
1111	Kovács	P2	4	Adatmodell	Veszprém
2222	Tóth	P1	6	Hardware	Budapest
4444	Kiss	P1	5	Hardware	Budapest
1111	Kovács	P1	2	Hardware	Budapest
1111	Kovács	P5	8	Teszt	Szeged
8888	Török	P2	12	Adatmodell	Veszprém
5555	Takács	P5	3	Teszt	Szeged
6666	Fekete	P5	4	Teszt	Szeged
8888	Török	P3	4	Software	Veszprém
7777	Nagy	P3	14	Software	Veszprém

19. ábra. A DolgProj séma feletti tábla

Függőségek:
adószám $\rightarrow$ név
projektkód $\rightarrow$ {projektnév, projekthely}
{adószám, projektkód} $\rightarrow$ óra

A sémában {adószám, projektkód} kulcs, mivel ettől minden attribútum függ, ugyanakkor akár *adószám*ot, akár *projektkód*ot elhagyva ez már nem teljesül.

A séma nincs 2NF-ben, mert *név* csak *adószám*tól függ, vagyis a kulcs részhalmazától függ. Felbontás az *adószám* $\rightarrow$ *név* függés mentén:

Dolg (<u>adószám</u>, név)
Dproj (<u>*adószám*, projektkód</u>, óra, projektnév, projekthely)

A Dproj séma a *projektkód* $\rightarrow$ *{projektnév, projekthely}* függőség miatt még mindig nincs 2NF-ben. Felbontás ezen függőség mentén:

Dolg (<u>adószám</u>, név)
Proj (<u>projektkód</u>, projektnév, projekthely)
DP (<u>*adószám, projektkód*</u>, óra)

Itt már mindhárom séma 2NF-ben van (20. ábra).

Dolg tábla:

Adószám	*Név*
1111	Kovács
2222	Tóth
4444	Kiss
8888	Török
5555	Takács
6666	Fekete
7777	Nagy

Proj tábla:

Projektkód	*Projektnév*	*Projekthely*
P1	Hardware	Budapest
P2	Adatmodell	Veszprém
P3	Software	Veszprém
P5	Teszt	Szeged

DP tábla:

Adószám	*Projektkód*	*Óra*
1111	P2	4
2222	P1	6
4444	P1	5
1111	P1	2
1111	P5	8
8888	P2	12
5555	P5	3
6666	P5	4
8888	P3	4
7777	P3	14

20. ábra. A DolgProj séma normalizálása után keletkező táblák

3. normálforma (3NF)

61. Definíció. Legyen X, Z $\subseteq$ A, és X$\rightarrow$Z. Azt mondjuk, hogy X-től *tranzitívan függ* Z, ha van olyan Y $\subseteq$ A, amelyre X$\rightarrow$Y és Y$\rightarrow$Z, de X nem függ Y-tól, és az Y$\rightarrow$Z függés teljesen nemtriviális. Ellenkező esetben Z *közvetlenül függ* X-től.

> *Megjegyzés:* Az "X nem függ Y-tól" és az "Y$\rightarrow$Z függés teljesen nemtriviális" kiegészítő feltételek nem csak a triviális esetek kiszűréséhez kellenek, hanem a későbbi állítások szempontjából is lényegesek.

62. Definíció. Egy relációséma 3NF-ben van, ha minden másodlagos attribútuma közvetlenül függ bármely kulcstól.

Következmény: Ha a sémában nincs másodlagos attribútum, akkor 3NF-ben van.

A séma nincs 3NF-ben, ha egy vagy több másodlagos attribútum tranzitívan függ valamely kulcstól.

> Ha a séma nincs 3NF-ben, akkor a táblában redundancia léphet fel. Tegyük fel ugyanis, hogy valamely K kulcstól tranzitívan függ a másodlagos attribútumok egy B halmaza, vagyis valamely Y attribútumhalmazra K$\rightarrow$Y és Y$\rightarrow$B, de K nem függ Y-tól és Y $\cap$ B üres. Mivel Y-tól nem függ K, így Y nem szuperkulcs, vagyis a táblában több olyan sor lehet, amelyek Y-on megegyeznek. Ezek a sorok az Y$\rightarrow$B függőség miatt szükségképpen B-n is megegyeznek, ami a B-értékek redundáns tárolását eredményezi (lásd az alábbi példát).

3NF-re hozás. Ha másodlagos attribútumok egy B halmazára és valamely K kulcsra $K \to Y \to B$ tranzitív függés fennáll, akkor a sémát felbontjuk Heath tétele szerint az $Y \to B$ függés mentén.

> B legyen az összes Y-tól függő attribútum halmaza. Legyen $C = A - (Y \cup B)$, ekkor az R(A) sémát az $R_1(C \cup Y)$ és $R_2(Y \cup B)$ sémákkal helyettesítjük. Heath tétele alapján a felbontás hűséges.

63. Példa. Tekintsük a vállalat dolgozóit és az osztályokat nyilvántartó sémát:

Dolgozó (név, adószám, cím, osztálykód, osztálynév, vezAdószám)

Függőségek:
adószám → {név, cím, osztálykód}
osztálykód → {osztálynév, vezAdószám}

A séma 2NF-ben van, mert egyetlen kulcs az *adószám*, amely egyelemű. Ugyanakkor nincs 3NF-ben, mert tranzitív függés van:
adószám → osztálykód → {osztálynév, vezAdószám}.

3NF-re hozás: dekompozíció a függőség szerint:
Dolg (<u>adószám</u>, név, cím, *osztálykód*)
Oszt (<u>osztálykód</u>, osztálynév, vezAdószám)

64. Állítás. Ha egy relációséma 3NF-ben van, akkor 2NF-ben is van.

> *Bizonyítás (indirekt)*. Tegyük fel, hogy az R=(A,F) séma 3NF-ben van, és még sincs 2NF-ben. Ez utóbbi azt jelenti, hogy valamely A_i másodlagos attribútum nem teljesen függ valamely K kulcstól, vagyis van olyan $L \subset K$, amelyre $L \to A_i$. Ekkor viszont K-tól tranzitíven függ A_i, ugyanis $K \to L \to A_i$, de L-től nem függ K (mivel K kulcs, tehát minimális), valamint A_i nem eleme L-nek (mivel másodlagos attribútum).

Boyce-Codd normálforma (BCNF)

65. Definíció. Egy relációséma BCNF-ben van, ha bármely nemtriviális $L \to B$ függés esetén L szuperkulcs.

A séma nincs BCNF-ben, ha van benne olyan nemtriviális függés, amelynek bal oldalán nem szuperkulcs áll.

> Ha a séma nincs BCNF-ben, akkor a táblában redundancia léphet fel. Tegyük fel ugyanis, hogy $L \to B$ és L nem szuperkulcs Ezért a táblában több olyan sor lehet, amelyek L-en megegyeznek, és a függőség miatt szükségképpen B-n is megegyeznek, ami a B-értékek redundáns tárolását eredményezi.

BCNF-re hozás: a sémát felbontjuk Heath tétele szerint, a normálformát sértő függőség mentén.

> Ha $L \to B$ teljesen nemtriviális függés és L nem szuperkulcs, akkor a sémát felbontjuk az $L \to B$ függőség szerint (itt B legyen az összes L-től függő attribútum halmaza). Legyen $C = A - (L \cup B)$, ekkor az R(A) sémát az $R_1(C \cup L)$ és $R_2(L \cup B)$ sémákkal helyettesítjük. Heath tétele alapján a felbontás hűséges.

66. Állítás. Ha egy relációséma BCNF-ben van, akkor 3NF-ben is van.

> *Bizonyítás (indirekt):* Tegyük fel, hogy a séma BCNF-ben van, de nincs 3NF-ben, vagyis van olyan $K \to L \to B$ tranzitív függés, ahol K kulcs. A tranzitív függés definíciójából adódóan

ekkor L-től nem függ K (ezért L nem szuperkulcs), továbbá L→B nemtriviális, ami ellentmond a BCNF feltételezésnek.

A gyakorlatban ha egy séma 3NF-ben van, akkor általában BCNF-ben is van. Adódnak azonban kivételek, ilyen az alábbi példa.

67. Példa. Tegyük fel, hogy a Fuvar sémában a gépkocsivezetők adószámát és TAJ-számát is nyilvántartják: Fuvar (vezAdószám, vezTAJszám, rendszám, indul, érkezik)

Ekkor a kulcsok:
{vezAdószám, indul}, {vezTAJszám, indul},
{vezAdószám, érkezik}, {vezTAJszám, érkezik},
{rendszám, indul}, {rendszám, érkezik}

Nincs másodlagos attribútum, ezért a séma 3NF-ben van.

További függések:
{vezAdószám} → {vezTAJszám}
{vezTAJszám} → {vezAdószám}
Ezek a függések sértik a BCNF-et.

BCNF-re hozás: felbontás a {vezAdószám} → {vezTAJszám} függés mentén:
Gkvez (<u>vezAdószám</u>, vezTAJszám)
Fuvar (*vezAdószám*, <u>rendszám, indul</u>, érkezik)

4. normálforma (4NF)

68. Példa. Tekintsük a Rendelhet (nagyker, kisker, áru) sémát, ahol a tábla egy sora adott kiskereskedőnek adott nagykereskedőtől beszerezhető árufajtáját jelenti. Ha egy kiskereskedő adott nagykereskedővel kapcsolatban áll, akkor a nagykereskedő összes áruját nyilvántartásba veszi (21. ábra). Ez azt jelenti, hogy ha valamely (N_i, K_j) és (N_i, A_k) párok szerepelnek a táblában, akkor az (N_i, K_j, A_k) hármas is kell hogy szerepeljen. Kulcs: az összes attribútum. Mivel nincs funkcionális függés, ezért a séma BCNF-ben van, ugyanakkor a tábla erőteljesen redundáns, amit egy Szállít és egy Kínál táblára való felbontással szüntethetünk meg. Ennek elméleti alapjait tárgyaljuk a továbbiakban.

Rendelhet tábla:

Nagyker	*Kisker*	*Áru*
N1	K1	A1
N1	K1	A2
N1	K1	A3
N1	K2	A1
N1	K2	A2
N1	K2	A3
N2	K2	A1
N2	K2	A4
N2	K3	A1
N2	K3	A4

Szállít tábla:

Nagyker	*Kisker*
N1	K1
N1	K2
N2	K2
N2	K3

Kínál tábla:

Nagyker	*Áru*
N1	A1
N1	A2
N1	A3
N2	A1
N2	A4

21. ábra. A Rendelhet tábla és felbontása

69. Definíció. Legyen K, L $\subseteq$ A, és legyen M = A - (K U L). Azt mondjuk, hogy K-tól *többértékűen függ* L, jelölésben K$\rightarrow\rightarrow$L, ha bármely R feletti T táblában ha két sor megegyezik K-n, akkor a két sor kombinációja is szerepel T-ben. Ez pontosabban azt jelenti, hogy ha a t_i, t_j sorokra $t_i(K) = t_j(K)$, akkor van olyan t sor, amelyre az alábbiak teljesülnek:

- $t(K) = t_i(K) = t_j(K)$
- $t(L) = t_i(L)$
- $t(M) = t_j(M)$

Jól látható, hogy a fenti példában *nagyker*$\rightarrow\rightarrow$*kisker* többértékű függés van.

70. Definíció. A K$\rightarrow\rightarrow$L függés *nemtriviális*, ha K $\cap$ L = 0 és K U L $\neq$ A. (Ugyanis K U L = A esetén M üres, és $t = t_i$ választásával a feltétel mindig teljesül.)

Állítás. Ha K$\rightarrow$L, akkor K$\rightarrow\rightarrow$L.
Bizonyítás: $t = t_j$ választással nyilvánvaló.

71. Állítás. Ha K$\rightarrow\rightarrow$L, akkor K$\rightarrow\rightarrow$M.
Bizonyítás: a szimmetriából nyilvánvaló.

72. Fagin tétele. Az R(A) relációsémánál legyen A = B U C U D, ahol B, C és D diszjunktak. R felbontása az R_1(B U C), R_2(C U D) sémákra *akkor és csak akkor hűséges*, ha C $\rightarrow\rightarrow$ D fennáll.

Bizonyítás (direkt):
a) Ha a felbontás hűséges, azaz $T=T_1*T_2$, akkor a többértékű függés a természetes join művelet definíciójából adódik: $t_1(B\ U\ C) \in T_1$, hasonlóan $t_2(C\ U\ D) \in T_2$, ezért szükségképpen $t \in T$.
b) Ha C $\rightarrow\rightarrow$ D, akkor a hűségességet kell bizonyítanunk. Legyen $t_1 \in T_1$ és $t_2 \in T_2$, amelyekre $t_1(C) = t_2(C)$. Ekkor a t_1 és t_2 egyesítésével előálló rekord a függőség miatt szerepel T-ben, vagyis $T_1*T_2 \subseteq T$. Ugyanakkor $T \subseteq T_1*T_2$ nyilvánvaló, így $T = T_1*T_2$.

73. Definíció. Egy relációséma 4NF-ben van, ha minden nemtriviális K$\rightarrow\rightarrow$L függés esetén K szuperkulcs.

74. Állítás. Ha egy relációséma 4NF-ben van, akkor BCNF-ben is van.

Bizonyítás (direkt). Legyen K$\rightarrow$L nemtriviális függés, belátjuk, hogy K szuperkulcs. Két eset lehetséges:
- Ha K U L = A, akkor K$\rightarrow$L miatt K szuperkucs.
- Ha K U L $\subset$ A, akkor legyen L_1 = L–K, ekkor K$\rightarrow L_1$, ezért K$\rightarrow\rightarrow L_1$ nemtriviális, amiből a 4NF tulajdonság miatt következik, hogy K szuperkulcs.

A séma nincs 4NF-ben, ha van benne olyan nemtriviális többértékű függés, amelynek bal oldalán nem szuperkulcs áll.

Ha egy séma nincs 4NF-ben, akkor a tábla redundanciát tartalmazhat. Ha ugyanis K$\rightarrow\rightarrow$L, és K nem szuperkulcs, akkor a táblában több olyan sor lehet, amely K-n megegyezik, és ezekben a sorokban az L és M-értékek redundánsan szerepelnek.

4NF-re hozás: a sémát felbontjuk Fagin tétele szerint, a normálformát sértő függőség mentén.

*H*a K$\rightarrow\rightarrow$L nemtriviális függés, és K nem szuperkulcs, akkor az R(A) sémát az R_1(K U L) és R_2(K U M) sémákkal helyettesítjük. Ez hűséges felbontás Fagin tétele szerint.

A Rendelhet séma az alábbi felbontással hozható 4NF-re (21. ábra):
Szállít (nagyker, kisker)
Kínál (nagyker, áru)

Normálformák összefoglalása

Az 1NF-re hozás a relációs modellnél kötelező. A további normálformák egyre szigorúbb feltételeket írnak elő (2NF <= 3NF <= BCNF <= 4NF), amelyek kiküszöbölik a redundanciát és az aktualizálási anomáliákat. Az ezek szerinti normalizálás célszerű, de nem kötelező. A gyakorlatban azt kell mérlegelni, hogy a redundancia és az anomáliák mennyire jelentenek súlyos veszélyt, indokolt-e azok megszüntetésével a táblák számát növelni (dekompozíció). Erre mutat rá az alábbi példa.

75. Példa. Tegyük fel, hogy egy biztosító társaság az ügyfelei lakcíme mellett azt is nyilvántartja, hogy hány lakásos házban laknak:
Ügyfél (adószám, név, szüldátum, lakcím, lakásszám)

A séma nincs 3NF-ben a *lakcím* $\rightarrow$ *lakásszám* függés miatt. Ez azonban csak akkor okoz redundanciát, ha a biztosítónak több ügyfele lakik ugyanabban a házban. Két eset lehetséges:

a) Ha ritkán fordul elő, hogy egy házban több ügyfél legyen, és a lakásszám nyilvántartásának csak statisztikai jelentősége van, akkor nem érdemes felbontani a táblát.

b) Ha viszont a biztosító társaság ellenőrizni kívánja, hogy az egy házban lakók azonos lakásszámot adnak-e meg (mert például ettől is függhet a biztosítás összege), akkor a felbontás indokolt.

Adatbázis tervezés összefoglalása

Az adatbázis tervezés folyamata három fő lépésből áll:
1. Egyed-kapcsolat modell felírása.
2. Relációs adatbázis séma felírása. Az 1NF-re hozás már itt elvégzendő.
3. Relációsémák normalizálása.
4. Szükség esetén az egyed-kapcsolat modell módosítása a normalizálás szerint.

6. Az SQL nyelv

SQL = Structured Query Language (= strukturált lekérdező nyelv). A relációs adatbázis-kezelés szabványos nyelve. Nem algoritmikus nyelv, de algoritmikus nyelvekbe beépíthető (beágyazott SQL). Története:

1976: SEQUEL (= Structured English QUEry Language) az SQL eredeti változata, IBM-nél fejlesztették ki.

1981: Oracle 2 (SQL alapú RDBMS, nagygépre).

1983: IBM: DB2 (SQL alapú RDBMS, nagygépre). A világ legnagyobb adatbázisait ma is jórészt DB2-ben kezelik.

SQL szabvány (1986), az ANSI (= American National Standards Institute) definiálta. Változatai: SQL-86, SQL-89.

SQL2 szabvány (1992), más néven SQL-92.

SQL3 szabvány (1999), más néven SQL:1999: rekurzió, triggerek, objektum-relációs modell.

SQL:2003 szabvány: többek között XML támogatással bővült.

2006, 2008: további bővítések.

A jelenlegi SQL-implementációk általában az SQL2-nél jóval többet tudnak, ugyanakkor előfordul, hogy az SQL2 bizonyos részleteit nem tartalmazzák, illetve a szabványtól eltérő formában tartalmazzák (Oracle, MySQL, PostgreSQL).

Jelen anyagban az SQL2 szabványt vesszük alapul, de az utasításoknak csak a fontosabb lehetőségeit tárgyaljuk. A konkrét rendszerek utasításai gyakran eltérnek az SQL2 szabványtól, ezért programozásnál mindig az adott rendszer kézikönyvei a mérvadók.

6.1. Általános jellemzés

Az SQL utasításait két fő csoportba szokták sorolni:

- DDL (= Data Definition Language): adatstruktúra definiáló utasítások.
- DML (= Data Manipulation Language): adatokon műveletet végző utasítások.

Jelen anyagban - az RDBMS fő feladatai alapján - az alábbi csoportokban tárgyaljuk az SQL utasításokat:

- adatbázisséma definiálása (DDL),
- adatok aktualizálása (DML),
- lekérdezési lehetőségek (DML).

Szintaxis

Kisbetű és nagybetű a nyelv alapszavaiban egyenértékű.

Utasítások sorfolytonosan írhatók, lezárás pontosvesszővel.

Változó nincs, csak tábla- és oszlopnevekre lehet hivatkozni. Kifejezésben hivatkozás egy tábla adott oszlopára: tábla.oszlop (ha a tábla egyértelmű, akkor elhagyható).

Alias név: név AS másodnév (egyes implementációkban AS elhagyható).

Szövegkonstans: 'szöveg'

Dátum: DATE '1968-05-12'. Egyes rendszerek az SQL szabványtól eltérő konvenciót alkalmaznak, például 13-NOV-94 (Oracle)..

Idő: TIME '15:31:02.5' (óra, perc, másodperc).

Stringek konkatenációja: + vagy || .

Relációjelek: =, <=, >=, !=, <>

Logikai műveletek: AND, OR, NOT. Az SQL rendszerek "háromértékű logikát" használnak, vagyis a TRUE és FALSE mellett a NULL (definiálatlan) érték is felléphet. Ha egy kifejezés valamelyik eleme NULL, akkor a kifejezés értéke is NULL lesz.

> Az SQL-szabvány szerint egy logikai kifejezés értéke ISMERETLEN (UNKNOWN), ha benne NULL érték szerepel.

Az utasítások szintaxisának leírásánál az elhagyható részleteket szögletes zárójellel jelöljük.

Speciális logikai kifejezések

x IS NULL: igaz, ha az x mező értéke NULL. Ez nem egyenértékű az "x = NULL" kifejezéssel, ugyanis ennek értéke definiálatlan, mivel definiálatlan komponenst tartalmaz. A gyakorlatban tehát az „x IS NULL” forma használandó.

x BETWEEN a AND b: igaz, ha $a \leq x \leq b$.

x IN halmaz: igaz, ha x megegyezik a megadott halmaz egy elemével. A halmazt explicit módon vagy lekérdezéssel lehet megadni.
Példa: város IN ('Szeged','Szolnok','Pécs')

x relációjel ALL halmaz: igaz, ha x a halmaz minden elemével a megadott relációban van.
Példa: fizetés != ALL (81000, 136000, 118000)

x relációjel ANY halmaz: igaz, ha a halmaznak van olyan eleme, amellyel x a megadott relációban van.
Példa: fizetés < ANY (81000, 136000, 118000)

EXISTS halmaz: igaz, ha a halmaz nem üres. Például egy "EXISTS lekérdezés" kifejezés értéke igaz, ha a lekérdezés legalább egy elemet ad vissza.

x LIKE minta: igaz, ha az x karaktersorozat megfelel a megadott mintának. Ha a mintában "%" illetve "_" jel szerepel, az tetszőleges karaktersorozatot illetve tetszőleges karaktert jelent. Példa: lakcím LIKE '%Vár u.%' igaz minden olyan lakcímre, amelyben szerepel a "Vár u." részlet.

A fentiekben általában a NOT is használható, például x IS NOT NULL, x NOT IN halmaz, stb.

6.2. Relációsémák definiálása (DDL)

Relációséma létrehozására a CREATE TABLE utasítás szolgál, amely egyben egy üres táblát is létrehoz a sémához. Az attribútumok definiálása mellett a kulcsok és külső kulcsok megadására is lehetőséget nyújt:

```
CREATE TABLE táblanév
  ( oszlopnév  adattípus  [feltétel],
       ...      ...,
    oszlopnév  adattípus  [feltétel]
    [, táblaFeltételek]
  );
```

Az adattípusok (rendszerenként eltérők lehetnek):

CHAR(n)	n hosszúságú karaktersorozat
VARCHAR(n)	legfeljebb n hosszúságú karaktersorozat
INTEGER	egész szám (röviden INT)
REAL	valós (lebegőpontos) szám, másnéven FLOAT
DECIMAL(n[,d])	n jegyű decimális szám, ebből d tizedesjegy
DATE	dátum (év, hó, nap)
TIME	idő (óra, perc, másodperc)

Az adattípushoz "DEFAULT érték" megadásával alapértelmezett érték definiálható. Ha ilyet nem adunk meg, az alapértelmezett érték NULL.

Feltételek (egy adott oszlopra vonatkoznak):
PRIMARY KEY: elsődleges kulcs (csak egy lehet)
UNIQUE: kulcs (több is lehet)
REFERENCES tábla(oszlop) [ON-feltételek]: külső kulcs

Táblafeltételek (az egész táblára vonatkoznak):
PRIMARY KEY (oszloplista): elsődleges kulcs
UNIQUE (oszloplista): kulcs
FOREIGN KEY (oszloplista) REFERENCES tábla(oszloplista) [ON-feltételek]: külső kulcs

Ha a (külső) kulcs több oszlopból áll, akkor csak táblafeltétel formájában adható meg.

> A PRIMARY KEY (elsődleges kulcs) és UNIQUE (kulcs) közötti különbségek:
> - Egy sémában csak egy elsődleges kulcs, de tetszőleges számú további kulcs lehet.
> - Külső kulcs általában a másik tábla elsődleges kulcsára hivatkozik.
> - Egyes DBMS-ek az elsődleges kulcshoz automatikusan indexet hoznak létre.
>
> A CREATE TABLE utasítással tulajdonképpen egy R = (A, F) relációsémát adunk meg, ahol *F* megadására szolgálnak a kulcsfeltételek. Ha a relációséma BCNF-ben van, akkor ezzel az összes függés megadható, hiszen ekkor csak szuperkulcstól lehet nemtriviális függés.

76. Példa. Hozzuk létre az
Osztály (<u>osztálykód</u>, osztálynév, vezAdószám)
Dolgozó (<u>adószám</u>, név, lakcím, *osztálykód*)
relációsémákat SQL-ben:

```
CREATE TABLE Osztály
   ( osztálykód   CHAR(3)  PRIMARY KEY,
     osztálynév   CHAR(20),
     vezAdószám   DECIMAL(10)
   );
CREATE TABLE Dolgozó
   ( adószám    DECIMAL(10)  PRIMARY KEY,
     név        CHAR(30),
     lakcím     CHAR(40)  DEFAULT 'ismeretlen',
     osztálykód CHAR(3)  REFERENCES Osztály(osztálykód)
   );
```

A Dolgozó sémát így is lehetne definiálni:

```
CREATE TABLE Dolgozó
   ( adószám    DECIMAL(10),
     név        CHAR(30),
     lakcím     CHAR(40),
     osztálykód CHAR(3),
     PRIMARY KEY (adószám),
     FOREIGN KEY (osztálykód) REFERENCES Osztály(osztálykód)
   );
```

77. Példa. A DolgProj (*adószám*, projektkód, óraszám) sémában összetett kulcs van, amelynek definiálása csak tábla-feltételként lehetséges:

```
CREATE TABLE DolgProj
   ( adószám    DECIMAL(10) REFERENCES Dolgozó(adószám),
     projektkód CHAR(5),
     óraszám    DECIMAL(2),
     PRIMARY KEY (adószám, projektkód)
   );
```

A tábla módosításakor a definiált kulcsfeltételek automatikusan ellenőrzésre kerülnek. PRIMARY KEY és UNIQUE esetén ez azt jelenti, hogy a rendszer nem enged olyan módosítást illetve új sor felvételét, amely egy már meglévő kulccsal ütközne.

REFERENCES (külső kulcs hivatkozás) esetén ON-feltételek megadásával szabályozhatjuk a rendszer viselkedését (jelölje T_1 a hivatkozó és T_2 a hivatkozott táblát):

- Alapértelmezés (ha nincs ON-feltétel): T_1-ben nem megengedett olyan beszúrás és módosítás, amely T_2-ben nem létező kulcs értékre hivatkozna, továbbá T_2-ben nem megengedett olyan kulcs módosítása vagy sor törlése, amelyre T_1 hivatkozik.
- ON UPDATE CASCADE: ha T_2 egy sorában változik a kulcs értéke, akkor a rá való T_1-beli hivatkozások is megfelelően módosulnak (módosítás továbbgyűrűzése).
- ON DELETE CASCADE: Ha T_2-ben törlünk egy sort, akkor T_1-ben is törlődnek a rá hivatkozó sorok (törlés továbbgyűrűzése).
- ON UPDATE SET NULL: ha T_2 egy sorában változik a kulcs értéke, akkor T_1-ben a rá való külső kulcs hivatkozások értéke NULL lesz.
- ON DELETE SET NULL: ha T_2-ben törlünk egy sort, akkor T_1-ben a rá való külső kulcs hivatkozások értéke NULL lesz.

A kulcsfeltételek ellenőrzése csak indexekkel oldható meg hatékonyan.

78. Példa.

```
CREATE TABLE Dolgozó
   ( adószám    DECIMAL(10)  PRIMARY KEY,
     név        CHAR(30),
     lakcím     CHAR(40)  DEFAULT 'ismeretlen',
     osztálykód CHAR(3)  REFERENCES Osztály(osztálykód)
                         ON UPDATE CASCADE
                         ON DELETE SET NULL
   );
```

Relációséma törlése:

DROP TABLE táblanév;

Hatására a séma és a hozzá tartozó adattábla törlődik.

Relációséma módosítása:

ALTER TABLE táblanév
[ADD (újelem, ..., újelem)]
[MODIFY (módosítás, ..., módosítás)]
[DROP (oszlop, ..., oszlop)];

újelem: egy "oszlopnév adattípus [feltétel]", vagy egy "táblafeltétel", mint a CREATE TABLE utasításban.

módosítás: "oszlopnév adattípus [feltétel]".

Az ALTER TABLE utasítás szintaxisa és szemantikája rendszerenként eltérő, például oszlopok törlését nem minden rendszer engedi meg.

Példák:

```
ALTER TABLE Dolgozó ADD (szüldátum DATE);
ALTER TABLE Dolgozó MODIFY (lakcím VARCHAR(60));
ALTER TABLE Osztály
  MODIFY (vezAdószám REFERENCES Dolgozó(adószám));
```

6.3. Indexek létrehozása

Az indexek kezelése nem része az SQL2 szabványnak, de valamilyen formában minden RDBMS támogatja. Index létrehozása általában a

CREATE [UNIQUE] INDEX indexnév ON tábla(oszloplista);

utasítással lehetséges, amely a megadott tábla felsorolt oszlopaira, mint indexkulcsra generál indexet. Ha UNIQUE szerepel, akkor a tábla nem tartalmazhat két azonos indexkulcsú rekordot. Index törlése a

DROP INDEX indexnév;

utasítással történik. Példák:

```
CREATE INDEX DolgInd1 ON Dolgozó(név);
CREATE INDEX DolgInd2 ON Dolgozó(osztálykód,név);
```

Az első példa egyszerű indexkulcsot tartalmaz, amely a dolgozók név szerinti keresését, illetve rendezését támogatja. A második példában szereplő összetett indexkulcs az osztálykód szerinti, osztályon belül pedig név szerinti keresést/rendezést segíti, mivel a rendszerek általában az *osztálykód* és *név* attribútumok konkatenációjával képezik az indexkulcsot. Ez a megoldás viszont a pusztán név szerinti keresést nem támogatja.

6.4. Adattábla aktualizálása (DML)

A táblába *új sor felvétele* az

INSERT INTO táblanév [(oszloplista)] VALUES (értéklista);

utasítással történik. Ha *oszloplista* nem szerepel, akkor valamennyi oszlop értéket kap a CREATE TABLE-ben megadott sorrendben. Egyébként csak az oszloplistában megadott mezők kapnak értéket, a többi mező értéke NULL lesz.

Példák:

```
INSERT INTO Dolgozó (név, adószám)
    VALUES ('Tóth Aladár', 1111);
INSERT INTO Dolgozó
    VALUES (1111, 'Tóth Aladár', , '12');
```

A táblába adatokat tölthetünk át másik táblából is, ha a VALUES(értéklista) helyére egy lekérdezést írunk (lásd az *Alkérdések* fejezetben).

Sor(ok) módosítása az

UPDATE táblanév
SET oszlop = kifejezés, ..., oszlop = kifejezés
[WHERE feltétel];

utasítással történik. Az értékadás minden olyan soron végrehajtódik, amely eleget tesz a WHERE feltételnek. Ha WHERE feltétel nem szerepel, akkor az értékadás az összes sorra megtörténik.

Példák:

```
UPDATE Dolgozó
    SET lakcím = 'Szeged, Rózsa u. 5.'
    WHERE név = 'Kovács József';
UPDATE Dolgozó
    SET osztálykód = '003'
    WHERE osztálykód = '012';
UPDATE Dolgozó SET osztálykód = NULL;
```

Sor(ok) törlése a

DELETE FROM táblanév
[WHERE feltétel];

utasítással lehetséges. Hatására azok a sorok törlődnek, amelyek eleget tesznek a WHERE feltételnek. Ha a WHERE feltételt elhagyjuk, akkor az összes sor törlődik (de a séma megmarad).

Példák:

```
DELETE FROM Dolgozó
    WHERE név = 'Kovács József';
DELETE FROM Osztály;
```

79. Példa. Tekintsük az alábbi utasításpárt:

```
INSERT INTO Dolgozó (név, adószám)
    VALUES ('Tóth Aladár',4321);
DELETE FROM Dolgozó WHERE adószám = 4321;
```

Ha a táblában korábban már volt egy 4321 adószámú sor, akkor a fenti utasításpár azt is kitörli. Általában, ha egy tábla két azonos sort tartalmaz, DELETE utasítással nem tudjuk csak az egyiket kitörölni. Ha ugyanis a WHERE feltétel az egyikre igaz, akkor szükségképpen a másikra is igaz. A PRIMARY KEY feltétellel az ilyen anomáliák megelőzhetők.

6.5. Lekérdezés (DML)

Lekérdezésre a SELECT utasítás szolgál, amely egy vagy több adattáblából egy *eredménytáblát* állít elő. Az eredménytábla a képernyőn listázásra kerül, vagy más módon használható fel. (Egyetlen SELECT akár egy komplex felhasználói programot helyettesíthet!)

A SELECT utasítás alapváltozata:

SELECT [DISTINCT] oszloplista
FROM táblanévlista
[WHERE feltétel];

A "SELECT DISTINCT $A_1,...,A_n$ FROM $T_1,...,T_m$ WHERE feltétel" utasítás egyenértékű a következő relációs algebrai kifejezéssel:

$$E = \pi_{A1,...,An}(\sigma_{feltétel}(T_1 \text{ x...x } T_m))$$

Vagyis, a felsorolt táblák Descartes-szorzatából szelektáljuk a *feltétel*nek eleget tévő sorokat, majd ezekből projekcióval választjuk ki az E eredménytábla oszlopait. A DISTINCT opciót akkor kell kiírni, ha az eredménytáblában az azonos sorokból csak egyet kívánunk megtartani.

Ha *oszloplista* helyére * karaktert írunk, ez valamennyi oszlop felsorolásával egyenértékű. A SELECT legegyszerűbb változatával adattábla listázását érhetjük el:

```
SELECT * FROM T;
```

A relációs algebra műveleteinek megvalósítása

Projekció:

```
SELECT [DISTINCT] A1,...,An FROM T;
```

Példa:

```
SELECT DISTINCT szerző, cím FROM Könyv;
```

Szelekció:

```
SELECT * FROM T WHERE feltétel;
```

Példa:

```
SELECT * FROM Könyv WHERE kivétel < 2013.01.01;
```

Descartes-szorzat: T_1 x T_2

```
SELECT * FROM T1,T2;
```

Természetes összekapcsolás. Állítsuk elő például az Áru (cikkszám, megnevezés) és Vásárlás (cikkszám, mennyiség) táblák természetes összekapcsolását:

```
SELECT Áru.cikkszám, megnevezés, mennyiség
  FROM Áru, Vásárlás
  WHERE Áru.cikkszám = Vásárlás.cikkszám;
```

A fentivel egyenértékű, szintén gyakran használt szintaxis:

```
SELECT Áru.cikkszám, megnevezés, mennyiség
  FROM Áru INNER JOIN Vásárlás ON Áru.cikkszám = Vásárlás.cikkszám;
```

Megjegyzés. A fenti példákban a SELECT után nem elegendő csak „cikkszám”-ot írni, annak ellenére, hogy esetünkben „Áru.cikkszám = Vásárlás.cikkszám”, tehát mindegy, melyik cikkszámot választja a rendszer. Általában, ha egy lekérdezésben több azonos oszlopnév szerepel, az SQL rendszerek megkövetelik a táblanév megadását.

Külső összekapcsolás. A fenti példát alapul véve, ha az eredménytáblában valamennyi áru adatait szerepeltetni szeretnénk, akkor ez – az Oracle rendszer korábbi verzióiban használt jelöléssel – az alábbi módon adható meg:

```
SELECT Áru.cikkszám, megnevezés, mennyiség
  FROM Áru, Vásárlás
  WHERE Áru.cikkszám (+)= Vásárlás.cikkszám;
```

Az SQL szabvány szerint a LEFT, RIGHT vagy FULL OUTER JOIN kulcsszavakkal adható meg külső összekapcsolás, például:

```
SELECT Áru.cikkszám, megnevezés, mennyiség
  FROM Áru LEFT OUTER JOIN Vásárlás
  ON Áru.cikkszám = Vásárlás.cikkszám;
```

Théta join:

```
SELECT * FROM T1,T2 WHERE feltétel;
```

Unió:

```
(SELECT * FROM T1)
UNION
(SELECT * FROM T2);
```

A két SELECT eredménytáblája kompatibilis kell, hogy legyen (lásd Relációs algebra).

Metszet:

```
(SELECT * FROM T1)
INTERSECT
(SELECT * FROM T2);
```

A két SELECT eredménytáblája kompatibilis kell, hogy legyen.

Különbség:

```
(SELECT * FROM T1)
EXCEPT
(SELECT * FROM T2);
```

A két SELECT eredménytáblája kompatibilis kell, hogy legyen. Egyes rendszereknél EXCEPT helyett MINUS használatos.

80. Példa. Tekintsük az alábbi helyiség-adatbázist:
Helyiség (épület, ajtószám, név, alapterület)
Tanterem (*épület, ajtószám*, férőhely, tábla, vetítő)
Gépterem (*épület, ajtószám*, gépszám)

Kérjük le az oktatási célú géptermek listáját:

```
(SELECT épület, ajtószám FROM Tanterem)
INTERSECT
(SELECT épület, ajtószám FROM Gépterem);
```

Alias nevek

A SELECT után megadott *oszloplista* valójában nem csak oszlopneveket, hanem tetszőleges kifejezéseket is tartalmazhat, és az eredménytábla oszlopainak elnevezésére alias neveket adhatunk meg:

81. Példa. a Raktár(cikkszám, név, egységár, mennyiség) táblából egy E(áru, érték) tábla létrehozása:

```
SELECT név AS áru, egységár*mennyiség AS érték FROM Raktár;
```

82. Példa. a Személy(adószám, név, születésiév) táblából egy E(név, életkor) tábla létrehozása:

```
SELECT név, 2013-születésiév AS életkor FROM Személy;
```

A FROM után megadott táblák esetén is használhatók alias nevek, és erre szükség is van akkor, ha egy táblának önmagával való összekapcsolását képezzük:

83. Példa. Azonos nevű dolgozók lekérése a Dolgozó (adószám, név, lakcím) táblából:

```
SELECT d1.név, d1.adószám, d2.adószám
FROM Dolgozó AS d1, Dolgozó AS d2
WHERE d1.név=d2.név AND d1.adószám < d2.adószám;
```

Az adószámokra előírt feltétel azért kell, hogy önmagával ne párosítson rekordot, illetve, hogy egy azonos nevű pár csak egyszer jelenjen meg.

Függvények

ABS(n): abszolút érték *Példa:*

```
ABS(-15) = 15
```

LOWER(char): konverzió kisbetűsre. *Példa:*

```
LOWER('Kovács') = 'kovács'
```

UPPER(char): konverzió nagybetűsre. *Példa:*

```
UPPER('Kovács') = 'KOVÁCS'
```

LTRIM(char): balról szóközök eltávolítása. *Példa:*

```
LTRIM('  alma  ') = 'alma  '
```

RTRIM(char): jobbról szóközök eltávolítása. *Példa:*

```
RTRIM('  alma  ') = '  alma'
```

SUBSTR(char, m[, n]): a *char* string *m*-edik karakterétől *n* hosszú részstringet ad vissza. (Ha *n* nem szerepel, akkor a végéig.) Az első karakter 1-es sorszámú. *Példa:*

```
SUBSTR('ABCDEFG',2,3) = 'BCD'
```

TO_CHAR(n): konverzió numerikusról vagy dátumról karakteresre. *Példa:*

```
TO_CHAR(123) = '123'
```

TO_DATE(char): konverzió karakteresről dátumra. *Példa:*

```
TO_DATE('15-JAN-06')
```

TO_NUMBER(char): konverzió karakteresről numerikusra. *Példa:*

```
TO_NUMBER('123') = 123
```

Összesítő függvények

Egy oszlop értékeiből egyetlen értéket hoznak létre (például átlag). Általános alakjuk:

függvénynév ([DISTINCT] oszlopnév)

Ha DISTINCT szerepel, akkor az oszlopban szereplő azonos értékeket csak egyszer kell figyelembe venni. A számításnál a NULL értékek figyelmen kívül maradnak. Az egyes függvények:

AVG: átlagérték.
SUM: összeg.
MAX: maximális érték.
MIN: minimális érték.
COUNT: elemek száma. Ennél a függvénynél *oszlopnév* helyére * is írható, amely valamennyi oszlopot együtt jelenti.

Példák:
- SELECT AVG(fizetés) FROM Dolgozó: az eredménytábla egyetlen elemből áll, amely az átlagfizetést adja.
- SELECT SUM(fizetés) FROM Dolgozó: a fizetések összege.
- SELECT COUNT(*) FROM Dolgozó: a Dolgozó tábla sorainak száma, vagyis a dolgozók száma.
- SELECT COUNT(DISTINCT osztkód) FROM Dolgozó: az osztályok száma.

Csoportosítás (GROUP BY, HAVING)

Ha a tábla sorait csoportonként szeretnénk összesíteni, akkor a SELECT utasítás a

GROUP BY oszloplista

alparanccsal bővítendő. Egy csoportba azok a sorok tartoznak, melyeknél *oszloplista* értéke azonos. Az eredménytáblában egy csoportból egy rekord lesz. Az összesítő függvények csoportonként hajtódnak végre.

84. Példa. A Dolgozó táblából osztályonként az átlagfizetést számoljuk. Az eredménytáblának annyi sora lesz, ahány osztály van:

```
SELECT osztkód, AVG(fizetés) FROM Dolgozó
  GROUP BY osztkód;
```

85. Példa. A Projóra (dolgozó, projekt, óra) táblából dolgozónkénti és projektenkénti óraszám összegzés:

```
SELECT dolgozó, SUM(óra) FROM Projóra GROUP BY dolgozó;
SELECT projekt, SUM(óra) FROM Projóra GROUP BY projekt;
```

Csoportosítási szabály: A SELECT után összesítő függvényen kívül csak olyan oszlopnév tüntethető fel, amely a GROUP BY-ban is szerepel.

Hibás például az alábbi lekérdezés, amely azt szeretné megtudni, hogy az egyes osztályokon kinek a legnagyobb a fizetése:

```
SELECT osztkód, név, MAX(fizetés) AS maxfiz FROM Dolgozó GROUP BY osztkód;
```

A hiba oka: *név* nem szerepelhet a SELECT után, mert a GROUP BY után sem szerepel. (Ha egy osztályon több személynek is maximális a fizetése, a rendszer nem tudja eldönteni, hogy melyiknek a nevét írja ki. A lekérdezés helyes megoldását majd az Alkérdések fejezetben látjuk.)

A GROUP BY által képezett csoportok közül válogathatunk a

HAVING feltétel

alparancs segítségével: csak a feltételnek eleget tevő csoportok kerülnek összesítésre az eredménytáblába.

86. Példa. Azon osztályok listája, ahol az átlagfizetés > 180 000 Ft:

```
SELECT osztkód, AVG(fizetés) FROM Dolgozó
  GROUP BY osztkód
  HAVING AVG(fizetés) > 180000;
```

Az eredménytábla rendezése

Bár a relációs modell nem definiálja a rekordok sorrendjét, a gyakorlatban rendszerint valamilyen rendezettségben kívánjuk látni az eredményt. Erre szolgál az

ORDER BY oszlopnév [DESC], ..., oszlopnév [DESC]

alparancs, amely a SELECT utasítás végére helyezhető, és az eredménytáblának a megadott oszlopok szerinti rendezését írja elő. Az oszlopnév után írt ASC (ascending) növekvő, DESC (descending) csökkenő sorrendben való rendezést jelent. Alapértelmezés szerint a rendezés növekvő sorrendben történik, ezért ASC kiírása fölösleges.

87. Példa. Dolgozók és fizetéseik listája az osztálykód szerint növekvő, osztályon belül pedig fizetés szerint csökkenő sorrendben:

```
SELECT osztkód, név, fizetés FROM Dolgozó
  ORDER BY osztkód, fizetés DESC;
```

A SELECT utasítás általános alakja

A SELECT utasítás az alábbi alparancsokból állhat az alábbi sorrendben (a szögletes zárójelben szereplő részek elhagyhatók):

SELECT [DISTINCT] oszloplista	projekció
FROM táblanévlista	Descartes-szorzat
[WHERE feltétel]	szelekció
[GROUP BY oszloplista	csoportonként összevonás
[HAVING feltétel]]	csoport-szelekció
[ORDER BY oszloplista];	rendezés

Ahol "oszloplista" szerepel, ott általában oszlopkifejezések listáját lehet megadni (példák az *Alias nevek* alpontban). Az egyes alparancsok megadási sorrendje az angol nyelv szabályait követi (lásd fent a mintautasítást), végrehajtási sorrendjük viszont az alábbi:

1. FROM	Descartes-szorzat
2. WHERE	szelekció
3. GROUP BY	csoportonként összevonás
4. HAVING	csoport-szelekció
5. SELECT	projekció
6. ORDER BY	rendezés

A végrehajtási sorrend határozza meg, hogy melyik alparancsban mire lehet hivatkozni. Például GROUP BY után végrehajtott alparancsokban csak összesítő függvény és összesített oszlop adható meg (lásd *csoportosítási szabály*).

88. Példa. A Dolgozó (név, adószám, lakcím, *osztkód*, fizetés) és Osztály (osztkód, osztálynév, *vezAdószám*) táblákból kérjük le ábécé sorrendben azon osztályok nevét, ahol a legkisebb fizetés is nagyobb, mint 200 000:

```
SELECT osztálynév, MIN(fizetés)
FROM Dolgozó, Osztály
WHERE Dolgozó.osztkód=Osztály.osztkód
GROUP BY Dolgozó.osztkód, osztálynév
HAVING MIN(fizetés)>200000
ORDER BY osztálynév;
```

6.6. Alkérdések

Az SQL nyelv ismertetésének elején láttunk halmazokat tartalmazó logikai kifejezéseket. Egy ilyen halmaz SELECT utasítással is előállítható, például a

```
'Tóth Pál' IN (SELECT név FROM Dolgozó WHERE osztálykód='015')
```

logikai kifejezés akkor igaz, ha Tóth Pál a 015 kódú osztály dolgozója, vagy

```
EXISTS (SELECT * FROM Dolgozó WHERE fizetés < 80000)
```

akkor igaz, ha van 80000 Ft-nál kisebb fizetésű dolgozó.

Ha egy SELECT utasítás WHERE vagy HAVING feltételében olyan logikai kifejezés szerepel, amely SELECT utasítást tartalmaz, ezt *alkérdés*nek vagy *belső SELECT*-nek is nevezik. Általában, valamely SQL utasítás belsejében szereplő SELECT utasítást *alkérdés*nek nevezzük.

89. Példa. Az alábbi utasítás azon dolgozók listáját adja, amelyek fizetése kisebb, mint az átlagfizetés:

```
SELECT név, fizetés FROM Dolgozó
  WHERE fizetés < ( SELECT AVG(fizetés) FROM dolgozó );
```

Ebben a példában az alkérdést elég csak egyszer kiértékelni, hiszen a Dolgozó tábla minden egyes sorára ugyanazt az eredményt kapjuk. Ha viszont a belső SELECT-ben a külső SELECT-beli táblák oszlopnevei szerepelnek, akkor a külső SELECT minden egyes rekordjára kiértékelődik a belső SELECT. Egy kiértékelés során a külső változónevek konstansnak tekintendők. Ilyen a következő példa:

90. Példa. A Dolgozó(név, cím, osztálykód, fizetés) táblából azon dolgozók listáját kérjük, akiknek az osztályon belül a legnagyobb a fizetése (ha több ilyen van, mindegyiket ki kell listázni). A Dolgozó tábla két példányát a *D1* és *D2* alias nevek különböztetik meg:

```
SELECT osztálykód, név, fizetés FROM Dolgozó AS D1
  WHERE fizetés = ( SELECT MAX(fizetés) FROM Dolgozó AS D2
                      WHERE D1.osztálykód = D2.osztálykód );
```

91. Példa. Ügyeljünk a típuskompatibilitásra! **Hibás** például az alábbi lekérdezés WHERE feltétele, mert az alkérdés rekordhalmazt ad vissza, amely nem hasonlítható össze a fizetés értékkel:

```
SELECT adószám, név FROM Dolgozó
  WHERE fizetés = (SELECT * FROM Dolgozó WHERE név='Kovács');
```

Helyesen:

```
SELECT adószám, név FROM Dolgozó
  WHERE fizetés = (SELECT fizetés FROM Dolgozó WHERE adószám=1234);
```

92. Példa. Bizonyos esetekben az alkérdés join-műveletet helyettesít, például a
Könyv (könyvszám, szerző, cím, *olvasószám*, kivétel)
Olvasó (olvasószám, név, lakcím)
sémák esetén az alábbi két lekérdezés egyaránt a pécsi olvasók által kikölcsönzött könyvek listáját adja:

```
SELECT szerző, cím FROM Könyv WHERE olvasószám IN
  (SELECT olvasószám FROM Olvasó WHERE lakcím LIKE '%Pécs%');
SELECT szerző, cím FROM Könyv, Olvasó
  WHERE Könyv.olvasószám = Olvasó.olvasószám AND lakcím LIKE '%Pécs%';
```

Nem csak SELECT utasításban alkalmazható alkérdés:

93. Példa. Tekintsük a következő táblákat:
Dolgozó (adószám, név, fizetés)
Projekt (*adószám*, pkód, óraszám)

Az alábbi utasítás fizetésemelést hajt végre az A12 projekt dolgozóinál:

```
UPDATE Dolgozó
  SET fizetés=fizetés+10000
  WHERE adószám IN ( SELECT adószám FROM Projekt
                       WHERE pkód='A12' );
```

Nem csak a logikai kifejezés tartalmazhat alkérdést, hanem az INSERT utasítás is:

INSERT INTO táblanév [(oszloplista)] SELECT ... ;

A SELECT annyi oszlopot kell hogy kiválasszon, amennyit oszloplista tartalmaz. A többi oszlop NULL értéket vesz fel.

94. Példa. Tegyük fel, hogy a Raktár (cikkszám, név, egységár, mennyiség) táblából egy Készlet (áru, érték) táblát szeretnénk létrehozni, amely az áruféleség megnevezését és az aktuálisan tárolt mennyiség értékét tartalmazza. Ez a következőképp lehetséges:

```
CREATE TABLE Készlet
   ( áru   CHAR(20),
     érték INTEGER
   );
INSERT INTO Készlet
   SELECT név, egységár*mennyiség FROM Raktár;
```

6.7. Nézettáblák (virtuális táblák)

Egy adatbázisban általában kétféle adatra van szükségünk:
- alapadatok: tartalmukat aktualizáló műveletekkel módosítjuk.
- származtatott adatok: az alapadatokból generálhatók.

Származtatott adattáblát például INSERT ... SELECT segítségével is létrehozhatunk (lásd az előző pontot), ekkor viszont az nem követi automatikusan az alapadatok módosulását, ha pedig minden aktualizáló műveletnél újragenerálnánk, az rendkívül lassú lenne. A problémát a nézettábla oldja meg.

A nézettábla (virtuális tábla, view) *nem tárol adatokat*. Tulajdonképpen egy transzformációs formula, amelyet úgy képzelhetünk el, mint ha ennek segítségével a tárolt táblák adatait látnánk egy speciális szűrőn, „optikán" keresztül.

Nézettáblák alkalmazási lehetőségei:
- Származtatott adattáblák létrehozása, amelyek a törzsadatok módosításakor automatikusan módosulnak (pl. összegzőtáblák).
- Bizonyos adatok elrejtése egyes felhasználók elől (adatbiztonság vagy egyszerűsítés céljából).

Nézettábla létrehozása:

CREATE VIEW táblanév [(oszloplista)] AS alkérdés;

A SELECT utasítás eredménytáblája alkotja a nézettáblát. "Oszloplista" megadásával a nézettábla oszlopainak új nevet adhatunk. A CREATE VIEW végrehajtásakor a rendszer csak letárolja a nézettábla definícióját, és majd csak a rá való hivatkozáskor generálja a szükséges adatokat. Ebből adódóan a nézettábla tartalma mindig aktuális.

A nézettáblák általában ugyanúgy használhatók, mint a tárolt adattáblák, vagyis ahol egy SQL parancsban táblanév adható meg, ott rendszerint nézettábla neve is szerepelhet.

95. Példa. Származtatott adatok kezelése. A Raktár (cikkszám, név, egységár, mennyiség) táblából létrehozott nézettábla:

```
CREATE VIEW Készlet (áru, érték) AS
    SELECT név, egységár*mennyiség FROM Raktár;
```

96. Példa. Adatok elrejtése. A Dolgozó (adószám, név, lakcím, osztálykód, fizetés) táblához létrehozzuk a következő nézettáblát:

```
CREATE VIEW Dolg2 AS
    SELECT adószám, név, lakcím FROM Dolgozó
         WHERE osztálykód='A02';
```

Ha a nézettábla tartalmát *módosítjuk*, akkor a módosítás a megfelelő tárolt táblákon hajtódik végre – és természetesen megjelenik a nézettáblában is. Alapelv, hogy egy SQL rendszer *csak akkor engedi meg a nézettábla módosítását, ha a módosítást egyértelműen végre tudja hajtani a tárolt táblákon*. Nem lehet módosítani például a fenti *Készlet* tábla *érték* mezőjét, de a *Dolg2* tábla *lakcím* mezője már gond nélkül módosítható. Nem lehet módosítani továbbá a nézettáblát, ha definíciója
- DISTINCT opciót,
- FROM után egynél több táblanevet (join művelet),
- GROUP BY alparancsot

tartalmaz.

Példák a fenti korlátozások indokolására, a Dolg (adószám, név, lakcím) és ProjÓra (*adószám*, projektkód, óra) táblák alapján:

- DISTINCT esetén:

```
CREATE VIEW HardProj(projkód) AS
SELECT DISTINCT projektkód FROM Projóra WHERE óra>10;
```

azon projektek listáját adja, amelyeken valaki 10-nél több órában dolgozik. *Projkód* módosítása esetén a rendszer nem tudja eldönteni, hogy a ProjÓra táblában *projektkód* valamennyi előfordulását módosítsa-e, vagy csak azokat, ahol ora>10.

- Join művelet esetén:

```
CREATE VIEW DolgProj AS
SELECT név, projektkód, óra FROM Dolg, Projóra WHERE
Dolg.adószám=Projóra.adószám;
```

Ha egy dolgozó több projekten dolgozik, és csak az egyik rekordban a nevét módosítom, a rendszer nem tudja eldönteni, hogy a dolg táblában módosítsa-e a nevet.

- GROUP BY esetén:

```
CREATE VIEW SumProj AS
SELECT projektkód, SUM(óra) FROM Projóra WHERE óra<10 GROUP BY
projektkód;
```

az egyes projektekre a 10-nél kisebb óraszámokat összegzi. Itt a SUM(óra) mező nyilván nem módosítható, *projektkód* módosítása esetén pedig a rendszer nem tudja eldönteni, hogy a ProjÓra táblában a projektkód összes előfordulását módosítsa, vagy csak azokat, ahol óra<10.

Ha egy módosítható nézettáblába új rekordot veszünk fel, akkor az alaptáblának a nézettáblában nem szereplő oszlopaiba szükségképpen NULL kerül felvételre.

Tegyük fel, hogy a fenti *Dolg2* táblába új rekordot szeretnénk felvenni:

```
INSERT INTO Dolg2 VALUES (3333, 'Tóth Pál');
```

Mivel *osztálykód* nem szerepel Dolg2-ben, így értéke az új rekordban szükségképpen NULL lesz, vagyis az új dolgozó nem az 'A02' osztályra kerül felvételre, és így nem jelenik meg Dolg2-ben. A hiba kiküszöbölhető, ha az *osztálykód*ot felvesszük Dolg2-be:

```
CREATE VIEW Dolg2 AS
    SELECT adószám, név, lakcím, osztálykód FROM Dolgozó
        WHERE osztálykód='A02';
INSERT INTO Dolg2
    VALUES (3333, 'Tóth Pál', , 'A02');
```

Ha a CREATE VIEW utasítás végére a WITH CHECK OPTION záradékot illesztjük, akkor a rendszer nem engedi meg a nézettábla olyan módosítását, amely nem tesz eleget a leválogatási feltételnek. Például,

```
CREATE VIEW Dolg2 AS
    SELECT adószám, név, lakcím, osztálykód FROM Dolgozó
        WHERE osztálykód='A02' WITH CHECK OPTION;
```

nem engedi meg az osztálykód módosítását, vagy 'A02'-től különböző osztálykód felvitelét.

Lekérdezések kiértékelése. A nézettáblára vonatkozó lekérdezést relációs algebrai formulával írjuk fel, ebbe behelyettesítjük a nézettábla definícióját, és a kapott formulát értékeljük ki az alaptáblákra. Példa:

```
SELECT lakcím FROM Dolg2 WHERE név='Tóth Pál';
```

Ez relációs algebrával felírva:

$E = \pi_{lakcim}(\sigma_{név='Tóth Pál'}(Dolg2))$, ahol

$Dolg2 = \pi_{adószám,név,lakcím,osztálykód}(\sigma_{osztálykód='A02'}(Dolgozó))$

A Dolg2 behelyettesítésével adódó formulát kell kiértékelni.

7. Aktív elemek (megszorítások, triggerek)

Aktív elem: olyan programrész, amely bizonyos szituációban automatikusan végrehajtódik. Ennek speciális esete a *megszorítás*, ami bizonyos feltételek ellenőrzését jelenti bizonyos helyzetekben.

7.1. Attribútumok megszorításai

A CREATE TABLE-ben valamely attribútum deklarációja után adhatók meg.

Kulcs feltételek: a CREATE TABLE utasításban adhatók meg a PRIMARY KEY, UNIQUE, REFERENCES kulcsszavakkal. Aktualizálási műveleteknél a megfelelő feltétel automatikus ellenőrzését váltják ki.

További megszorítások:

NOT NULL

Adott attribútum értéke nem lehet NULL. Hatására a rendszer megakadályoz minden olyan műveletet, amely az adott attribútum NULL értékét eredményezné. Adatbevitelnél például ez azt jelenti, hogy az attribútum értékét kötelező megadni.

CHECK (feltétel)

Az adott attribútum módosítását a rendszer csak akkor engedi meg, ha a feltétel teljesül.

97. Példa: A dolgozók nemét is nyilvántartjuk (F=férfi, N=nő):

```
CREATE TABLE Dolgozó
   ( adószám  DECIMAL(10)  PRIMARY KEY,
     név      CHAR(30)  NOT NULL,
     nem      CHAR(1)  CHECK (nem IN ('F', 'N')),
     lakcím   CHAR(40),
     osztkód  CHAR(3) REFERENCES Osztály(osztkód)
   );
```

98. Példa. Külső kulcs feltétel csak korlátozottan ellenőrizhető CHECK-feltétellel:

```
CREATE TABLE Dolgozó
   ( adószám    DECIMAL(10)  PRIMARY KEY,
     név        CHAR(30),
     lakcím     CHAR(40),
     osztálykód CHAR(3)
       CHECK (osztálykód IN (SELECT osztálykód FROM Osztály))
   );
```

A fenti CHECK biztosítja, hogy a Dolgozó tábla csak létező osztálykódra hivatkozhat, de az Osztály tábla változásainál már nem ellenőrzi a külső kulcs feltételt. Vagyis a CHECK feltétel ellenére előállhat olyan Dolgozó tábla, amelyre a feltétel nem teljesül.

Értéktartomány definiálása:

CREATE DOMAIN név típus [DEFAULT érték] [CHECK (feltétel)];

Értéktartomány módosítása ALTER DOMAIN, törlése DROP DOMAIN utasítással történik.

99. Példa. A nemekhez tartozó konstansértékek definiálása:

```
CREATE DOMAIN NemÉrték CHAR(1) CHECK (VALUE IN ('F', 'N'));
```

Használata:

```
CREATE TABLE Dolgozó
   ( adószám  DECIMAL(10)  PRIMARY KEY,
     név      CHAR(30),
     nem      NemÉrték,
     lakcím   CHAR(40)
   );
```

7.2. Táblára vonatkozó megszorítások

A CREATE TABLE végére, a táblaFeltételeknél helyezendők el. *Kulcs feltételek:* PRIMARY KEY, UNIQUE, FOREIGN KEY kulcsszavakkal. Ha a CHECK feltétel egyszerre több attribútumot érint, akkor szintén a táblaFeltételeknél helyezendő el.

100. Példa. Biztonsági ellenőrzésként megköveteljük, hogy a könyvek kölcsönzésénél a kivétel dátuma előzze meg a visszahozási határidőt::

```
CREATE TABLE Könyv
   ( könyvszám DECIMAL(6)  PRIMARY KEY,
     szerző    CHAR(30),
     cím       CHAR(30),
     kivétel   DATE,
     vissza    DATE,
     CHECK (kivétel < vissza)
   );
```

7.3. Általános megszorítások

Több táblára (általában, a teljes adatbázissémára) vonatkozhatnak. Megadásuk:

CREATE ASSERTION név CHECK (feltétel);

A feltételben szereplő táblák bármelyikének módosításakor a feltétel ellenőrzésre kerül.

101. Példa. A Dolgozó(adószám, név, fizetés, osztálykód) és Osztály(osztálykód, osztálynév, vezAdószám) táblák esetén megköveteljük, hogy a vezetők fizetése legalább 100 000 Ft legyen:

```
CREATE ASSERTION VezetőFizetés
  CHECK (NOT EXISTS
    (SELECT * FROM Dolgozó, Osztály
    WHERE Dolgozó.adószám = Osztály.vezAdószám
    AND fizetés < 100000));
```

A feltétel két esetben sérülhet: ha egy dolgozó fizetését változtatjuk, vagy ha egy dolgozót vezetőnek nevezünk ki. Ezért a fenti önálló megszorítás nem helyettesíthető egyetlen táblára vonatkozó megszorítással.

Az önálló megszorítás törlése:

DROP ASSERTION név;

7.4. Megszorítások kezelése

A megszorításokat célszerű elnevezni a "CONSTRAINT név" előtag segítségével. Például a Dolgozó tábla név attribútuma esetén:

```
név CHAR(30) CONSTRAINT NévKulcs UNIQUE
```

Ezután a kulcsfeltétel elvethető a következő utasítással:

```
ALTER TABLE Dolgozó DROP CONSTRAINT NévKulcs;
```

A kulcsfeltétel újra érvényesíthető táblafeltételként:

```
ALTER TABLE Dolgozó ADD CONSTRAINT NévKulcs UNIQUE (név);
```

Értéktartományra vonatkozó megszorítás esetén:

```
CREATE DOMAIN NemÉrték AS CHAR(1)
  CONSTRAINT FérfiVagyNő CHECK (VALUE IN ('F', 'N'));
```

Értéktartományra vonatkozó megszorítás hasonlóan módosítható:

```
ALTER DOMAIN NemÉrték DROP CONSTRAINT FérfiVagyNő;
```

7.5. Triggerek

A trigger egy aktualizálási művelet esetén végrehajtandó programrészletet definiál. Megadása:

CREATE TRIGGER név
{ BEFORE | AFTER | INSTEAD OF }
{ DELETE | INSERT | UPDATE [OF oszlopok] }
ON tábla
[REFERENCING [OLD AS régi] [NEW AS új]
[FOR EACH ROW]
[WHEN (feltétel)] programblokk;

Jelölés: a fenti szintaxis leírásban { x | y } azt jelenti, hogy x és y egyike választható.

név: a trigger neve.

BEFORE, AFTER, INSTEAD OF: az aktualizálási művelet előtt, után, vagy helyette lép működésbe a trigger.

DELETE, INSERT, UPDATE OF: az aktualizálási művelet neve.

ON tábla: ezen tábla aktualizálásakor lép működésbe a trigger.

REFERENCING: lehetővé teszi, hogy a tábla aktuális sorának aktualizálás előtti és utáni állapotára névvel hivatkozzunk.

FOR EACH ROW: ha megadjuk, akkor a trigger a tábla minden egyes sorára lefut, amelyet az aktualizálási művelet érint (sor szintű trigger). Ha nem adjuk meg, akkor egy aktualizálási művelet esetén csak egyszer fut le a trigger (utasítás szintű trigger).

WHEN feltétel: a megadott feltétel teljesülése esetén hajtódik végre a trigger.

programblokk: egy vagy több SQL utasításból álló, vagy valamely programozási nyelven írt blokk.

102. Példa sor szintű triggerre. Az alábbi trigger egy FizetésNapló (dátum, *adószám*, régifiz, újfiz) táblában gyűjti a fizetés-módosítások adatait:

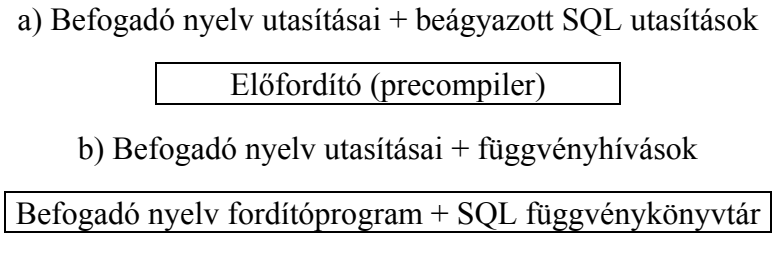
```
CREATE TRIGGER fiz_napló
  AFTER UPDATE OF fizetés ON Dolgozó
  REFERENCING OLD AS régi NEW AS új
  FOR EACH ROW
INSERT INTO FizetésNapló
  VALUES (SYSDATE, régi.adószám,
          régi.fizetés, új.fizetés);
```

A trigger engedélyezett vagy letiltott állapotban lehet. Létrehozáskor engedélyezett, változtatás ALTER TRIGGER utasítással lehetséges (nem részletezzük).

8. Beágyazott SQL

Az SQL lehetőségeivel nem oldható meg minden adatbázis kezelési feladat. SQL-ben például nem használhatók változók és vezérlési szerkezetek, így az adatbázis algoritmikus kezelése sem lehetséges.

Ezért az SQL utasításokat általában egy hagyományos algoritmikus programnyelv (C, Java, stb.) utasításaival keverten használjuk, és az SQL utasításokban felhasználhatók a befogadó programnyelv változói is. Ezt a megoldást nevezzük *beágyazott SQL*-nek (embedded SQL).



22. ábra. Beágyazott SQL fordítása

Jellemző megoldási módok:

- Precompiler alkalmazása (22. ábra), amely a forráskódban felismeri az SQL utasításokat, és lecseréli azokat a befogadó nyelv függvényhívásaira (például Oracle Pro*C).
- Az SQL nyelvet algoritmikus lehetőségekkel bővítik. Itt valójában nincs befogadó nyelv, az algoritmikus nyelv és az SQL szerves egységet képez. (Ilyen például az Oracle rendszer PL/SQL nyelve.) Ezt úgy képzelhetjük el, mint ha a 22. ábrán *a)*-ból közvetlen fordítással adódna *c)*.
- A befogadó nyelvben beágyazott SQL utasítások helyett csak a nekik megfelelő függvényhívások használhatók (például ODBC, JDBC, PHP). Ekkor a 22. ábrán eleve a *b)* fokozatról indulunk.

A továbbiakban részletesebben megnézünk néhány SQL beágyazási módszert.

8.1. SQL beágyazás ANSI C-be

Nem konkrét implementációt, hanem az SQL2 szabvány által definiált általános megoldást tárgyaljuk. Befogadó nyelvként ANSI C-t tételezünk fel. Minden beágyazott SQL utasítás elé EXEC SQL írandó, az előfordító ez alapján ismeri fel a neki szóló utasításokat.

Kommunikációs változók: a befogadó nyelv azon változói, amelyeket SQL utasításokban is használni kívánunk. Ezeket

EXEC SQL BEGIN DECLARE SECTION;
...
EXEC SQL END DECLARE SECTION;

utasítások között kell deklarálni. Csak olyan típusok használhatók, amelyeket a befogadó nyelv és az SQL implementáció egyaránt támogat.

A beágyazott SQL utasításokban lényegében bárhol használhatunk kommunikációs változót, ilyenkor annak neve elé kettőspont írandó.

SQLSTATE változó: hibakódot tartalmaz, az SQL utasítások állítják be. Általában 5 karakterből áll, hibátlan végrehajtás esetén értéke '00000'.

103. Példa. Rekord felvétele a könyv táblába. A program a 23. ábrán látható.

```
void újkönyv()
{ EXEC SQL BEGIN DECLARE SECTION;
    char kszám[6];
    char kszerző[30];
    char kcím[50];
    char SQLSTATE[6]; // a stringlezáró karakter miatt 5+1 elemű
  EXEC SQL END DECLARE SECTION;
  /* Itt a képernyőről bekéri a könyvszám, szerző, cím adatokat
     és letárolja a megfelelő változókba. */
  EXEC SQL INSERT INTO Könyv VALUES (:kszám, :kszerző, :kcím);
  if (strcmp(SQLSTATE,"00000")) ...;      // hibaüzenet kiírása
}
```

23. ábra. Új rekord felvétele a Könyv táblába

Lekérdezések, kurzorok

A SELECT utasítás beágyazása problematikus, mivel eredménytáblát ad vissza. Két eset lehetséges:

a) Egysoros lekérdezés. Ha a SELECT csak egy sort ad vissza, akkor

EXEC SQL SELECT oszlopok INTO változók FROM ...;

alakban használható. Ha a SELECT nem egy sort ad vissza, akkor a változók nem kapnak értéket, és SQLSTATE megfelelően beállításra kerül. Példák:

```
EXEC SQL SELECT szerző, cím INTO :kszerző, :kcim
  FROM Könyv WHERE könyvszám = :kszám;

EXEC SQL SELECT AVG(fizetés) INTO :átlagfiz FROM Dolgozó;
```

b) Többsoros lekérdezés. Ha a SELECT több sort ad vissza, akkor egy rekordmutatót, úgynevezett *kurzor*t kell definiálni:

EXEC SQL DECLARE kurzornév CURSOR FOR alkérdés;

A kurzor a *lekérdezés* (SELECT utasítás) által definiált eredménytáblához rendelődik. Használat előtt a kurzort meg kell nyitni:

EXEC SQL OPEN kurzor;

Hatására a kurzor a tábla első sora elé mutat. A kurzort léptetni az

EXEC SQL FETCH FROM kurzor INTO változólista;

utasítással lehet. Hatására a kurzor a soron következő rekordra lép, és annak mezői a *változólista* megfelelő elemeibe tárolódnak. Ha a FETCH elérte a tábla végét (az utolsó utáni rekordra lép), akkor a változók nem kapnak értéket, és SQLSTATE-be a "02000" konstans kerül.

Használat után a kurzort le kell zárni:

EXEC SQL CLOSE kurzor;

A lejárt kurzor újabb OPEN-nel újra megnyitható, így a tábla többször végigjárható.

104. Példa. Készítsünk kimutatást egy vállalat dolgozóiról, amely megadja, hogy a 80 000, 120 000, 200 000, 300 000, 500 000 értékek által határolt jövedelemsávokba hány dolgozó esik. A program a 24. ábrán látható. Az eredmény a *dolgozoSzam* tömbben keletkezik.

```
void jövedelemSávok()
{ int határ[5] = {80000, 120000, 200000, 300000, 500000};
  int dolgozóSzám[6] = {0, 0, 0, 0, 0, 0};
  int i;
  EXEC SQL BEGIN DECLARE SECTION;
    int jövedelem;
    char SQLSTATE[6];
  EXEC SQL END DECLARE SECTION;

  EXEC SQL DECLARE sor CURSOR FOR
    SELECT fizetés FROM Dolgozó;
  EXEC SQL OPEN sor;

  while (1)
  { EXEC SQL FETCH FROM sor INTO :jövedelem;
    if ( strcmp(SQLSTATE,"02000")==0 ) break;
    for (i=0; i<5; i++)
      if (jövedelem < határ[i]) break;
    dolgozóSzám[i]++;
  }
  EXEC SQL CLOSE sor;
}
```

24. ábra. Jövedelem statisztikát készítő program

Ha a tábla rekordjait más sorrendben kívánjuk bejárni, a kurzor deklarációjába a SCROLL szót kell illeszteni:

EXEC SQL DECLARE kurzornév SCROLL CURSOR FOR lekérdezés;

Ezután a FETCH utasításban az alábbi kulcsszavak használhatók:
- NEXT: következő sor (ez az alapértelmezés),
- PRIOR: előző sor,
- FIRST, LAST: első ill. utolsó sor,
- RELATIVE n: n sorral előre (vagy vissza, ha n negatív),
- ABSOLUTE n: az n-edik sor.

Példa:

```
EXEC SQL FETCH LAST FROM sor INTO :jövedelem;
```

Ha a sorokat valamilyen rendezettség szerint kívánjuk bejárni, akkor a kurzort deklaráló SELECT-ben az ORDER BY alparancsot kell alkalmazni. Példa:

```
EXEC SQL DECLARE sor CURSOR FOR
  SELECT fizetés FROM Dolgozó
  ORDER BY név;
```

```
void rendelés()
{ EXEC SQL BEGIN DECLARE SECTION;
    char vevő[20];
    char csz[12];
    int eár, menny, érték;
    char SQLSTATE[6];
  EXEC SQL END DECLARE SECTION;

  EXEC SQL DECLARE rendelésSor CURSOR FOR
    SELECT * FROM Rendelés;
  EXEC SQL OPEN rendelésSor;

  while (1)
  { EXEC SQL FETCH FROM rendelésSor INTO :vevő, :csz, :menny, :érték;
    if ( strcmp(SQLSTATE,"02000")==0 ) break;
    EXEC SQL SELECT egységár INTO :eár FROM Áru
      WHERE cikkszám = :csz;
    érték = eár * menny;
    if (érték < 2000)
      EXEC SQL DELETE FROM Rendelés
        WHERE CURRENT OF rendelésSor;
    else
      EXEC SQL UPDATE Rendelés SET érték = :érték
        WHERE CURRENT OF rendelésSor;
  }
  EXEC SQL CLOSE rendelésSor;
}
```

25. ábra. Rendelések feldolgozása

Aktualizáló műveletek kurzorral

Az UPDATE és DELETE utasítások a kurzor sorára is alkalmazhatók, ha a WHERE feltételben *CURRENT OF kurzornév* szerepel.

105. Példa. Egy kereskedő cég az árukat és a beérkező rendeléseket az alábbi táblákban tartja nyilván:

Áru (cikkszám, megnevezés, egységár)
Rendelés (vevő, *cikkszám*, mennyiség, érték)

Feladat: a rendelések feldolgozása úgy, hogy meghatározzuk minden tétel értékét (egységár*mennyiség). Ha ez kisebb 2000-nél, akkor a rendelést töröljük, egyébként beírjuk az értéket a Rendelés táblába. A program a 25. ábrán látható.

Dinamikus SQL

Ha egy adatbázis-alkalmazást igazán rugalmassá kívánunk tenni, akkor a felhasználó számára biztosíthatjuk, hogy maga is megfogalmazhasson lekérdezéseket. Ilyenkor a megfelelő SQL utasítás csak futás közben állítható elő, fordítási időben még nem. Ezt teszi lehetővé az

EXEC SQL PREPARE sqlutasítás FROM string;

utasítás, amely a befogadó nyelven előállított *string* karaktersorozatot elemzi, és belőle az *sqlutasítás* SQL-változóba előállítja a megfelelő (végrehajtható belső formátumú) SQL-utasítást. Ezután az

EXEC SQL EXECUTE sqlutasítás;

segítségével végrehajtható az utasítás. Minden egy lépésben is elvégezhető az

EXEC SQL EXECUTE IMMEDIATE string;

utasítással. (A szétválasztás akkor indokolt, ha az elemzett utasítást sokszor kell végrehajtani, és a többszöri elemzés idejét meg akarjuk takarítani.) Az eljárás alkalmazására a 26. ábra ad példát.

```
void felhasználóiKérdés()
{ EXEC SQL BEGIN DECLARE SECTION;
    char *kérdés;
  EXEC SQL END DECLARE SECTION;

  /* A felhasználó által megadott kérdésből SQL utasítást
     tartalmazó string szerkesztése 'kérdés'-be */

  EXEC SQL EXECUTE IMMEDIATE :kérdés;
}
```

26. ábra. Felhasználói lekérdezést feldolgozó program

8.2. ODBC

ODBC = Open Database Connectivity
ODBC 1.0 specifikáció: Microsoft, 1992.
Az ODBC magja szabványos, vagyis lényegében megfelel az SQL:1999 szabvány CLI (= Call-Level Interface) specifikációjának, amelyet röviden SQL/CLI-nek neveznek.

A lényeg: normál C nyelvű programot írhatunk, amelynél egy függvénykönyvtár segítségével érjük el az adatbázist, alapvetően SQL utasításokat küldhetünk a DBMS-nek. Ezzel bizonyos rendszerfüggetlenség érhető el: különböző platformokon és különböző DBMS-ek esetén ugyanaz a forrásprogram használható. (Probléma viszont, hogy az egyes DBMS-ek SQL-szintaxisa eltérhet.)
Windows környezetben a befogadó program elején általában az alábbiakat kell include-olni:

```
#include <stdio.h>
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
```

Hatására az ODBC függvények, típusok, konstansok használhatók. Az ODBC-függvények által visszaadott érték SQLRETURN típusú, értéke 0 hibátlan végrehajtás esetén.

Az alábbi adatstruktúrák használhatók:
- *Környezet (Environment):* a kliens hozza létre a DBMS-sel való kapcsolat előkészítéséhez.
- *Kapcsolat (Connection):* DBMS-sel való kapcsolat leírására szolgál. Egy környezethez több kapcsolat tartozhat.
- *ODBC-utasítás (Statement):* egy SQL utasítás leírására szolgál. Minden ODBC-utasítás valamely kapcsolathoz tartozik. Ugyanaz az ODBC-utasítás különböző időpontokban különböző SQL-utasításokat tartalmazhat.

A fentiek kezelése handle-k (az adatstruktúrára mutató pointerek) segítségével történik. Ezek típusai sorrendben SQLHENV, SQLHDBC, SQLHSTMT.

Handle létrehozására szolgáló függvény:

```
SQLAllocHandle(hType, hIn, hOut)
```

hType: a handle típusa, lehetséges értékei:
SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT.
hIn: a magasabb szintű elemet megadó handle.
Környezet esetén SQL_NULL_HANDLE adandó meg.
hOut: az SQLAllocHandle által létrehozott handle címe.

Példa adatbázis-szerverhez való kapcsolódásra (a kipontozott részek a konkrét szoftverkörnyezettől függenek):

```
SQLHENV env;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLRETURN ret;
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
// beállítjuk a környezeti paramétereket:
SQLSetEnvAttr(env, ...);
```

```
SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc);
// megnyitjuk a kapcsolatot:
ret = SQLDriverConnect(dbc, ...);
// ellenőrizzük, hogy a kapcsolatteremtés sikeres volt-e
  if (SQL_SUCCEEDED(ret)) {
    printf("Kapcsolat létrejött\n");
  } else {
    printf("Sikertelen kapcsolódás\n");
    exit(-1);
  }
SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
```

SQL utasítás előkészítése:

```
SQLPrepare(sh, st, sl)
```

sh: utasítás handle

st: SQL utasításra mutató pointer

sl: az SQL utasítás hossza (karaktereinek száma). SQL_NTS megadása esetén a rendszer maga állapítja meg a hosszat a lezáró null-karakter alapján.

A függvény hatására az *sh* handle a továbbiakban az *st* utasítást reprezentálja.

SQL utasítás végrehajtása:

```
SQLExecute(sh)
```

sh: utasítás handle

A végrehajtás evidens INSERT, UPDATE, DELETE esetén. SELECT esetén úgy kell elképzelni, hogy a lekérdezés eredménye valahol létrejön készen arra, hogy elérjük egy implicit kurzorral.

SQL utasítás közvetlen végrehajtása:

```
SQLExecDirect(sh, st, sl)
```

sh: utasítás handle

st: SQL utasításra mutató pointer

sl: az SQL utasítás hossza

Az utasítás hatása egyenértékű az

```
SQLPrepare(sh, st, sl)
SQLExecute(sh)
```

párral.

Példa: Egy árukat nyilvántartó Raktár(cikkszám, megnevezés, egységár, mennyiség) táblában az árakat csökkenti 10%-kal:

```
  SQLPrepare(stmt, "UPDATE raktár SET egységár = egységár*0.9", SQL_NTS);
  SQLExecute(stmt);
```

Implicit kurzor léptetése:

```
SQLFetch(sh)
```

Feltételezzük, hogy az *sh* utasítás már végrehajtásra került, egyébként a fetch hibát jelez. Ha a függvény visszaadott értéke az SQL_NO_DATA_FOUND konstanssal jelölt érték, ez azt jelenti, hogy a lekérdezés nem adott vissza több értéket (tábla vége).

Példa:

```
  ret = SQLFetch(stmt);
  if (ret == SQL_NO_DATA_FOUND) printf("\n Nincs adat.\n");
```

Tábla oszlopainak kapcsolása befogadó nyelvi változókhoz:

```
SQLBindCol(sh, colNo, colType, pVar, varSize, varInfo)
```

sh: utasítás handle

colNo: az oszlop sorszáma a táblában

colType: az oszlopnak megfelelő befogadó nyelvi típus. Lehetséges értékei például SQL_C_CHAR, SQL_C_SHORT.

pVar: pointer a befogadó nyelvi változóra.

varSize: a pVar-nak megfelelő változó mérete byte-ban.

varInfo: pointer egy integer változóra, amelyben az SQLBindCol függvény további információt helyezhet el.

Példa: A Raktár táblában adott árhoz legközelebbi egységárú cikk adatainak lekérése:

```
int legközelebbiCikk(int adottár) {
  int diff, különbség, jóCikk;
  SQLHENV env;
  SQLHDBC con;
  SQLHSTMT stmt;
  SQLINTEGER c, a, cInfo, aInfo;

  diff = jóCikk = -1;
  SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
  SQLSetEnvAttr(...);
  SQLAllocHandle(SQL_HANDLE_DBC, env, &con);
  SQLAllocHandle(SQL_HANDLE_STMT, con, &stmt);
  SQLDriverConnect(...);
  SQLPrepare(stmt, "SELECT cikkszám, egységár FROM Raktár", SQL_NTS);
  SQLExecute(stmt);
  SQLBindCol(stmt, 1, SQL_C_SHORT, &c, size(c), &cInfo);
  SQLBindCol(stmt, 2, SQL_C_SHORT, &a, size(a), &aInfo);
  while (SQLFetch(stmt) != SQL_NO_DATA_FOUND) {
    különbség = abs(a - adottár);
    if (diff == -1 || diff > különbség) {
      diff = különbség;
      jóCikk = c;
    }
  }
  return (jóCikk);
}
```

Paraméterek átadása:

Az SQLPrepare-ben a paraméterek helyére kérdőjel írandó. Az i-edik kérdőjel felel meg az i-edik paraméternek. A paraméterekhez érték rendelhető

```
SQLBindParameter(...)
```

segítségével. A függvénynek 10 argumentuma van, alább csak a fontosabbakat használjuk.

Példa: INSERT utasítás paraméterezése a Dolgozó(adószám, név, cím, fizetés) táblára:

```
SQLPrepare(utasitas, "INSERT INTO dolgozo(nev, cim) VALUES (?, ?)",
  SQL_NTS);
SQLBindParameter(utasitas, 1,..., dolgozonev,...);
SQLBindParameter(utasitas, 2,..., dolgozocim,...);
SQLExecute(utasitas);
```

Lekérdező ciklusok optimalizálása:

Mivel az eredményhalmaz soronkénti lekérése lassú, megadhatunk sorhalmazt:

```
SQLSetStmtAttr(stmt, SQL_ATTR_ROW_ARRAY_SIZE, 20,...)
```

A fenti példában egyszerre 20 sort kérünk le, amely például kényelmesen elférhet egy hálózati csomagban. Ezzel viszont a ciklusszervezés jóval bonyolultabbá válik, a részletektől eltekintünk.

8.3. JDBC

JDBC = (Java Database Connectivity). Az ODBC-hez hasonló, de a Java objektum-orientált jellegének felel meg.

Először egy *JDBC driver betöltése* szükséges a megfelelő DBMS-hez (ennek módja platformfüggő). Eredményeként egy DriverManager objektum jön létre, amely az ODBC-beli „környezet”-nek felel meg.

Kapcsolódás az adatbázishoz (az ODBC „kapcsolat” létrehozásához hasonlóan):

```
Connection kapcsolat = DriverManager.getConnection(url,user,password)
```

Vagyis, a DriverManager getConnection metódusát alkalmazva egy Connection típusú változó jön létre.

url: az adatbázist azonosítja, például "jdbc:mysql://home.cab.u-szeged.hu:3306/test".

user: a DBMS-felhasználó azonosítója.

password: a DBMS-felhasználó jelszava.

Utasítás létrehozása a CreateStatement metódus paraméteres és paraméter nélküli változatával lehetséges:

```
CreateStatement()
```

Statement típusú objektumot ad vissza. SQL utasítás nem tartozik hozzá, hasonlóan a ODBC-beli SQLAllocHandle-hez.

```
CreateStatement(sqlutasitás)
```

SQL-utasításstringet kap, és PreparedStatement típusú objektumot ad vissza. ODBC-ben az SQLAllocHandle + SQLPrepare párnak felel meg.

Utasítás végrehajtása: két-két (paraméteres és paraméter nélküli) változat: "query" lekérdezésekre, "update" minden módosító utasításra (INSERT, CREATE TABLE stb.) vonatkozik.

```
executeQuery(sqllekérdezés)
```

Statement objektumra hajtódik végre, ResultSet típusú objektumot ad vissza, amely az eredménysorok multihalmaza.

```
executeQuery()
```

PreparedStatement objektumra hajtódik végre. Szintén ResultSet objektumot ad vissza.

```
executeUpdate(sqlmódosítás)
```

Statement objektumra hajtódik végre, az adatbázist módosítja, nincs visszaadott eredményhalmaz.

```
executeUpdate()
```

PreparedStatement objektumra hajtódik végre, egyébként mint az előző.

Példa: Egy árukat nyilvántartó Raktár(cikkszám, megnevezés, egységár, mennyiség) táblában a cikkek árát csökkenti 10%-kal:

```
void árcsökkentés() {
  Connection kapcsolat = DriverManager.getConnection(...);
  Statement stmt = kapcsolat.createStatement();
  stmt.executeUpdate("UPDATE Raktár SET egységár = egységár*0.9");
  kapcsolat.close();    //kapcsolat lezárása
}
```

Implicit kurzor használata:

A ResultSet osztályhoz az alábbi metódusok tartoznak:

`next()`: az implicit kurzort a következő sorra lépteti (első meghíváskor lép az első sorra). FALSE értéket ad vissza, ha nincs több sor.

`getString(i), getInt(i), getFloat(i)`, stb.: az aktuális sor i-edik mezőjét adja vissza.

Példa: A Raktár táblában adott árhoz legközelebbi egységárú cikk adatainak lekérése:

```
int legközelebbiár(int adottár) {
  Connection kapcsolat = DriverManager.getConnection(...);
  PreparedStatement stmt = kapcsolat.createStatement(
      "SELECT cikkszám, egységár FROM Raktár"
  );
  ResultSet tábla = stmt.executeQuery();
  int diff = -1;
  int jóCikk = -1;
  while(tábla.next()) {
    int c = tábla.getInt(1);
    int a = tábla.getInt(2);
    int aktdiff = (a - adottár)*(a - adottár);
    if(diff == -1 || diff > aktdiff) {
      diff = aktdiff;
      jóCikk = c;
    }
  }
  kapcsolat.close();
  return(jóCikk);
}
```

Paraméterek átadása:

Az ODBC-hez hasonlóan kérdőjelekkel történik. A `setString(i, v)`, `setInt(i, v)`, stb. metódusokat használhatjuk, amelyek az SQL-utasítás *i*-edik paraméteréhez a *v* értéket rendelik.

Példa: INSERT utasítás paraméterezése a Dolgozó(adószám, név, cím, fizetés) táblára:

```
PreparedStatement utasitas = kapcsolat.createStatement(
  "INSERT INTO dolgozo(nev, cim) VALUES (?, ?)");
utasitas.setString(1, nev);
utasitas.setString(2, cim);
utasitas.executeUpdate();
```

8.4. PHP

A PHP tulajdonképpen egy általános célú algoritmikus nyelv, amelyet dinamikus weboldalak előállítására terveztek (PHP = Personal Home Page). Rendszerint az alábbi három szoftvert együtt alkalmazzák:

- *Apache:* közkedvelt web szerver program. Letölthető: www.apache.org
- *PHP interpreter.* Letölthető: www.php.net.

- *MySQL:* adatbázis szerver. Letölthető: www.mysql.com.

Célszerűbb azonban ezeket nem külön-külön letölteni, hanem együtt, az XAMPP telepítő csomag formájában: www.apachefriends.org/en/xampp.html

A fejlesztési technológia lényege:

- A statikus, HTML nyelvű weblapok forrásszövegébe PHP programrészeket illesztünk. Az Apache-ba integrált PHP-értelmező ezeket végrehajtja, melynek eredményeként egy módosított HTML-kód generálódik, és az Apache ezt a weblapot küldi ki a kliens felé.
- A PHP program függvényhívásokon keresztül éri el a MySQL szervert, és az adatbázisból lekért adatokkal építheti fel a dinamikus weblapot.

A fenti technológia részletes bemutatását a **pub/Adatbázisok/PhpMysql.ppt** tananyag tartalmazza.

9. A MySQL adatbázis-szerver

Nyílt forráskódú szoftver, letölthető a `www.mysql.com` honlapról. Gyakran alkalmazzák a PHP nyelvvel és az Apache webszerverrel együtt internetes alkalmazásoknál.

> *Történet:*
> 1979: UNIREG: belső használatra szánt adatbázis-kezelő (fejlesztője Michael Widenius, becenevén Monty) (Indexelt ISAM tárolóhelyeket kezel.)
> 1981. Monty a svéd TcX DataKonsult AB vállalatnál dolgozik.
> 1994. A TcX az UNIREG-et alkalmazza dinamikus weblapok készítéséhez, de az UNIREG-et túlságosan költségesnek találta. Ezért a Hughes Technologies által fejlesztett mSQL (a miniSQL rövidítése, fejlesztője David Hughes) adatbázis-kezelővel próbálkozott, amely azonban nem kezelte az indexeket, ezért jóval kisebb hatékonyságú volt, mint az indexelt adatstruktúrákat kezelő UNIREG.
> 1995. A TcX elkészíti MySQL 3.11-et Monty és Hughes együttműködésével, az mSQL felületének megtartásával és az UNIREG indexelési technikájának beépítésével.
> Később a TcX átalakul MySQL AB néven, a MySQL nyílt forráskódúvá válik. Becslések szerint jelenleg több mint négymillió szerveren fut.
> 2008. A Sun felvásárolja a MySQL AB-t.
> 2009. Az Oracle felvásárolja a Sun-t.

A MySQL jellemzői:

- Nyílt forráskódú, többféle platformon futtatható (pl. Win, Mac, Solaris).
- Többszálas rendszer: minden bejövő kapcsolatot (kliens folyamatot) külön szál kezel.
- Hatékonyság szempontjából az egyik legjobb rendszer.
- Kevesebb szolgáltatást nyújt, mint egyes kereskedelmi rendszerek, pl. Oracle.
- A tranzakciókezelést csak a MySQL újabb változatainál valósították meg, tranzakciók izolációs szintjeit a rendszer támogatja. A tranzakciókezelés csak akkor van jelen, ha engedélyezzük. A hatékonyságot rontja.
- SQL3-ból az objektum-relációs lehetőségeket a MySQL egyelőre nem tartalmazza.
- Alkalmazásprogramozási felület (API) a legtöbb nyelvhez, pl. C, C++, Java, PHP.
- Külső összekapcsolások támogatása.

A MySQL fontosabb segédprogramjai:

- *mysql:* SQL-alapú konzol program, kliens folyamatok vezérlésére. A begépelt parancsok több sorosak lehetnek, pontosvesszővel kell őket lezárni.
- *mysqladmin:* rendszeradminisztrációs feladatok elvégzésére.
- *mysqldump:* adattáblák definíciójának és tartalmának fájlra írása.
- *mysqlhotcopy:* futásidőben végrehajtott biztonsági mentés.
- *mysqlimport:* különféle formátumú adatok beolvasása MySQL táblákba.

A MySQL többféle adattárolási mechanizmust (storage engine) használ, ezek két fő típusba sorolhatók:

- *Tranzakciós táblák:* biztonságosabbak, rendszerösszeomlás esetén helyreállíthatók. COMMIT, ROLLBACK használható (lásd az adatbiztonságról szóló fejezetet). Hibás módosítás esetén a korábbi állapot áll helyre. Hatékonyabb párhuzamos végrehajtás.
- *Nem tranzakciós táblák:* a fenti előnyök nélkül, viszont gyorsabbak és kevesebb tárolóhelyet igényelnek.

Fontosabb tárolási típusok:
- MyISAM: gyors, és fulltext search-et támogat, nem tranzakciós.
- MERGE: több MyISAM táblát egy táblaként kezel, nem tranzakciós.
- InnoDB: tranzakciós táblatípus sorzárolással.
- BDB (Berkeley-DB): tranzakciós táblatípus lapzárolással.

A tárolási típust a CREATE TABLE utasítás TYPE paraméterében kell megadni, alapértelmezés a MyISAM.

Kliens parancsok

Belépés:

```
MYSQL –U felhasználó –P
```

A –U kapcsoló a felhasználónévre, a –P kapcsoló a jelszó bekérésére utal. (Ha ez utóbbit nem adjuk meg, akkor parancssorban kell megadni a jelszót, ami viszont ekkor látható lenne a képernyőn.) A belépés sikeres, ha utána megjelenik a `mysql>` prompt.

Kilépés:

```
QUIT
```

Adatbázisok listája:

```
SHOW DATABASES;
```

Telepítés után a rendszer – verziótól függően – például az alábbi adatbázisokat tartalmazhatja:

- *information_schema:* rendszerkatalógus (a fontosabb táblák: tables, columns, views, triggers, user_privileges, ...)
- *mysql:* a rendszer saját adminisztrációs adatbázisa (táblák: user, ...).
- *test:* üres adatbázis tesztelési célokra.

Adatbázis létrehozása:

```
CREATE DATABASE adatbázis;
```

Adatbázis megnyitása:

```
USE adatbázis;
```

Adatbázis törlése:

```
DROP DATABASE adatbázis;
```

Megnyitott adatbázis tábláinak listája:

```
SHOW TABLES;
```

Adott tábla struktúrájának lekérése:

```
SHOW COLUMNS FROM tábla;
```

Több soros SQL parancsok is beírhatók (lezárás pontosvesszővel), de ajánlatos ezeket külön TXT fájlon elkészíteni, és átirányítással végrehajtani: `<parancsfile.txt`

10. Xbase típusú rendszerek

Xbase család: az 1980-as évek elejétől különböző cégek által fejlesztett, de közös alapelvekre épülő és többé-kevésbé kompatibilis PC alapú relációs adatbáziskezelő rendszerek (RDBMS-ek): *dBase, FoxBase, FoxPro, Clipper*. Az első változatok igen egyszerűek voltak (az első PC-k lehetőségeihez igazodva), ezeket fokozatosan továbbfejlesztették az alapelvek megtartásával.

Általános jellemzők:

– *Minden adattábla külön fájlon van.* (.DBF kiterjesztés, szabványos, nyilvános adatformátum. Számos más rendszer is felismeri.)

– *Algoritmikus programnyelv,* amely – az SQL beágyazáshoz hasonlóan – tartalmazza az adatbázis-kezelő utasításokat is. Végrehajtása interpreterrel.

– *Nem SQL-alapú rendszerek,* bár az újabb változatok több-kevesebb SQL támogatást is tartalmaznak.

Az Xbase rendszerek ma már elavultnak számítanak, elsősorban azért, mert szemléletmódjuk idegen az SQL-től (pl. munkaterület, aktuális tábla fogalma). Ugyanakkor még igen sok működő alkalmazással találkozunk, ezért az alapelvek megismerése ajánlott. A továbbiakban a FoxPro parancsnyelvének alapjaival ismerkedünk meg, amelyek lényegében változatlan formában érvényesek az Xbase család valamennyi rendszerénél.

Megjegyzés. Az Xbase rendszereknél egyetlen adattáblát szoktak adatbázisnak nevezni, mi azonban továbbra is adattáblák együttesét tekintjük adatbázisnak.

10.1. A parancsnyelv alapjai

Minden parancsot új sorban kell kezdeni. Ha egy parancs nem fér ki egy sorban, a sor végén pontosvesszővel jelzendő, hogy a következő sorban folytatódik.

Speciális adattípusok, konstansok:

- *dátum:* 'mm/dd/yy' string, a CTOD() függvénnyel konvertálható dátum típusúra.
- *logikai:* .T., .F.
- *memo:* változó hosszúságú szövegmező. Tetszőleges szöveges információt tartalmazhat.

Műveleti jelek: +, -, *, /, .AND., .OR., .XOR., .NOT.

Stringek konkatenációja: +

Változónevek:

- *mezőnév:* az aktuális adattábla aktuális rekordjának "mezőnév" mezőjét jelenti.
- *táblanév->mezőnév* vagy *táblanév.mezőnév:* a "táblanév" adattábla aktuális rekordjának "mezőnév" mezőjét jelenti (például `DOLG->LAKCIM`)
- *munkaváltozó:* nem kell deklarálni, az első értékadással definiálódik a típusa. Újabb értékadáskor újradeklarálódik (például VAL='szoveg', VAL=25).
- *&változó:* a "változó" nevű karakteres változó aktuális értékét helyettesíti a parancsba (makróhelyettesítés, például USE &adat).

10.2. Relációsémák és adattáblák létrehozása, kezelése

SELECT munkaterület

Munkaterület kiválasztása. Az Xbase rendszerek legalább 10 munkaterületet biztosítanak az adattáblák kezelésére, egy munkaterületen egyszerre csak egy táblát használhatunk. Az egyes munkaterületek jelölésére az 1, 2, ..., 10 számokat, vagy az A, B, ..., J betűket, vagy a munkaterületen megnyitott tábla nevét használhatjuk. Például

```
SELECT 2
```

a 2. számú munkaterület kiválasztását jelenti. Minden további parancs a kiválasztott munkaterületre, illetve az ott megnyitott táblára (aktuális tábla) vonatkozik.

CREATE táblanév

Új relációséma (és adattábla) létrehozása. A parancs begépelése után egy ablakban megadhatjuk a tábla mezőinek nevét, típusát és hosszát. Az eljárás végén az újonnan létrehozott adattábla megnyitásra kerül az aktuális munkaterületen.

USE táblanév

Adattábla megnyitása. Ezzel egy már létező táblát (DBF file-t) nyitunk meg az aktuális munkaterületen. Műveletet végezni csak megnyitott táblán lehet. A táblanév nélküli USE parancs az aktuális munkaterületen lévő táblát lezárja.

MODIFY STRUCTURE

Relációséma módosítása. Az aktuális tábla mezőinek nevét, típusát és hosszát lehet módosítani.

BROWSE

Tábla megjelenítése "táblázat" formában, módosítási lehetőséggel.

INDEX ON kifejezés TO indexfile [UNIQUE]

Tábla indexelése. A "kifejezés" tetszőleges karakteres típusú kifejezés lehet, az indexkulcsot adja meg (általában mezőnév vagy mezőnevek konkatenációja, amelyet + jellel jelölünk). A parancs hatására a megadott nevű indexfile jön létre. A tábla a továbbiakban az index szerint rendezve jelenik meg a képernyőn, és a parancsok is eszerint kezelik. UNIQUE esetén az azonos kulcsú rekordokból csak egy példányt indexel.

Példa a Könyv tábla indexelésére:

```
INDEX ON szerző+cím TO Szercím UNIQUE
```

Az indexfile kiterjesztése és formátuma rendszerenként változik, például dBase típusú rendszereknél NDX, Fox típusú rendszereknél IDX a kiterjesztés.

SET FILTER TO feltétel

Szelekciós szűrő megadása. A továbbiakban csak a "feltétel"-nek eleget tevő rekordok érhetők el, minden kiadott parancs csak ezekre vonatkozik. Például a könyvtári adatbázis 2. változatában ha egy adott olvasó által kikölcsönzött könyveket szeretnénk áttekinteni, akkor

```
SET FILTER TO olvasószám='355'
```

parancs kiadása után BROWSE segítségével kényelmesen megtekinthetjük és módosíthatjuk az adott olvasóhoz tartozó rekordokat. A feltétel nélkül kiadott SET FILTER TO kikapcsolja a szűrőt.

SET FIELDS TO mezőnévlista

Projekciós szűrő megadása. A továbbiakban csak a felsorolt mezők jelennek meg a képernyőn. A szűrő a SET FIELDS OFF/ON paranccsal ki/bekapcsolható.

10.3. Kapcsolat táblák között, algoritmikus eszközök

A relációs adatmodell lényege, hogy több tábla között külső kulcsok segítségével kapcsolatot tud teremteni. Ennek gyakorlati használatát támogatja az alábbi parancs:

SET RELATION TO kapcsolómező INTO táblanév

Két tábla rekordmutatóinak összekapcsolása. Az aktuális munkaterületen megnyitott tábla kerül összekapcsolásra egy másik munkaterületen megnyitott "táblanév" táblával. Az aktuális tábla kell, hogy tartalmazzon egy "kapcsolómező" nevű mezőt (külső kulcs), és a "táblanév" tábla egy ennek megfelelő (gyakran azonos nevű, általában elsődleges kulcs szerepét betöltő) mező szerint kell, hogy legyen indexelve.

A parancs hatására az aktuális tábla rekordmutatójának mozgását automatikusan követi a másik tábla rekordmutatója. Pontosabban, a másik tábla rekordmutatója mindig éppen arra a rekordra áll, amelynek index értéke megegyezik az aktuális tábla aktuális rekordjának "kapcsolómező" értékével.

Tekintsük például a könyvtári adatbázis 2. változatát, vagyis a következő relációs sémákat:
Könyv (könyvszám, szerző, cím, *olvasószám*, kivétel)
Olvasó (olvasószám, név, lakcím)

Adjuk ki a következő parancssorozatot:

```
SELECT 1
USE Olvasó
INDEX ON olvasószám TO Olvind
SELECT 2
USE Könyv
SET RELATION TO olvasószám INTO Olvasó
```

Ciklusszervezés:

DO WHILE feltétel
  ciklusmag
ENDDO

Feltételes elágazás:

IF feltétel
  utasítások
[**ELSE**
  utasítások]
ENDIF

Rekord mezője értékadó utasítással nem módosítható, erre a célra az alábbi szolgál:

REPLACE mezőnév WITH kifejezés

Program (.PRG fájl) futtatása:

DO programnév

11. Adatbiztonsági mechanizmusok

Nagy adatbázis-alkalmazásoknál sok felhasználó és nagyszámú, párhuzamosan futó kliens folyamat mellett is biztosítani kell az adatbázis sértetlenségét, még esetleges üzemzavar (rendszerleállás, áramkimaradás, stb.) esetén is. Ezzel a kérdéskörrel foglalkozunk a továbbiakban.

11.1. Tranzakciós feldolgozás

Tranzakció: adatbázis-kezelő műveletek sorozata, amelyeket egy egységként célszerű kezelni, mert a részműveletek közben átmenetileg sérülhet az adatbázis integritása.

106. Példa. A Számla (számlaszám, egyenleg) táblán banki átutalás végrehajtása egyik számláról a másikra. A megfelelő beágyazott SQL program a 27. ábrán látható.

```
void átutalás()
{ EXEC SQL BEGIN DECLARE SECTION;
    int szsz1, szsz2;   // számlaszámok
    int egyenl;
    int összeg;
  EXEC SQL END DECLARE SECTION;

  EXEC SQL SELECT egyenleg INTO :egyenl FROM Számla
    WHERE számlaszám = :szsz1;

  if (egyenl >= összeg)
  { EXEC SQL UPDATE Számla
      SET egyenleg = egyenleg - :összeg
      WHERE számlaszám = :szsz1;
    EXEC SQL UPDATE Számla
      SET egyenleg = egyenleg + :összeg
      WHERE számlaszám = :szsz2;
  }
  else printf("Nincs fedezet!");
}
```

27. ábra. Banki átutalást végrehajtó program

Probléma: Ha hardver vagy szoftver hiba miatt egy tranzakció végrehajtása közben a DBMS leáll (rendszerösszeomlás), és ez a fenti példában a két UPDATE utasítás között következik be, akkor az átutalt összeg elvész.

Megoldás: Biztosítani kell, hogy vagy végrehajtódjon a tranzakció valamennyi utasítása, vagy egyik se hajtódjon végre. Rendszerösszeomlás esetén ez azt jelenti, hogy a rendszer újraindításakor a félkész tranzakciók visszavonásra kerülnek.

Tranzakciós feldolgozást támogató SQL utasítások:

COMMIT;

Tranzakció lezárása, az eddig kiadott SQL parancsok hatásának véglegesítése. COMMIT előtt a változások még visszafordíthatók.

Általában két COMMIT között kiadott SQL parancsok sorozatát tekintjük tranzakciónak. A fenti átutalás() függvényt úgy alakíthatjuk tranzakcióvá, hogy az elejére és végére

```
EXEC SQL COMMIT;
```

utasítást írunk.

SAVEPOINT azonosító;
Tranzakción belüli pontot azonosít (címke jellegű funkció).

ROLLBACK [TO savepoint];
Változások visszapörgetése a tranzakció elejéig, vagy a tranzakción belül megadott „savepoint”-ig. A ROLLBACK műveletet alkalmazhatja a rendszer (pl. újraindításkor), de élhet vele a programozó is, ha adott szituációban vissza kívánja vonni a tranzakciót.

11.2. Párhuzamos hozzáférések

Kliens-szerver modellben, ha a tranzakciók párhuzamosan időosztásban futnak, akkor egymást megzavarhatják, ha egyik vagy mindkettő módosítja az adatbázist. A 27. ábra szerinti program esetén, ha az első számla egyenlege 100 000 Ft, és ebből ketten egyszerre kívánnak átutalni 80 000 Ft-ot, akkor a fedezetellenőrzés dacára az egyenleg negatív lesz (28. ábra).

1. folyamat: sz1 fedezet ellenőrzés OK.
2. folyamat: sz1 fedezet ellenőrzés OK.
1. folyamat: sz1: egyenleg := egyenleg – 80000 (új egyenleg: 20000)
2. folyamat: sz1: egyenleg := egyenleg – 80000 (új egyenleg: –60000)
1. folyamat: sz2: egyenleg := egyenleg + 80000
2. folyamat: sz3: egyenleg := egyenleg + 80000

28. ábra. Hibás fedezetellenőrzés időosztással összefésülődő sz1 → sz2 és sz1 → sz3 számlák közötti átutalások esetén

Zárolás

A fenti jellegű problémákra a megoldás: az *adatok zárolása (locking)*, vagyis az adatok elérhetőségének korlátozása más tranzakciók részére. A zárolás általában tranzakció közben jön létre, és a tranzakció végéig (COMMIT vagy ROLLBACK végrehajtásáig) tart.

Zárolási szintek:
1. A teljes adatbázis zárolása. Az előbb induló tranzakció zárolja az egész adatbázist mindaddig, amíg véget nem ér. Ekkor a második tranzakció el sem tud indulni az első befejezése előtt. Ez tulajdonképpen azt jelenti, hogy nem engedünk meg párhuzamos hozzáféréseket, amely nagy adatbázis és sok egyidejű kliens folyamat esetén elfogadhatatlan.
2. Tábla zárolása: a tranzakció csak azt a táblát zárolja, amellyel dolgozik. Ez már sok esetben megfelelő lehet, de a banki átutalás esetén a teljes Számla tábla (összes folyószámla) zárolása még mindig elfogadhatatlan.

3. Sor szintű zárolás. Nem a teljes táblát, hanem csak a művelet által érintett sor(oka)t zároljuk.

Zárolási módok. A DBMS-ek sokféle zárolási módot alkalmaznak, a két legfontosabb:

- *Megosztott (shared) zár:* lényegében olvasási jogot ad a zároló tranzakciónak. Egy objektumra (táblára vagy sorra) egyszerre több megosztott zár lehet érvényben.

- *Kizárólagos (exclusive) zár:* módosítást is lehetővé tesz. Egy objektumra egyszerre csak egy kizárólagos zár lehet érvényben, és mellette semmilyen más zár nem megengedett.

A zárolás implicit vagy explicit formában történhet.

Implicit zárolás: A DBMS adateléréskor általában automatikus zárolást hajt végre, például minden INSERT, UPDATE, DELETE utasítás végrehajtásakor az érintett objektumokon zárolás történik.

Explicit zárolás: Egyes DBMS-ek SQL utasításokat biztosítanak felhasználói zároláshoz.

> Tábla zárolása (Oracle):
> **LOCK TABLE táblanév IN zárolásimód MODE [NOWAIT];**
> Ha az utasítás végrehajtásakor a táblát más tranzakció már zárolta, akkor az utasítás várakozik a zárolás feloldásáig. Viszont, ha az utasítás végére a NOWAIT opciót illesztjük, akkor a DBMS egy üzenet kíséretében azonnal visszaadja a vezérlést. Példa:
> `LOCK TABLE dolgozó IN EXCLUSIVE MODE NOWAIT;`
>
> Sor szintű zároláshoz a SELECT végére
> **FOR UPDATE [OF oszlopok]**
> írandó, ekkor az utasítás zárolja a SELECT által kiválasztott sorokat. Példa:
> `SELECT * FROM Dolgozó WHERE osztálykód='A11' FOR UPDATE;`

A zárolások miatt *holtpont (deadlock)* léphet fel, vagyis párhuzamos folyamatok egymásra várhatnak, például:

1. tranzakció: T tábla zárolása, S tábla zárolása, COMMIT
2. tranzakció: S tábla zárolása, T tábla zárolása, COMMIT
----------> idő

Ebben a példában az 1. tranzakció először zárolja a T táblát, egyidejűleg a 2. tranzakció az S táblát. Ezután az 1. tranzakció S-et, a 2. tranzakció T-t zárolná, de kölcsönösen egymásra várnak, és sohasem jutnak el a tranzakció végéig, amely a zárolás feloldását jelentené.

A DBMS általában nem tudja megakadályozni a holtpontot, de észleli azt, és ilyenkor visszapörgeti az előidéző tranzakciókat, majd kis időkülönbséggel újra elindítja azokat.

Izolációs szintek

Párhuzamosan futó tranzakciók esetén az alábbi anomáliák léphetnek fel:

a) Kétes adat olvasása (dirty read): más tranzakció által módosított, de még nem véglegesített adat olvasása. Ez akkor okoz gondot, ha a másik tranzakció valamiért visszavonásra kerül, és így hibás adatot olvastunk (lásd az alábbi példát).

b) Változó adat olvasása (nonrepeatable read): a tranzakció újraolvas egy adatot, amelyet közben más (véglegesített) tranzakció módosított vagy törölt, így a két olvasás eredménye eltér egymástól.

c) fantom adat olvasása (phantom read): a tranzakció újraolvas egy táblát, amelybe közben más (véglegesített) tranzakció új sorokat szúrt be.

A fenti anomáliák kiszűrése SQL-ben a tranzakció izolációs szintjének megadásával lehetséges:

SET TRANSACTION [elérés] [ISOLATION LEVEL izoláció];

Az utasítás a tranzakció elején adható ki. Az *elérés* paraméter lehetséges értékei:
- READ ONLY: a tranzakció csak olvassa az adatbázist.
- READ WRITE: a tranzakció olvassa és írja is az adatbázist.

Az *izoláció* paraméter lehetséges értékei:
- READ UNCOMMITTED: kétes adat olvasása engedélyezett. Ekkor az *a), b), c)* anomáliák egyaránt felléphetnek.
- READ COMMITTED: kétes adat olvasása nem engedélyezett. Itt csak a *b), c)* anomáliák léphetnek fel.
- REPEATABLE READ: kétes adat olvasása nem engedélyezett, és az olvasott adatokat más folyamat nem módosíthatja a tranzakció végéig. Itt csak a *c)* anomália fordulhat elő.
- SERIALIZABLE: sorosítható tranzakció, vagyis párhuzamos végrehajtása egyenértékű kell, hogy legyen a sorban egymás utáni végrehajtással. Itt egyik anomália sem léphet fel.

Alapértelmezés: SERIALIZABLE.

Minél magasabb szintű izolációt alkalmazunk, annál nagyobb az adatbiztonság, de csökken a párhuzamosítás lehetősége.

Megjegyzések:
- A READ ONLY tranzakciók korlátlanul párhuzamosíthatók egymással.
- A READ WRITE + READ UNCOMMITTED tranzakciók a legveszélyesebbek (ezért READ UNCOMMITTED esetén az alapértelmezés READ ONLY).

A 28. ábra szerinti anomália megszüntethető, ha a tranzakció elején kiadjuk a

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

utasítást, amely egyébként az SQL2-ben alapértelmezés.

107. Példa. Repülőgépre helyfoglalás: a program először lefoglalja az első szabad helyet, majd megkérdezi az ügyfelet, hogy elfogadja-e azt. Ha igen, akkor véglegesít (COMMIT), ha nem akkor visszavon (ROLLBACK).

Ebben a példában kétes adat olvasása történhet a következőképpen: a T1 tranzakció ideiglenesen lefoglalja például az 52. számú helyet. A párhuzamosan futó tranzakció már az 52-es helyet foglaltnak érzékeli, ezért csak más helyet tud foglalni (ha van). Ugyanakkor - az ügyfél visszautasítása miatt - T1 később felszabadítja az 52-es helyet, de azt T2 mégsem tudta lefoglalni. Ha úgy döntünk, hogy a leírt anomália nem jelent komolyabb veszélyt és várhatóan igen ritkán fog fellépni, akkor

SET TRANSACTION READ WRITE ISOLATION READ UNCOMMITTED;

kiadásával gyorsíthatjuk a párhuzamos tranzakciók feldolgozását.

11.3. Jogosultságok

Minden adatbáziselemnek van tulajdonosa, éspedig az a felhasználó, aki létrehozta. A tulajdonos minden joggal rendelkezik az adott elem felett.

Jogosultság adományozása SQL-ben:

GRANT jogosultságok ON adatbáziselemek TO felhasználók
[WITH GRANT OPTION];

Jogosultság:
- SELECT: lekérdezés engedélyezése.
- ALTER: struktúramódosítás engedélyezése (ALTER TABLE).
- INSERT[(oszlopok)], UPDATE[(oszlopok)], DELETE: tábla módosítás engedélyezése a megfelelő utasítással. Oszlopok megadása esetén az engedély csak az adott oszlopokra vonatkozik.
- REFERENCES: külső kulcs hivatkozás engedélyezése az adatbáziselemre,
- ALL PRIVILEGES: az összes adományozható jogosultság.

Adatbáziselem: amelyre a jogosultságot adományozzuk.
Felhasználó: akinek a jogosultságot adományozzuk.
WITH GRANT OPTION: továbbadományozási jog adása.

Engedélyezési diagram: csomópontjai jogosultságot, élei adományozást jelentenek.
- csomópont: adott F felhasználónak adott A adatbáziselemre vonatkozó adott J jogosultsága (F,A,J).
- él: $(F_1,A_1,J_1) \rightarrow (F_2,A_2,J_2)$ azt fejezi ki, hogy F_1 felhasználó az A_1 elemre érvényes J_1 jogosultsága alapján F_2-nek az A_2 elemre J_2 jogot adományozott.

Jogosultság visszavonása:

REVOKE jogosultságok ON adatbáziselemek FROM felhasználó [CASCADE];

CASCADE: a visszavont jogosultság alapján továbbadományozott jogosultságok is visszavonásra kerülnek - feltéve, hogy ugyanazt a jogot az illető más forrásból nem szerezte meg.

Egy SQL utasítást csak akkor hajt végre a rendszer, ha a felhasználó a végrehajtáshoz szükséges valamennyi jogosultsággal rendelkezik.

Példák:

```
GRANT SELECT ON Dolgozó TO Kovács, Tóth;
GRANT UPDATE(lakcím) ON Dolg1 TO Horváth WITH GRANT OPTION;
REVOKE SELECT ON Dolgozó FROM Tóth;
```

Irodalom

Az Object Data Management Group honlapja. http://www.odmg.org

Gazsó Zoltán: *Adatbáziskezelés FoxPro-ban (2.5, 2.6 DOS, Windows)*. ComputerBooks, Budapest, 1995.

Gruber M.: *SQL A-Z.* Kiskapu kiadó, 2003.

Gulutzan P., Pelzer T.: *SQL teljesítményfokozás.* Kiskapu kiadó, 2003.

Hernandez, M. J.: *Adatbázis-tervezés.* Kiskapu kiadó, 2004.

Kende Mária, Kotsis Domokos, Nagy István: *Adatbázis-kezelés Oracle-rendszerben.* Panem, Budapest, 2002.

László József: *Dinamikus weboldalak, CGI programozás Windows és Linux rendszereken.* ComputerBooks, Budapest, 2002.

Ramakrishnan R., Gehrke J.: *Database Management Systems.* McGraw-Hill, 2000.

Reese G, Yarger R. J., King T.: *A MySQL kezelése és használata.* Kossuth Kiadó, 2003.

Ullman J. D., Widom J.: *Adatbázis rendszerek – Alapvetés.* Második, átdolgozott kiadás, Panem, 2008.