Budapesti Műszaki és Gazdaságtudományi Egyetem
Automatizálási és Alkalmazott Informatikai Tanszék

INFORMATIKA 2

ADATBÁZISOK

Iváncsy Szabolcs és Vajk István

2007 Október

Tartalomjegyzék

Ábrák jegyzéke **iv**

Táblázatok jegyzéke **v**

1. Fejezet
Adatbázisok **1**
1.1. Alapfogalmak 1
1.1.1. A programozó és a felhasználó kapcsolata a rendszerrel . . . 2
1.1.2. Járulékos feladatok 3
1.1.3. Személyek 6
1.1.4. Az absztrakciós szintek 8
1.2. Koncepciós adatmodellezés 9
1.2.1. Az adatbázis sémája és tartalma 9
1.2.2. Adatfüggetlenség 9
1.2.3. Entitások 9
1.2.4. Attribútumok 10
1.2.5. Kapcsolatok 11
1.3. Fizikai adatszervezés 13
1.3.1. Heap szervezés 13
1.3.2. Hash állományok 16
1.3.3. Indexelt állományok 18
1.3.4. B* - fák, mint többszintes ritka indexek 19
1.3.5. Másodlagos indexek, invertálás 21
1.4. Adatmodellek 22
1.4.1. A hierarchikus adatmodell 22
1.4.2. A hálós adatmodell 22
1.4.3. Relációs adatmodell 24
1.5. Relációs adatbázis tervezése 32
1.5.1. Tervezés ER diagrammal 32
1.5.2. Tervezés sémadekompozícióval 36

1.5.3. Funkcionális függőség 38
1.5.4. Relációk felbontása 46
1.5.5. A relációk normal formái 50
1.5.6. Sémadekompozíciók 52
1.6. Az SQL nyelv 56
1.6.1. Bevezetés 56
1.6.2. Objektumok létrehozása, törlése, módosítása 58
1.6.3. Lekérdezések 62
1.6.4. Nézet és index létrehozása 84
1.6.5. Adatelérések szabályozása 86
1.6.6. Tranzakciók 88
1.7. Bővítések 92
1.7.1. Konzisztencia feltételek 92
1.8. Procedurális adatbázis kezelés 95
1.8.1. A PL/SQL nyelv 95
1.8.2. A Transact-SQL nyelv 97
1.9. Beépített függvények, adatok kezelése 99
1.9.1. Számok kezelése 99
1.9.2. Szövegek kezelése 101
1.9.3. Dátumok kezelése 102
1.9.4. Konverziós függvények 103
1.10. Példák az SQL-re 107
1.10.1. A cukrász adatbázis 107
1.11. Függelék 118
1.11.1. Minta táblák 118
1.11.2. A számok formátuma 119
1.11.3. Dátum formátumok 119

Ábrák jegyzéke

1.1. Adatbáziskezelő 3
1.2. Kapcsolatok egyed-kapcsolati diagramja 13
1.3. Attribútumok és az egyed-kapcsolati diagram 14
1.4. A blokk és a rekord felépítése 14
1.5. Az owner és a member rekordok megjelenítése 23
1.6. Hálós modell 24
1.7. Az egy-egy kapcsolat átalakítása 32
1.8. A egy-több kapcsolat átalakítása 34
1.9. A több-több kapcsolat átalakítása 35
1.10. A cukrász adatbázis sémája 107

Táblázatok jegyzéke

1. fejezet

Adatbázisok

1.1. Alapfogalmak

Adatbázisnak a valós világ egy részének leírásához használt adatok összefüggő, rendezett, tárolt halmazát nevezzük. Ezeket az adatokat számítógépeken tároljuk. Nem tekinthető adatbázisnak a kartotékrendszerben nyilvántartott adatok összessége, ugyanis ezeken a később részletezett műveletek nem, vagy csak nehezen végezhetők el. Ennek megfelelően alapvetően a számítógépes adatbázist tekintjük adatbázisnak.

Azt a szoftver rendszert, amellyel az adatbázis adataihoz hozzáférhetünk, adatbáziskezelő-rendszernek (database management system, DBMS) nevezzük. Az adatokhoz való hozzáférés lehet az adatok létrehozása, módosítása, törlése vagy olvasása.

Az adatbáziskezelő feladata, hogy az adatbázis felhasználójának biztosítsa a tárolt adatok kezelését. Az adatbáziskezelőn keresztül a felhasználó annélkül tudja elvégezni a teendőket, hogy az adatbáziskezelő algoritmusait, vagy az adatok tárolási elvét ismerné. Ennek megfelelően az adatbáziskezelő nagyon egyszerű parancsok segítségével kapja meg az elvégzendő feladatokat, amely lehetővé teszi az adatok kezelését.

Az adatbázis kezelés bizonyos összetartozó adatoknak a tárolását, naprakész karbantartását, ezekből a szükséges adatoknak a gyors és megbízható kigyűjtését jelenti. Az adatbázis kezelés az a terület, amelyre a számítógépet talán leggyakrabban használjuk. Az adatbázis tervezés pedig az előbb említett adatok logikai

tárolási szerkezetének maghatározását, fizikai tárolási módjának megadását, az adatokon elvégezhető műveletek megtervezését jelenti.

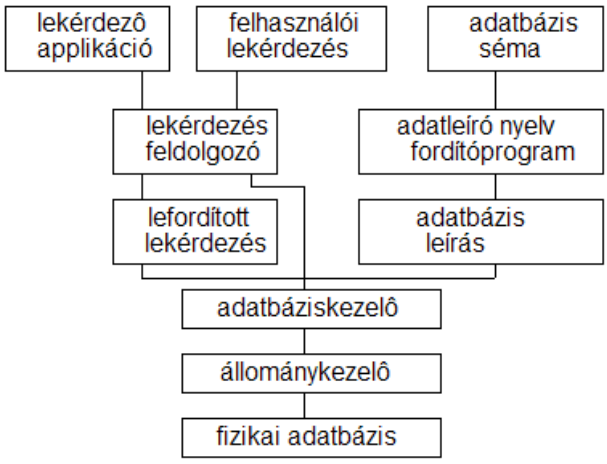
A továbbiakban azokat az általános ismérveket fogjuk megvizsgálni, amelyek jellemzőek az adatbázisokra és adatbáziskezelő-rendszerekre. Az itt tárgyalt jellemzők, feladatok, megoldások jelentkeznek általában az adatbázis tervezés, kezelés folyamatában. Az egyes felhasználási területek és a felhasználók a saját igényeiknek megfelelően további igényeket támaszthatnak e területtel szemben, ugyanakkor a korszerű adatbáziskezelő-rendszerek egyre több speciális igény kielégítésére is alkalmasak. Éppen ezért a forgalomban lévő adatbáziskezelő-rendszerek az igényeknek és a piaci versenynek megfelelően dinamikusan fejlődnek, állandóan korszerűsítik őket.

1.1.1. A programozó és a felhasználó kapcsolata a rendszerrel

Az 1.1 ábra egyesítve mutatja a DBMS leggyakoribb felépítését és környezetét. Előfordul, hogy az adatbáziskezelő-rendszert közvetlenül egy adott alkalmazás megvalósítására írják. Ebben az esetben a parancsfeldolgozó modul információval rendelkezhet az adatbázisról. Ma a rendszereket legtöbbször a kereskedelemben kapható általános célú modulokból építik fel. Itt beépített információról nem lehet szó, tehát kulcsfontosságú, hogy legyen alkalmas leíró formalizmusuk. Erre szolgálnak az adatleíró nyelvek (data definition language, DDL).

Az 1.1 ábrán látható, hogy az adatbáziskezelő használatának két fő vonulata van. Az első a lekérdezések, amely a felhasználótól közvetlenül származhat, vagy felhasználói applikáció állítja elő őket. A lekérdezéseket lefordítva átadjuk az adatbáziskezelő modulnak. Ennek feladata, hogy az állománykezelő által értelmezhető parancsokká alakítsa azokat. A másik ág az adatbázis létrehozása, módosítása. Itt az adatbázis megtervezett sémái alapján elkészül az adatbázis leírása, amely azután alkalmas az adatok tárolására.

Az állománykezelő (file manager) biztosítja a hozzáférést a fizikai adatbázishoz. Az állománykezelő egyszerűbb esetben része lehet az alkalmazott operációs rendszernek, de ennél többet is elvárhatunk tőle, ha figyelembe vesszük a későbbiekben megismerendő speciális állományszerkezeteket.



1.1. ábra. Adatbáziskezelő

1.1.2. Járulékos feladatok

Az adatbáziskezelőtől és környezetétől az adatbázis kezelésén túlmenően néhány egyéb, kiegészítő feladat megoldását is elvárjuk.

1.1.2.1. Megbízható tárolás

Gondoskodni kell arról, hogy a tárolt adatok ne sérülhessenek meg, ne veszhessenek el. Ezt az adattárolásban beépített redundanciával, vagy az adatok többszörös tárolásával érhetjük el.
Egyszerűbb esetekben az adatok időszakos mentésével biztosítják, hogy meghibásodás esetén a régebbi, még hibátlan adatok rendelkezésre álljanak és visszaállíthatók legyenek. A mentés óta elvégzett műveletek újbóli végrehajtásával - amennyiben ezeket tároltuk - a sérülés előtti állapot viszszaállítható.

Drágább megoldás, hogy minden adatot a keletkezés pillanatától kezdve folyamatosan többször tárolunk. Pl. 2-szeres tárolás esetén dupla tárolókapacitás szükséges. Csökkentheti a ráfordított többlet tárolási kapacitást ha pl. egy 8 bites adatot kiegészítünk egy paritásbittel, ami csak 9/8-szoros kapacitásnövekedést eredményez, és minden bitet külön fizikai tárolón tárolunk, ami csak igen nagy adatbázisok esetén gazdaságos. Ebben az esetben, ha egy fizikai adathordozó tönkremegy, akkor a maradék 8 adathordozó adatai alapján (a paritás és a meghibásodott tároló bitpozíciója alapján) az adatok visszaállíthatók.

1.1.2.2. Adatvédelem

Az illetéktelen felhasználók elleni védelmet értjük alatta. Nem minden felhasználó férhet hozzá minden tárolt adathoz. Ennek biztosítására különböző megoldások állhatnak rendelkezésre.

Legbiztonságosabb megoldás, hogy a védett adatokat külön számítógépen, olyan teremben tárolják, ahová csak az illetékesek léphetnek be. (pl. katonai adatbázis esetén az ajtóban még akár fegyveres őr is lehet.)

Egyszerűbb esetekben az adatok hozzáférését hardveres védelemmel biztosítják, csak az úgynevezett hardverkulcs (és megfelelő jelszó) alkalmazásával lehet az adatokhoz hozzáférni.

Hálózatba kapcsolt adatbáziskezelők esetében a hardverkulcs, jelszó szerepét a mágneskártya és a pin kód veszi át (pl. banki pénzfelvevő automata).

Sokszor azonban elegendő a szoftveres védelem, amikor egyszerűen egy jelszóhoz kötik a hozzáférés jogát. Ilyenkor a belépéskor, vagy bizonyos feladatok elvégzéséhez a program bekéri a felhasználó nevét, és a hozzá tartozó jelszót, és csak helyes név-jelszó együttes esetén lehet hozzáférni az adatbázishoz, vagy elvégezni a kívánt műveletet. A jelszavas adatvédelemnél a kezelő szoftver egy nem invertálható algoritmussal kódolja a jelszót, és ezt a kódolt értéket ellenőrzi. Ezért, ha valaki hozzá fér is a fájlokhoz, nem tudja a jelszót kiolvasni, csak szoftveres visszafejtésre van lehetősége, aminek az időigénye nagy. Pl.: egy 10 karakteres jelszó esetén $62^{10} \approx 8 * 10^{17}$ a lehetséges jelszavak száma (kisbetűk, nagybetűk és számok feltételezésével). Ha másodpercenként 10^6 jelszót tudunk leellenőrizni (μs-onként 1-et), akkor a biztos visszafejtés a nyers erő módszerével $\approx 8 * 10^{11} s \approx 2.5 * 10^4$ év, ami jó visszafejtő algoritmusok esetén is akár több év is lehet.

A hálózaton átküldött adatok védelmére különböző szintű kódolásokat, titkosítást alkalmaznak.

Bizonyos esetekben magukat az adatokat is titkositottan tárolják.

1.1.2.3. Integritás

Az adatok integritása azt jelenti, hogy az adatoknak valamilyen előre meghatározott kritériumnak meg kell felelniük, az adatoknak egymással összhangban kell

lenniük.

- *Formai vagy tartományi integritás*:
  A legegyszerűbb a formai vagy tartományi integritás, ami azt jelenti, hogy az adott adatnak egy előre meghatározott tartományból, halmazból kell értéket kapnia. Könnyen kideríthető hiba, ha pl. valakinek a cipőmérete negativ, vagy a nevében szám is található. Ilyen esetben az ellenőrzés elég egyszerű. Első esetben meg kell nézni, hogy egy tartományból választottuk e ki az értéket (pl. 15 és 65 között), míg a második esetben az egyes karakterek csak betűk lehetnek, amit szintén egyszerű ellenőrizni. Előfordul, hogy az értékek csak algoritmikusan ellenőrizhetők, pl. a személyi számára nem teljesül az ellenőrző összeg, de ez már nem tartozik a formai integritáshoz.
- *Referenciális integritás*:
  Kissé bonyolultabb, ha egy mező értéke egy másik mezővel kell, hogy összhangban legyen. Ez a referenciális integritás. Pl. ha valakinek a lakhelye Budapest, az irányítószámnak 1-gyel kell kezdődnie, vagy egy nőnek a személyi száma páros számmal kell, hogy kezdődjön, vagy pl.: egy raktárban a készlet maximum(ami elfér benne) legyen nagyobb, mint a készlet minimum (amikor a raktárba új készletet kell beszerezni). Ennek ellenőrzése szintén könnyen megoldható.
- *Strukturális integritás*:
  Az előbbieknél sokkal bonyolultabb a strukturális integritás. Ezalatt azt kell értenünk és ellenőriznünk, hogy nem sérült-e meg valamely feltételezés, amelyre az adatbázis szerkezetének megtervezésekor építettünk. Pl. egy iskolai adatbázisban feltételezzük, hogy egy osztályban nincs 2 azonos nevű tanuló. Ha az iskolába jön 2 azonos nevű tanuló, azt különböző osztályba tesszük. Van 4 osztályunk, mi történik, ha 5 azonos nevű tanulónk van?

  Leggyakrabban előforduló ilyen hiba az egyértelműség, azaz a tervezéskor valamely feltételre nem is gondolunk, ezt egyértelműnek tételezzük fel. Pl. egyértelműnek tekintjük, nem mohamedán országokban, hogy egy férfinak egy felesége van, vagy pl. egy iskolának egy igazgatója van. Ilyenkor 2 feleség vagy 2 igazgató bejegyzése sérti az integritást, de az adatbáziskezelő-rendszer nem tudja ellenőrizni a hibát.

Az adatbáziskezelő-rendszernek kell, hogy legyen olyan beépített szolgáltatása, amely segítségével az adatbázis integritása ellenőrizhető, mivel a különböző funkciók végrahajtásakor integritási problémák előfordulhatnak.

1.1.2.4. Konzisztencia

Az adatbáziskezelő rendszerek általában többfelhasználósak, és gyakran számítógéphálózaton üzemelnek. Nagyon fontos, hogy az azonos adatokon közel egyidőben műveleteket végző felhasználók tevékenységeinek ne legyenek nemkívánatos mellékhatásai az adatbázis tartalmára.

Pl. egy raktárkészletben van reggel 100 db porszívó. Két ügynök napközben elad 10 ill. 12 porszívót. Este elvégzik az adatbázis módosítását. Ha egymás után végzik el a módosítást, az első beolvassa, hogy a raktáron van 100, levonja az eladott 10-et, visszaír 90-et. Ezután a második beolvassa, hogy van 90, levonja az eladott 12-t, visszaír 78-at, ami helyes. Amennyiben az adatbázis módosítását közel egy időben végzik (ami itt azt jelenti, hogy mindketten beolvassák a készletet, mielőtt a másik még módosította volna) mindkettő beolvassa hogy a készlet 100, az első 100-ból levonja a 10-et, marad 90, a második szintén a 100-ból levonja a 12-t, marad 88. Ezután visszírják a kiszámolt eredményt. Aki visszaíráskor gyorsabb, annak az eredményét a másik felülírja, és igy a végeredmény 90 vagy 88 lesz, de egyik sem helyes. Az adatbáziskezelőnek kell biztosítania, hogy ez utóbbi eset ne fordulhasson elő, és ne csak az egyik változtatás legyen bejegyezve. Ezt a problémát a *tranzakció kezeléssel* (lásd a 1.6.6 fejezetet) lehet kiküszöbölni.

Az adatmódosításokat úgy kell végrehajtani, hogy az adatbázist konzisztens állapotból konzisztens állapotba vigye.

1.1.3. Személyek

Az adatbázissal kapcsolatba kerülő személyeket tevékenységük illetve jogosultságaik szerint csoportokra osztjuk.

1.1.3.1. Képzetlen, vagy külső felhasználó

A felhasználók legszélesebb csoportja, akik csak bizonyos ismeretekkel rendelkeznek a rendszerről (pl. vasuti társaság alkalmazottja, amikor helyet foglal egy vonatra),

vagy még ennyivel sem, csak a rendszer útmutatásainak a segítségével tudnak dolgozni, (pl. egy áruházi katalógus lapozgatója), vagy nincs az adott adatbázishoz magasabb szintű jogosultságuk, igy csak a megengedett (pl. katalógus lapozgatási) műveleteket hajthatják végre. Ezek az adatbázishoz csak már előre megírt programokon keresztül csatlakozhatnak, ahol ezek a programok biztosítják a felhasználó és az adatbázis között a megengedett tevékenységeket, segítik a felhasználót a megengedett műveletek elvégzésében.

1.1.3.2. Applikáció programozó

Applikáció programozó az a szakember, aki az előbbi, felhasználó által látott programot készíti és karbantartja. Az applikáció programozónak ismernie kell azt a nyelvet, amely lehetővé teszi az adatbázisban tárolt adatok elérését, valamint ismernie kell egy olyan programozási nyelvet, amely segítségével egy olyan felületet ad a felhasználónak, amely alkalmazásával a felhasználó különböző feldolgozási feladatokat meg tud oldani. Ezek olyan feladatok, amelyek programozót igényelnek, az illetőnek az adatbázis belső szerkezetét ismernie kell, de nem szükséges, hogy az adatbázis tartalmát ismerje, a tartalmát vagy a szerkezetét módosítani tudja.

1.1.3.3. Adatbázis adminisztrátor

Adatbázis adminisztrátornak nevezzük azt a személyt, aki az adatbázis felett gyakorlatilag korlátlan jogokkal rendelkezik. Vannak olyan kitüntetett tevékenységek, amelyeket kizárólag ő végezhet el az adatbázison, vagy az elvégzéséhez az ő engedélye szükséges. Ilyen feladatok:

- *Generálás*: Az adatbázis szerkezetének kijelölése, és annak meghatározása, hogy milyen állomány-szerkezetben tároljuk az adatokat.
- *Szerkezetmódosítás*: Az adatbázis eredeti szerkezetének módosítása. Ez feltételezi azt az alapvető igényt, hogy egyetlen adat se semmisüljön vagy sérüljön meg azért, mert az összetartozó adatok mellé újabbakat is felveszünk a tárolandók közé.
- *Jogosultságok karbantartása*: A hozzáférések jogának naprakészen tartása, módosításai.

- *Mentés-visszaállítás*: Biztonsági okokból időszakonként másolatot célszerű készíteni az adatbázisról. Ha az adatbázis megsérül, ez a másolat teszi lehetővé a visszaállítást a mentés időpontjának állapotába. A mentést alkalmas célprogram felhasználásával bárki elvégezheti (akinek joga van hozzá), de a visszaállítás nagy körültekintést igénylő feladat.

1.1.4. Az absztrakciós szintek

Az adatbázisban lévő tárolt adatokhoz különböző módon férhetünk hozzá. Ezeket a hozzáférési módokat absztrakciós szinteknek is nevezzük.

1.1.4.1. Fizikai adatbázis

Fizikai adatbázison azt értjük, hogyan helyezkednek el az adatbázis adatai a fizikai tárolókon. Ide érthetjük a fizikailag megvalósított szerkezetet is (lásd a 1.3. fejezetet).

1.1.4.2. Fogalmi (logikai) adatbázis

Fogalmi adatbázis voltaképpen az a modell, ahogyan az adatbázis tükrözi a való világot (koncepcionális sémának is nevezzük). Azt határozza meg, hogy melyik adatot hogyan kell értelmezni, milyen az adatbázis struktúrája. (Pl. egy adat értéke 35, de mi ez az adat, cipőméret, házszám?)

1.1.4.3. Nézet (view)

Nézet az, amit egy-egy felhasználó az adatbázisból lát. Ha az adatbázisnak több felhasználási lehetősége van, ezek mindegyikéhez külön nézet tartozhat. Ez lehet a felhasználók jogosítványaihoz kötött is. (pl. a légitársaság egységes nyilvántartásából más adatok érdekesek, ha a pilóták fizetését szeretnénk módosítani, és az adatok másik körére van szükségünk, ha egy gép utasainak listáját akarjuk megtekinteni.)

1.2. Koncepciós adatmodellezés

A következő fejezetben az adatbázissal kapcsolatos alapvető fogalmakat tekintjük át, beszéljük meg ezek értelmezését, használatát.

1.2.1. Az adatbázis sémája és tartalma

Az adatbzis tervezéskor egy modell felállítása és az adatbázis felépítése a célunk. Ilyenkor tevezzük meg, hogy milyen adatokat akarunk tárolni, ezeket mire fogjuk használni, milyen összefüggések vannak az egyes adatok között. Erre szolgálnak az adatbázis tárolandó adatainak struktúráját leíró sémák.

Amikor az adatbázist használjuk, a sémáknak megfelelő tartalommal, azaz a tárolt információval van dolgunk. Először konkrét adatokkal feltöltjük az adatbázist, majd ezeket használjuk, lekérdezzük, módosítjuk. Az adatbázis sémák konkrét adatait eseteknek, példányoknak nevezzük.

1.2.2. Adatfüggetlenség

Az adatfüggetlenség azt jelenti, hogy az adatok tárolásának szervezésében ne használjuk olyan megoldásokat, amelyek az adatbázis sémáinak megváltoztatásakor nem biztosítják az adatok változatlanságát. (Vagyis az adatoknak a sémák változtatásaitól kell függetlennek lenniük.) Ennek megfelelően kétféle adatfüggetleségről szokás beszélni.

- *Fizikai adatfüggetlenségen* azt értjük, hogy a fizikai működés sémáiban végbezvitt változtatások ne érintsék a fogalmi (logikai) adatbázist.
- *Logikai adatfüggetlenségről* akkor beszélünk, ha a logikai adatbázis sémáinak megváltozása nem jár az egyes felhasználásokhoz, felhasználókhoz tartozó nézetek megváltozásával.

1.2.3. Entitások

Entitásnak (egyednek) nevezünk mindent, ami létezik és megkülönböztethető, amiről adatokat tárolunk az adatbázisban. Az entitás az adatbázis szempontjából értelmezett. Pl. egy település nyilvántartó rendszerben a város az entitás, hiszen

erről tartanak nyilván különböző adatokat, azonban egy személyi adatok nyilvántartó rendszerében egy város, mint a személy lakóhelye, csak jellemző. Másik példa lehet, hogy két tyúktojás általában nem különböztethető meg, és pl. egy üzlet adatbázisában legfeljebb a darabszámuk szerepel. Ugyanakkor egy biológiai vizsgálatkor, amikor a kiscsirke kikelésének különböző feltételeit vizsgálják, az egyes tojásokat megkülönböztethetővé kell tenni (megjelölik, vagy a keltetőbeli pozíciója alapján különböztetik meg), hiszen ezekről külön-külön gyüjtünk és tárolunk adatokat.

1.2.4. Attribútumok

Az entitásoknak jellemzőik vannak. Ezeket attribútumoknak vagy tulajdonságoknak nevezzük. Alapvetően az egyes entitásokról ezeket az adatokat fogjuk tárolni. Az attribútumok szolgálnak az entitások tulajdonságainak tárolására, azaz ezek írják le azt, hogy mit szeretnénk tárolni. Önmagukban nem álhatnak, csak egyedhez vagy kapcsolathoz hozzárendelve.

Megkülönböztetünk egyszerű és összetett attribútumokat, valamint egyértékű és többértékű attribútumokat:

- *Egyszerű attribútum*: Egyszerű attribútum egy elemi információ Pl.: Vezetéknév, szeme színe,...
- *Összetett attribútum*: Olyan attribútum, amely több rész-attribútumból áll Pl.: Lakcím, anyja neve, ...
- *Egyértékű attribútum*: Olyan attribútum, mely egy egyed példányhoz csak egy értéket vehet fel. Pl.: Neve, anyja neve, stb.
- *Többértékű attribútum*: Olyan attribútumok, melyek egy egyed példányhoz egyszerre több értéket is felvehet. Pl.: Gyermekei neve, beszélt idegen nyelvek, stb.

Azt az attribútumot, vagy az attribútumoknak azt a csoportját, amelynek alapján az entitás példányai egyértelműen megkülönböztethetők, *kulcsnak* nevezzük. Sokszor ez alapján keresünk (el akarjuk dönteni, hogy az illető entitás példány számunkra érdekes-e, vagy sem). Keresni azonban bármely attribútum vagy attribútum csoport alapján lehet, legfeljebb több entitás példány is kielégíti a keresési szempontokat. Az előbbi "tojás" példára visszatérve, az üzletben az entitás az áru,

ennek attribútumai pl: az áru neve és mennyisége, ahol a tojás az egyik áru neve. A biológiai vizsgálatnál a tojás az entitás, és ennek attribútumai az azonosító (amit ráírtak, vagy ahová elhelyezték), keltetési hőmérséklet, forgatások száma, gyakorisága, kikelési idő stb.

Szokásos jelölés, hogy az entitás név után zárójelben felsorolják az attribútumokat. Pl. legyen az entitás egy személy, attribútumai a név, város, cím, személyi szám, születési dátum, telefonszám. Ekkor:

SZEMÉLY (név, város, cím, személyi szám, születési dátum, telefonszám)

lehet a megadási formátum. A kulcsot aláhúzással jelöljük.

1.2.5. Kapcsolatok

Az egyes entitások között kapcsolatok értelmezhetők.

A kapcsolat foka azt határozza meg, hogy hány egyed vesz részt a kapcsolatban. Ez lehet egyes (unary) kettes (binary) hármas (ternary) vagy magasabb. Kettes kapcsolatok a leggyakoribbak. A kettes kapcsolatoknak (relationship) három alapvető típusa van.

- *Egy-egy típusú kapcsolat*
- *Több-egy típusú kapcsolat*
- *Több-több típusú kapcsolat*

A kapcsolat kardinalitása azt mutatja meg, hogy az adott egyedtípusból hány példány vehet, és hány példány vesz kötelezően részt a kapcsolatban. Minimum kardinalitás az a minimális példányszám az egyedekből, amelynek részt kell vennie a kapcsolatban. Maximum kardinalitás az egyedpéldányok maximális száma, ami egy adott egyedhez kapcsolódhat a kapcsolatban. A kapcsolatok "egy"-oldalán a minimum egy a kötelező.

1.2.5.1. Egy-egy kapcsolat

Az egy-egy típusú kapcsolat (pl. férj-feleség, szervezet-vezetője), olyan kapcsolat, melyben az egyik entitáshalmaz egyes egyedei egy másik entitáshalmaznak pontosan egy egyedével vannak kapcsolatban. Pl.:

HÁZASSÁG (ember, ember)

VEZETŐ (cég, ember) (1.2 ábra)

1.2.5.2. Több-egy kapcsolat

Több-egy kapcsolat esetén az egyik entitáshalmaz pontosan egy egyedéhez tartozik a másik entitáshalmaz több példánya. Minden esetben egyszerűen eldönthető a hozzátartozás kérdése. Pl:

DOLGOZIK (tanszék, oktató)(1.2 ábra)

Itt egy oktató egy tanszéken dolgozik, de egy tanszéken több oktató is dolgozhat.

1.2.5.3. Több-több kapcsolat

Az előbbiekbe nem tartozó kapcsolat, itt egy entitáshalmaz több eleme van kapcsolatban egy másik entitáshalmaz több elemével. Pl.:

TANUL (tanár, diák)(1.2 ábra)

Itt egy tanár több diákot is tanít, ugyanakkor egy diák több tanárnál is tanul.

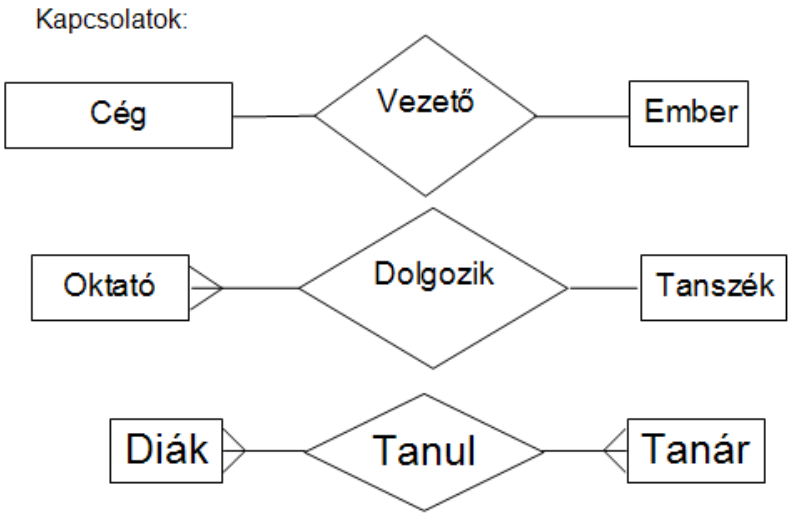
1.2.5.4. A kapcsolat attribútuma

A kapcsolatoknak is lehetnek attribútumai. Ezek olyan attribútumok, amelyek egyik résztvevő példányra sem jellemzőek, hanem a köztük lévő kapcsolat tulajdonságai. Pl. az előző példában a félév lehet ilyen, hogy melyik félévben tanult a diák az adott tanárnál, azaz

TANUL (tanár, diák, félév).

1.2.5.5. A kapcsolatok ábrázolása

Gyakran alkalmazzák az ú.n. entity-relationship (E-R, magyarul: egyed-kapcsolat E-K) ábrázolásmódot. Ennek lényege, hogy az entitásokat téglalapokkal ábrázoljuk, az attribútumokat körökkel vagy ellipszisekkel, a kapcsolatokat rombuszokkal jelöljük. Az összetartozást irányított élekkel jelenítjük meg. Erre példát az 1.2. és az 1.3. ábrákon láthatunk.



1.2. ábra. Kapcsolatok egyed-kapcsolati diagramja

1.3. Fizikai adatszervezés

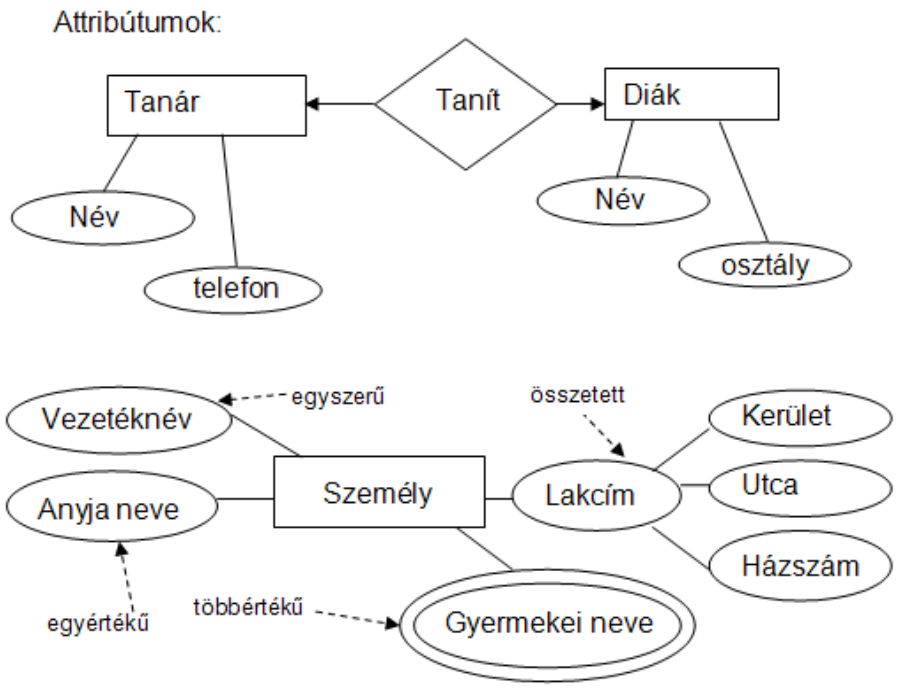
Ebben a fejezetben arról lesz szó, hogyan tárolják az adatbázis adatait - lehetőleg hatékonyan - egy nem felejtő tárolóeszközön (pl. mágneslemezen). Ennek mégértéséhez az adott tárolóeszköz ismerete is szükséges.

A tároló mágneslemez blokkokra van osztva. A blokk egy fizikai egység, ami egyszerre írható vagy olvasható. A blokkok fizikailag címezhetőek. A blokkon belül helyezkednek el a blokk fejrésze, majd a rekordok, amik a tárolás logikai egységei. Egy-egy rekord a fejrészből és a tárolt adatmezőkből áll. A blokk és a rekord szerkezete az 1.4. ábrán látható.

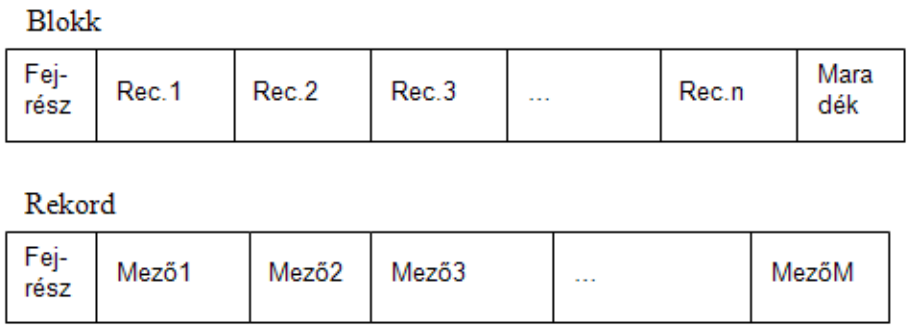
A blokk hasznos mérete nem feltétlenül osztható a rekordmérettel, bizonyos rendszerekben egy rekord nem nyúlik át blokkhatáron, ilyenkor a blokkokban kihasználatlan helyek maradhatnak, egyébként csak az utolsó blokkban lesz kihasználatlan hely.

1.3.1. Heap szervezés

Ennél az ábrázolási módnál a legegyszerűbb a tárolás. Az adatokhoz entitásonként külön tárolási területet, fájlt(állományt) rendelünk. Az egyes entitások adatait a felmerülésük sorrendjében egymás után felírjuk a megfelelő fájlba. Megadjuk az



1.3. ábra. Attribútumok és az egyed-kapcsolati diagram



1.4. ábra. A blokk és a rekord felépítése

entitás kulcsát, és a tárolt adat hosszát. Az utóbbira csak akkor van szükség, ha nem fix méretű rekordokkal dolgozunk.

1.3.1.1. Keresés

Mivel mindent éppen akkora helyen tárolunk, amennyi hely szükséges, a tárolásban nincs rendszer. Egy entitás példány keresése az állomány valamennyi adatának elolvasását jelenti, amíg csak rá nem találunk a keresettre. Az olvasást mindig az

állomány elejéről kell kezdeni, hiszen egy közbülső adat kezdetét nem ismerjük. Ez különösen akkor gond, ha megsérül a tároló lemezen egy adat hosszát jelentő bejegyzés, ekkor ugyanis nagyon nehéz a következő adatokat megtalálni. Ha egy blokkban átlagosan R rekord található, akkor ennél a szervezésnél n adat esetén a blokkok száma B = (n-1)/R +1. Egy adat megtalálása átlagosan (B+1)/2 blokk olvasását jelent. Legjobb esetben 1, legrosszabb esetben B blokk olvasás kell.

1.3.1.2. Törlés

A törlendő adatot megkeressük. Jelezzük, hogy a terület törölt. (Egy szemétgyűjtő programmodul e jelzés alapján tudja, hogy ezt a területet ki lehet söpörni.) Amenynyiben a törlendő adat létezik, átlagosan (B+1)/2 blokk olvasás és egy írás kell egy törléshez. Ha nem létezik a törlendő adat, akkor B blokk olvasás szükséges (végig kell olvasni valamennyi adatot, mire kiderül, hogy ilyen nincs).

Időnként szükséges a tárolt adatok átrendezése, a törölt helyek megszüntetése. Ezt szemétgyűjtésnek (garbage collection) szokás nevezni, amely az állomány újraírásával összefüggő területre írja az adatokat.

1.3.1.3. Beszúrás

Végigolvassuk az állományt, hogy van-e már ilyen kulcsú adat (B blokk olvasás). Amennyiben van, hibaüzenetet küldünk (duplikált adat), amennyiben nincs, először a törlés által felszabadított területekre írunk. Ha nincs ilyen, vagy az uj adat hosszabb, mint a törölt hely, az állomány végére írjuk fel. Értelemszerűen olvasás közben tároljuk a szabad helyeket, hogy az írást közvetlenül elvégezhessük.

1.3.1.4. Módosítás

Megkeressük az adatot(keresés), majd a módosított adatot visszaírjuk, he ez elfér a régi helyére. Ha nem, akkor a régit töröljük, és egy új helyre írjuk fel a módosított adatot.

1.3.2. Hash állományok

A hash címzés vagy csonkolásos címzés onnan kapta nevét, hogy elvileg a kulcs bitmintájából csonkolással nyerhető cím is használható címgenerátorként. A módszer lényege ugyanis az, hogy a kulcsból, mint bitmintából, alkalmas függvény segítségével címet generálunk, és ez lesz a keresett adatsor címe. A függvénynek olyannak kell lennie, hogy a szóba jöhető kulcsokat egyenletesen terítse a címtartományon.

Minden bejegyzett adatsort kiegészítünk két bittel, az egyik jelentése "szabad / foglalt", a másiké pedig "még sohasem volt foglalt". Az állományt használatba vétel előtt ezzel a két bittel inicializálni kell. Ezen bitek együttes vizsgálatával lehet eldönteni egy rekord állapotát.

Szabad/ foglalt	*Soha sem volt foglalt*	*A rekord állapota*
1	1	üres
0	0	foglalt
1	0	törölt

Ha valóban csonkolásos címgenerátort használunk, akkor a címtartomány kettőnek egészszámú hatványa. Amennyiben moduló n címgenerátort használunk, akkor a cimtartomány éppen n lesz. Az adatok helyét logikailag tömbnek képzeljük, így az adatrekordok rögzített hosszúságúak. Ha két kulcshoz ugyanazt a címet generáljuk, akkor egy másodlagos címmelfolytatjuk a műveletet. A másodlagos cím egy továbblépést jelent az adott címhez képest. Ennek relatív prímnek kell lennie a tartomány méretéhez, hogy szükség esetén akár az egész tartomány bejárhassuk. Legegyszerűbb esetben ez az érték 1.

Egy változat az ú. n. "vödrös" hash (bucket hashing), amikor a hash függvény által visszaadott cím csak adatblokkok halmazát (vödör) azonosítja. A "vödrön" belül a tárolás a heap-hez hasonlóan rendezetlen, vagy rendezett, vagy rendezett lista. Ha a "vödör" betelik, egy túlcsordulási területre írunk.

1.3.2.1. Keresés

Generálujuk a címet a rekord kulcsa alapján. Ha a megcímzett rekeszben van érvényes adat, ellenőrizzük a kulcsot, ha megfelel, megtaláltuk az adatot. Ha a rekesz üres ("még sohasem volt foglalt"), akkor a keresett adat nem szerepel az

állományban. Ha a rekesz törölt vagy foglalt, másodlagos címgenerátort használunk, és addig kérünk másodlagos címet, míg meg nem találjuk az adatot, vagy üres helyet nem találunk. Ez utóbbi esetben nincs találat.

1.3.2.2. Törlés

Megkeressük a kívánt adatot. Ha megtaláltuk, foglaltsági jelzését szabadra állítjuk, ha nem találjuk meg, hibajelzést adunk (nincs ilyen adat).

1.3.2.3. Beszúrás

Generálujuk a címet a rekord kulcsa alapján. Megkeressük az adatot. Ha megtaláljuk, hibaüzenetet adunk (duplikált kulcs). Ha nincs ilyen adat és a címhez tartozó rekesz még sohasem volt foglalt, akkor használttá és foglalttá tesszük. Az adatot beírjuk. Ha már volt foglalt, de most szabad, foglalttá tesszük és beírjuk az adatot. Ha foglalt, másodlagos címgenerátort használunk, amíg üres vagy szabad helyet nem találunk, és ide beírjuk az adatot. Foglalt rekeszekre ellenőrizzük a kulcsot, ha megegyezik a beszurandó kulccsal, duplikált kulcs hibaüzenetet adunk.

1.3.2.4. Módosítás

Megkeressük a módosítandó adatot és beolvassuk. Ha a módosítás nem a kulcsmezőt érinti, javítjuk a szükséges mezőket, majd visszaírjuk. Amennyiben a módosítás a kulcsmezőt is érinti, a beolvasott adatot töröljük (foglaltsági bitjét szabadra állítjuk és visszaírjuk), az új kulcssal bevisszük az adatot új bevitelként(beszúrás).

1.3.2.5. Keresés vödrös hash esetén

Generálujuk a címet a rekord kulcsa alapján. Ha a megcímzett blokkban benne van az érvényes adat (ellenőrizzük a kulcsokat), megtaláltuk az adatot. Ha a blokkban nincs benne az érvényes adat, és van benne üres, "még sohasem volt foglalt" jelzésű rekesz, akkor az adat nem szerepel. Ha a rekesz egy túlcsordulási területre mutató pointert tartalmaz, akkor a túlcsordulási területen lineáris kereséssel keressük az adatot.

1.3.2.6. Törlés vödrös hash esetén

Megkeressük a kívánt adatot. Ha megtaláltuk, foglaltsági jelzését szabadra állítjuk, ha nem találjuk meg, hibajelzést adunk.

1.3.2.7. Beszúrás vödrös hash esetén

Generálujuk a címet a rekord kulcsa alapján. Megkeressük az adatot. Ha megtaláljuk, hibaüzenetet adunk (duplikált kulcs). Ha nincs ilyen adat és a címhez tartozó blokkban van üres, "még sohasem volt foglalt" rekesz, akkor használttá és foglalttá tesszük és az adatot beírjuk. Ha már volt foglalt, de most szabad (törölt rekesz), foglalttá tesszük és beírjuk az adatot. Ha minden rekesz foglalt, egy túlcsordulási területre írjuk az adatokat, és a blokkban beállítjuk a túlcsordulási pointert.

1.3.2.8. Módosítás vödrös hash esetén

Megkeressük a módosítandó adatot, beolvassuk, ha a módosítás nem a kulcsmezőt érinti, javítjuk a szükséges mezőket, majd visszaírjuk. Amennyiben a módosítás a kulcsmezőt is érinti, a beolvasott adatot töröljük, az új kulcssal bevisszük az adatot új bevitelként (beszúrás).

1.3.3. Indexelt állományok

Alapgondolata az, hogy a kulcsot egy index állományban megismételjük, és hozzárendeljük a tárolt adatrekordra mutató mutatót. A kulcsot rögzített hosszúsággal ábrázoljuk. Az index állományt a kulcs szerint mindig rendezve tartjuk. Az adatállomány méretével takarékoskodhatunk, ha megtartjuk a korábban megismert foglaltsági jelzés bitet. (Ha az index állományba tesszük, gyorsabb lehet a beszúrás folyamata).

- *Sűrű index:* Ha minden egyes adatrekordra mutat mutató.
- *Ritka index:* Ha csak az adatrekordok halmazára - tipikusan blokkokra - mutat mutató.

Ritka index esetén a blokkon belül az adatokat a gyorsabb keresés miatt célszerű rendezetten tárolni.

1.3.3.1. Keresés sűrű index esetén

Az index állományban megkeressük a kulcsot, pl. bináris kereséssel. Ha nincs, hibaüzenetet adunk. Ha van, a hozzá tartozó mutatóval elérhetjük a tárolt adatot.

1.3.3.2. Törlés sűrű index esetén

Megkeressük a kívánt adatot. Foglaltsági jelzését szabadra állítjuk. A kulcsot kivesszük az index állományból, és az index állományt tömörítjük, vagy csak jelöljük a törlést az index állományban, és későbbre hagyjuk a tényleges törlést és tömörítést.

1.3.3.3. Beszúrás sűrű indexek esetén

Az index állományban megkeressük a kulcsot, pl. bináris kereséssel. Ha van, akkor duplikált kulcs jelzéssel hibaüzenetet adunk, Ha nincs, keresünk egy üres helyet a tárolandó adatnak (a törölt rekordok helyén). Ha nem találunk, akkor az állomány végére vesszük fel. Beállítjuk a foglaltsági jelzést, és beírjuk az adatot. A kulcsot és a tárolás helyére hivatkozó mutatót a kulcs szerint berendezzük az index állományba.

1.3.3.4. Módosítás sűrű index esetén

Megkeressük a kívánt adatot. Ha a módosítás nem érinti a kulcsot, az adatrekordot beolvassuk, módosítjuk, majd visszaírjuk. Ha a módosítás a kulcsot is érinti, akkor a visszaírás után az indextáblában is módosítjuk a kulcsot, majd rendezzük.

1.3.4. B* - fák, mint többszintes ritka indexek

Az index állomány tárolásában is különbözik az előzőtől. A bináris keresésnél gyorsabb, $log_k - val$ arányos keresési időt érhetünk el, ha az indexeket pontosan k-ágú fában tároljuk. A legalsó szint mutatói az adatállomány egy-egy blokkjára mutatnak, a fölötte levő szintek mutatói pedig az index állomány egy-egy részfáját azonosítják. Az egy csomópontban ábrázolt k mutatóhoz elegendő k-1 kulcs tárolása, mert a kulcs jelentése a kijelölt részfában tárolt legkisebb érték. Így az első bejegyzés nem hordozna információt.

1.3.4.1. Keresés

Az index állományban logaritmikus kereséssel megkeressük a kulcshoz tartozó blokk mutatóját. Ezt ugy tehetjük meg, hogy megkeressük azt a kulcs bejegyzést az index táblában, amely a keresett kulcsnál nagyobbak közül a legkisebb, és az ez előtti blokkot olvassuk be. Ha a kulcsnál nincs nagyobb bejegyzés, akkor az utolsó mutatóval olvasunk. Ha a blokkban nem létezik a megadott kulcsú adat, hibaüzenetet adunk.

1.3.4.2. Törlés

Megkeressük a kívánt adatot és töröljük. Az adatblokkokat lehetőség szerint összevonjuk. Ha a kulcs index bejegyzést is érint, a törölt kulcsot kivesszük az index állomány érintett részfájából. Bizonyos esetekben (pl. a törlés hatására a blokk üres lett) az egész fát át kell rendezni.

1.3.4.3. Beszúrás

Megkeressük, hogy a tárolandó adat melyik blokkba tartozik. Ezt ugy tehetjük meg, hogy megkeressük azt a kulcs bejegyzést az index táblában, amely a keresett kulcsnál nagyobbak közül a legkisebb, és az ez előtti blokkot olvassuk be. Ha ebben van ilyen kulcsú adat, hibaüzenetet adunk. Amennyiben az adat még nem létezik, megnézzük, hogy találunk-e az adott blokkban számára helyet, ha van, beírjuk az adatot, ha nincs, akkor új blokkot kérünk. A megfelelő kulcsot és a tárolás helyére hivatkozó mutatót a kulcs szerint berendezzük az index állomány legalsó szintjébe. Ha a szint betelt, új szintet nyitunk, és az egész fát rendezzük. Egyébként csak az érintett részfát rendezzük.

1.3.4.4. Módosítás

Kiolvassuk a módosítandó adatot, elvégezzük a módosítást, majd visszaírjuk. Ha a módosítás a kulcsmezőt is érinti, általában egy beolvasás utáni törlés és új adatbevitel(beszúrás) lehet a megoldás.

1.3.5. Másodlagos indexek, invertálás

A cél az, hogy több kulcs szerint is kereshessünk. Szélső esetben akár minden mező lehet kulcs. Ennek következménye, hogy több index állományra lehet szükségünk. Legrosszabb esetben, ha minden mező kulcs, akkor ez kétszeres tárolási igényt jelent az adatokra nézve, és ehhez jön még a mutatók tárolásának helyfoglalása. Elvi probléma azonban nem merül fel. Ez a helypazarlás elkerülhető az invertálási technikával. Az index állományok mutatói egy olyan adatállományra mutatnak, amelyben az egyes mezők - ha ezek kulcsmezők -, a megfelelő index állományokba mutató hivatkozásokat tartalmaznak, ami alapján az adat az indexállományból kiolvasható, és csak akkor tartalmaznak tényleges adatot, ha a mező szerint nincs indexelve az állomány. Ez beszúrás és törlés esestén sokszoros adminisztrációt jelent, ezért gyakran az index állományok végén függeléket használnak, ahol az újabban felvett adatok bejegyzéseit tárolják. A függelékben lineáris keresés alkalmazható. Ha a függelék mérete egy kritikus határt elér, az index állományokat rendezni kell. Ha a kulcs módosul, a módosítás a legegyszerűbben a törlés-beszúrás módszerével végezhető el.

1.3.5.1. Keresés

A megfelelő index állományban megkeressük a kulcsot. A hozzá tartozó mutatóval elérhetjük az invertált állományban tárolt adatot. A kiolvasott mutatókkal összeállítjuk a rekordot az egyes index állományokban tárolt kulcsokból.

1.3.5.2. Törlés

Megkeressük a kívánt adatot az invertált állományban és a bejegyzés foglaltsági jelzését szabadra állítjuk. Valamennyi index állományból töröljük a hivatkozásokat, de a kulcsokat nem.

1.3.5.3. Beszúrás

Az új adatot az állomány végére a függelékbe vesszük fel. Beállítjuk a foglaltsági jelzést, beírjuk az adatokat. Ha a függelék mérete egy kritikus határt elér, az index állományokat és az invertált állományt is rendezni kell.

1.4. Adatmodellek

Az adatmodell meghatározza, hogy az adatbázisban az adatok milyen szerkezetben tároljuk és milyen mechanizmusokon keresztül lehet az adatokhoz hozzáférni. Így az adatbázis kezelő rendszer legalapvetőbb tulajdonságait rögzíti. Egyetlen adatbáziskezelő-rendszer mindig egyetlen adatmodellnek megfelelően működik.

1.4.1. A hierarchikus adatmodell

A legrégebbi adatmodell. Lényegében az IBM cég IMS (Information Management System) rendszerén alapul. Ma már új rendszereket nem telepítenek, jelentősége csekély, elsősorban elvi.

1.4.2. A hálós adatmodell

A hálós adatmodellre épülő adatbázisok mintapéldája a COBOL nyelv szabványosításáról ismert Conference on Data System Languages (CODASYL) Data Base Task Group (DBTG) nevű csoportjához fűződik. Az általuk kidolgozott ajánlásnak két eleme van. Az adatdefiníciós formalizmus Subschema Data Definition Language (Subschema DDL) néven, az applikációs programok írására alkalmas formalizmus Data Manipulation Language (DML) néven vált ismertté.

1.4.2.1. Alaptulajdonságok

A hálós adatmodell egy olyan egyed-kapcsolati adatmodell, amely csak többes-egyes típusú bináris kapcsolatokat enged meg a típusok szintjén. Ez a megszorítás lehetővé teszi, hogy adatainkat egyszerű irányított gráffal jellemezzük. Ez a kapcsolatok implementálását is megkönnyíti. Alepvető stuktúraegysége a rekord, amely számos atomi komponensből tevődhet össze (mezők és pointerek). A következő strukturaegység lehetővé teszi a rekordok összetartozásának megjelenítését láncolás formájában, így születnek meg a CODASYL terminológia szerinti Set-ek. A rekordok egyidejűleg több kapcsolatban is szerepet játszhatnak, így a rekordok változatos módon kapcsolódhatnak össze. Innen az adatmodell elnevezése.

Az összetartozó rekordok rendezett összefogása céljából vezették be a Set fogalmát, amely kétféle rekordból áll, az egyenrangú, összeláncolt Member-rekordoknak

egy halmaza (lehet üres is), és egy Owner-rekord, aminek a Member-rekordok alárendeltek. Az összeláncolt rekordok ugyanannak a kapcsolatnak a példányait (eseteit) valósítják meg.

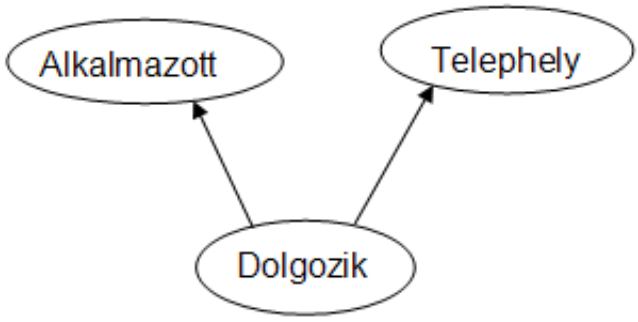
A Set-típusokat grafikus ábrázolásban hagyományosan a Member-típustól az Owner-típushoz irányított nyilakkal jelezzük, mint ezt az 1.5 ábrán láthatjuk. Pl.

ALKALMAZOTT (A_kod, nev, beosztas, fizetes) - entitás típus 1

TELEPHELY (T_kod, város, cim) - entitás típus 2

DOLGOZIK (A_kod, T_kod, ev) - kapcsolat

Itt Owner rekord típus lesz az alkalmazott és a telephely is, míg Member record típus lesz a dolgozik kapcsolat.



1.5. ábra. Az owner és a member rekordok megjelenítése

Legyenek az alkalmazottak az alábbi tábla szerintiek.

A_kod	**nev**	**beosztas**	**fizetes**
101	Kiss István	könyvelő	152000
102	Nagy Tibor	technikus	133000
103	Sós Emil	kézbesítő	112000

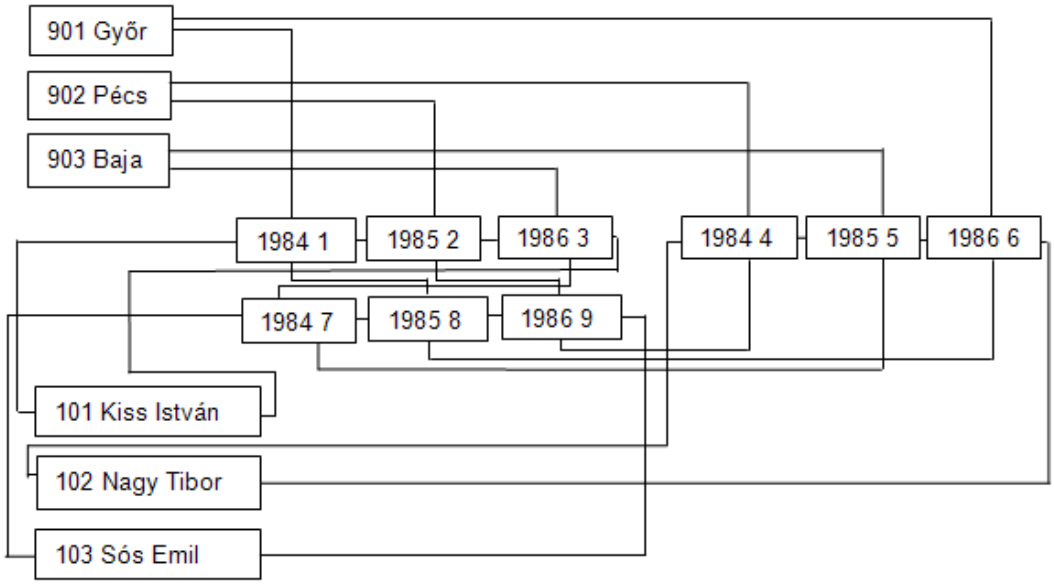
Legyenek a telephelyek az alábbi táblának megfelelőek.

T_kod	**város**	**cim**
901	Győr	Fő u. 2.
902	Pécs	Szél u. 12.
903	Baja	Híd u. 33.

Az egyes alkalmazottak pedig egy megadott időben a megadott kirendeltségnél dolgoztak, amint ezt az alábbi táblán megadtuk.

alkalmazott	telephely	idő	sorszám
Kiss István	Győr	1984	1
Kiss István	Pécs	1985	2
Kiss István	Baja	1986	3
Nagy Tibor	Győr	1986	4
Nagy Tibor	Pécs	1984	5
Nagy Tibor	Baja	1985	6
Sós Emil	Győr	1985	7
Sós Emil	Pécs	1986	8
Sós Emil	Baja	1984	9

Ezt hálós modellben ábrázolva a 24. oldalon az 1.6. ábrán láthatjuk.



1.6. ábra. Hálós modell

1.4.3. Relációs adatmodell

A reláció szót itt halmazelméleti értelemben használjuk. Adott n halmaz, amelyekből képzett Descartes-szorzat egy részhalmaza a reláció.

A reláció matematikai definíciója: Legyenek adottak a $T_1, T_2, ...T_n$ halmazok és képezzük ezen halmazok $T = T_1 \times T_2 \times ... \times T_n$ Descartes-szorzatát. (A T szorzat

elemei olyan $t_1, t_2, ...t_n$ elem n-esek, amelykre $t_1 \subset T_1$, $t_2 \subset T_2$, $...t_n \subset T_n$.) A T szorzathalmaz egy R résztartományát relációnak, a $T_1, T_2, ...T_n$ halmazokat pedig a reláció tartományainak, attribútumainak nevezzük.

Magát a relációt is névvel látjuk el. A reláció neve után az attribútumneveket zárójelek között adjuk meg. Pl.:

SZEMÉLY (név, kor, foglalkozás),

amit relációs sémának nevezünk. Áttekinthetőbben ábrázolhatjuk relációnkat táblázatos formában. A táblázat oszlopai jelentik a tartományokat, a sorai tartalmazzák a relációban álló n-esek konkrét előfordulásait, eseteit. A fejlécbe az attribútumok megnevezése kerül. A SZEMÉLY (név, kor, foglalkozás) relációt az alábbi táblázat mutatja.

név	**kor**	**foglalkozás**
Nagy István	37	üzletkötő
Kiss Oszkár	26	programozó
Tóth Tibor	44	teremőr
Varga János	22	hallgató

- A relációban lévő oszlopok (tulajdonságok, attribútumok, tartományok) számát a **reláció fokának** nevezzük.
- A relációban lévő sorok számát (a konkrét előfordulások számát) a **reláció számosságának** nevezzük.
- Azt a tulajdonságot vagy tulajdonsághalmazt, amely a táblázat egy-egy sorát egyértelműen meghatározza, **kulcsnak** nevezzük.

A reláció:

- Nem tartalmaz két azonos sort.
- A sorok sorrendje nem számít.
- Az oszlopoknak egyértelmű neve, helye, sorszáma van. (Amennyiben az oszlopokra a nevükkel hivatkozunk, akkor számunkra a helye és a sorszáma közömbös, ugyanakkor az adatbáziskezelő számára fontos a sorszáma és a helye az oszlopnak, mert ez alapján kezeli ezeket).
- Tetszőleges számú sort tartalmazhat.
- Az oszlop és sor kereszteződésében egy érték szerepel.

1.4.3.1. Műveletek relációs adatbázisokon

A halmazelméletben megismertek alapján néhány halmazalgebrai műveletet relációs műveletként kívánunk használni, vagyis operandusaink relációk, illetve ezek attribútumai.

Egyesítés (unió)

Az egyesítés feltétele, hogy az egyesítendő relációknak azonos n-esekből kell állniuk. Nem szükséges azonban, hogy ezek ténylegesen azonos attribútumokat jelentsenek. Ez azt jelenti, hogy a műveletet ilyenkor mindig el tudjuk végezni, de nem biztos, hogy az eredmény attribútumait sorszámukon kívül nevükkel is azonosítani tudjuk. Erre példát az alábbi táblázat mutat.

R_1

A	**B**	**C**
a	b	c
c	b	a
a	d	c

R_2

D	**E**	**F**
a	c	d
a	d	c
b	b	c

$R_1 \cup R_2$

1	**2**	**3**
a	b	c
a	c	d
c	b	a
a	d	c
b	b	c

Egyesítéskor az eredő reláció tartalmazza az egyesítendő relációk összes sorát. Ha van megegyező sor a két relációban, az eredőben ez csak egyszer kerül bele. Természetesen a valóságos esetekben csak olyan unióknak van értelmük, ahol az egyes oszlopok azonos attribútumokat tartalmaznak (persze lehet, hogy az oszlopok nevei különböznek, de tartalmilag azonosak). Ha két olyan relációt szeretnénk egyesíteni, ahol pl. az egyikben kevesebb oszlop van, akkor ezt a megfelelő helyen egy olyan oszloppal egészítjük ki, ami csupa NULL értéket tartalmaz (később a 1.9.4.1 fejezetben részletesen tárgyaljuk), majd ezután már elvégezhető az unió.

Különbségképzés

Ugyanazok a megkötések érvényesek, mint az egyesítésnél. Különbségképzéskor az $R_1 - R_2$ reláció tartalmazza az R_1 összes olyan sorát, ami R_2-ben nem szerepel. A különbségképzésre egy példát az alábbi táblázat mutat.

R_1

A	**B**	**C**
a	b	c
c	b	a
a	d	c

R_2

D	**E**	**F**
a	c	d
a	d	c
b	b	c

$R_1 - R_2$

1	**2**	**3**
a	b	c
c	b	a

Metszet

Értelme megegyezik a halmazalgebrai jelentéssel. Ugyanazok a megkötések érvényesek, mint az egyesítésnél. Erra példát az alábbi táblázat mutat.

R_1

X	**Y**	**Z**
a	b	c
c	b	a
a	d	c

R_2

U	**V**	**W**
a	c	d
a	d	c
b	b	c

$R_1 \cap R_2$

1	**2**	**3**
a	d	c

Vetítés (projekció)

A művelet azt jelenti, hogy egy meglevő R reláció egyes oszlopaiból egy új relációt hozunk létre. Ehhez ki kell jelölnünk, hogy mely oszlopokat kívánjuk felhasználni a relációból, és az új relációban mi legyen az oszlopok sorrendje. A halmazalgebrában szokásos jelölésmód az eredeti oszlopokat sorszámukkal azonosítja. Ez megengedett a reláció algebrában is, de szokás helyette az attribútumokat nevükkel azonosítani.

$$R_2 = \Pi_{1,3,7,2}(R_1)$$

Fenti jelölés azt írja elő, hogy vegyük az R_1 reláció első, harmadik, hetedik és második attribútumát és ebben a sorrendben vegyük fel az új R_2 relációba.
Hasonlóképpen a

GÉPKOCSI_TIPUS $=\Pi_{\text{évjárat, fogyasztás}}$ (GÉPKOCSI)

az eredeti (GÉPKOCSI) reláció két oszlopát tartalmazza. Az eredeti (GÉPKOCSI) reláció ezen kívül tartalmazhatta még az ÁR , RENDSZÁM, ELSŐ_TULAJDONOS, VIZSGA_ÉRVÉNYESSÉGE, stb. attribútumokat.

A vetítés eredményeként kaphatunk több azonos sort. Az eredmény relációban az azonos sorok csak egyszer fognak megjelenni.

Kiválasztás (szelekció)

A kiválasztás művelete egy részhalmaz képzése az R reláción, amelynek vezérlésére egy logikai kifejezés szolgál. Az R reláció valamennyi sorára kiértékeljük a logikai kifejezést, és azokat a sorokat vesszük be az új relációba, amelyekre a kifejezés igaz. Jelölése:

$$\sigma_{LK}R,$$

ahol LK a logikai kifejezés.

Adatbázisoknál 3 értékű logikát használunk, IGAZ (TRUE) , NULL és HAMIS (FALSE) lehet egy logikai kifejezés értéke. Az ÉS(AND) illetve a VAGY(OR) műveletekre az igazságtábla az alábbiakban látható:

AND

	TRUE	**NULL**	**FALSE**
TRUE	TRUE	NULL	FALSE
NULL	NULL	NULL	FALSE
FALSE	FALSE	FALSE	FALSE

OR

	TRUE	**NULL**	**FALSE**
TRUE	TRUE	TRUE	TRUE
NULL	TURE	NULL	NULL
FALSE	TRUE	NULL	FALSE

A logikai kifejezés felépítése a következő lehet:

- operandusok amelyek lehetnek konstansok, vagy attribútumok azonosítói,
- aritmetikai operátorok ($<=>\leq\geq$)
- logikaioperátorok ($\wedge \vee \neg$)

Megjegyezzük, hogy az egyértelműség érdekében a numerikus konstansokat is aposztrófok közé írjuk, hogy meg lehessen különbözteni az oszlopok sorszámától. Ennek megfelelően $\sigma_{2>5}(R)$ az R reláció azon elemeinek halmazát jelenti, amelyekre igaz, hogy a második oszlop értéke nagyobb az ötödik oszlop értékénél, $\sigma_{KOR<'23'\wedge 2='Nagy'}$(NÉVSOR) pedig a NÉVSOR reláció azon elemeit jelenti, amelyeknek KOR azonosítójú összetevője kisebb huszonháromnál, második összetevője pedig Nagy, vagyis a 23 évesnél fiatalabb, Nagy nevű személyek (feltéve, hogy a 2. oszlop a NÉV attribútum, de ha ez a házastársa neve, akkor a 23 évesnél fiatalabb, olyan személyek, akiknek a házastársuk neve Nagy) jelennek meg az új relációban.

Hányados

A hányados képzés voltaképpen szójáték, ugyanis a Descartes-i szorzás megfordítását jelenti. Jelölje $R \div S$ azt a relációt, amely úgy keletkezik, hogy vesszük az

R reláció n_1 -esei közül azokat, amelyek utolsó n_2 összetevője - mint önálló n_2-es - eleme az S relációnak, és az első $(n_1 - n_2)$ elemét $(K - t)$ - mint $(n_1 - n_2)$-est vegyük be az $R \div S$ relációba, feltéve, hogy $K \times S$ minden eleme benne van R-ben. (hátulról osztunk). Eben az esetben $(R \div S) \times S \subseteq R$.

Amennyiben az R reláció n_1 -esei közül azokat vesszük, amelyek első $n2$ összetevője - mint önálló n_2-es - eleme az S relációnak, és az utolsó $(n_1 - n_2)$ elemét $(V - t)$- mint $(n_1 - n_2)$-est vesszük be az $R \div S$ relációba, feltéve, hogy $S \times V$ minden eleme benne van R-ben (elölről osztunk), akkor $S \times (R \div S) \subseteq R$.
Az elölről osztásra egy példát az alábbi táblázatban láthatunk.

R

X	**Y**	**Z**	**U**
A	B	A	C
A	B	C	D
A	B	D	A
E	F	A	C
E	F	C	D
E	F	B	D

S

X	**Y**
A	B
E	F

$R \div S$

Z	**U**
A	C
C	D

A műveletet úgy végezhetjük el mint egy osztást, pl. az S-beli AB-vel osztjuk R első sorát (ABAC), megvan benne AC-szer, majd visszaszorzunk S EF elemével (EFAC), és ennek benne kell lennie R-ben, azaz EFAC R eleme kell hogy legyen, mivel ez teljesül, ezért a hányadosba belekerül AC. Ezután AB-vel osztjuk R második sorát(ABCD), ez megvan benne CD-szer, és a visszaszorzás EF-fel (EFCD) is jó, mert benne van R-ben. A harmadik sort(ABDA) osztva AB-vel DA-t kapunk, amit visszaszorozva EF-fel EFDA-t eredményez, ez nincs benne R-be, ezért DA nem eleme a hányadosnak. A további sorok AB-vel nem oszthatóak, így készen vagyunk.

Descartes-szorzat

Adott az R_1 reláció n_1-esek halmaza (n_1 attribútumból kiválasztott értékek halmaza), míg az R_2 reláció n_2-esek halmaza. Az $R_1 \times R_2$ Descartes-szorzat eredménye olyan $(n_1 + n_2)$-esekből áll, amelyeknek első n_1 eleme az első operandusból(relációból), második n_2 eleme a második relációból származik, ebben a rögzített

sorrendben. Az operandusok szerkezetére ebben az esetben semmilyen megkötést sem kell tennünk. Az R_1 reláció összes sorát kapcsolatbe kell hozni az R_2 reláció összes sorával. A Descartes szorzatra példát az alábbi táblázat mutat.

R_1

A	**B**
a	b
b	a

R_2

C	**D**	**E**
c	d	e
a	c	f

$R_1 \times R_2$

A	**B**	**C**	**D**	**E**
a	b	c	d	e
a	b	a	c	f
b	a	c	d	e
b	a	a	c	f

Természetes illesztés

A természetes illesztést más néven összekapcsolásnak vagy natural join-nak is nevezik. Legyen R és S két reláció, amelyeknek van legalább egy, de akár több név szerint megegyező oszlopa (azonos attribútuma). Vegyük sorra a két reláció valamennyi elemét, és válasszuk ki azokat, amelyekben a megegyező nevű oszlopai érték szerint is megegyeznek. Egyesítsük ezeket olyan Descartes-szorzattá, amelyben a mindkét relációban szereplő attribútumokat csak egyszer vesszük figyelembe. Jelölése: $R \bowtie S$

A természetes illesztésre példát az alábbi táblázat mutat.

R

X	**Y**	**Z**
A	B	C
A	B	E
A	D	E
C	D	A

S

Y	**Z**	**U**
B	C	D
B	C	E
B	E	F
B	E	A
D	A	F

$R \bowtie S$

X	**Y**	**Z**	**U**
A	B	C	D
A	B	C	E
A	B	E	F
A	B	E	A
C	D	A	F

Θ-Illesztés (Θ-join)

Legyen R és S két reláció. Θ jelölje valamelyik aritmetikai hasonlító operátort. R és S reláció Θ-illesztésén az i, j pontban azt a relációt értjük, amely az R és S relációk Descartes-szorzatának az a részhalmaza, amelyre igaz, hogy az R-beli

n-es i-edik összetevője az adott hasonlító relációban áll az S beli m-es j-edik összetevőjével. Jelölése:

$$\underset{i\Theta j}{R \bowtie S} = \sigma_{i\Theta j}(RXS)$$

Erre példát az alábbi táblázatban láthatunk.

R

X	**Y**	**Z**
A	B	C
A	A	D
A	D	E
B	C	E
C	D	A

S

U	**V**	**W**
B	C	D
B	C	E
A	E	F
B	E	A
D	A	F

$R \bowtie S$ 2=1

X	**Y**	**Z**	**U**	**V**	**W**
A	B	C	B	C	D
A	B	C	B	C	E
A	B	C	B	E	A
A	A	D	A	E	F
A	D	E	D	A	F
C	D	A	D	A	F

1.5. Relációs adatbázis tervezése

A relációs adatbázis tervezésének két alapvető módja létezik. Első esetben elkészítjük az adatbázisunk egyed kapcsolati diagrammját, ebből kiindulva tervezzük meg az adatbázist.

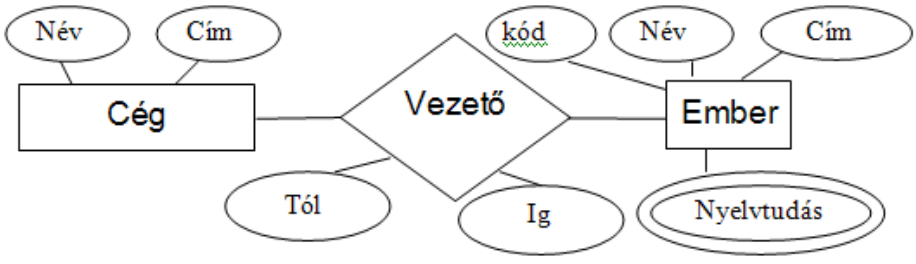
A másik módszer, hogy a tárolni kívánt adatokat egyetlen táblába tesszük, és ennek megfelelő szabályok szerinti szétbontásával hozzuk létre a kívánt relációs adatbázisunkat. Az alábbiakban ezeket nézzük meg részletesen.

1.5.1. Tervezés ER diagrammal

A szöveges megfogalmazásból megtervezzük az ER diagrammot. Az egyes entitásokhoz egy táblát hozunk létre, ahol az oszlopok az egyértékű attribútumok lesznek. Ha többértékű attribútum is van, akkor ehhez is létrehozunk egy táblát az entitás kulcsával együtt, és a kulcs-attribútum párosok adják az összetartozást. Végül, ha szükséges, a kapcsolathoz is rendelünk egy táblát. Nézzük meg az alábbiakban az egyes kapcsolat tipusoknak megfelelő táblák létrehozását.

1.5.1.1. Az egy-egy kapcsolat átalakítása

Az egy-egy kapcsolat esetén, ha a kapcsolatnak nincs attribútuma az egyik entitás tábláját kibővítjük a másik entitás kulcs attribútumával, ez jelöli majd a kapcsolatokat. (lásd az 1.7 ábrát illetve az alábbi táblákat.)



1.7. ábra. Az egy-egy kapcsolat átalakítása

Az EMBER entitás táblája:
EMBER (kód, Név,Fizetés)

kód	**Név**	**Cím**
153	Kovács Péter	Miskolc Arany J. u. 12
215	Vadász Tibor	Tiszaújváros Fő u.33
356	Asztalos Géza	Sopron Magyar u. 43

A cég entitás táblája:

cég (Név, Cím)

Cégnév	**Cím**	**kód**
Vegyiművek	Tiszaújváros Ipartelep 1	215
Acélgyár	Miskolc Vaskohász u. 1-3	153
Erdészet	Sopron Csengeri u 17	356

Ez a tábla módosul a kapcsolat miatt. Ha a kapcsolatnak nincs attribútuma: A cég entitás táblája, kiegészítve a kapcsolat másik résztvevőjének a kulcsával.

cég (Név, Cím, kód)

Cégnév	**Cím**	**kód**
Vegyiművek	Tiszaújváros Ipartelep 1	215
Acélgyár	Miskolc Vaskohász u. 1-3	153
Erdészet	Sopron Csengeri u 17	356

Tábla a többértékű nyelvtudáshoz:

nyelvtudás (kód, nyelv)

kód	**nyelv**
153	magyar
153	német
215	magyar
215	angol
356	magyar
356	német
356	francia

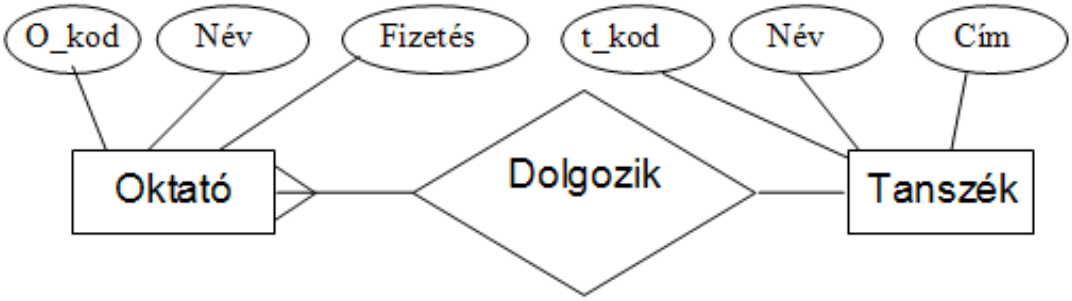
Ha a kapcsolatnak van attribútuma, akkor a kapcsolathoz is rendelünk egy táblát a két résztvevő kulcsával és a kapcsolat paraméterével. Ebben az esetben a cég tábláját nem bővítjük az ember tábla kulcsával.

Cégnév	**kód**	**Tól**	**Ig**
Vegyiművek	153	1990	1995
Vegyiművek	215	1990	1995
Acélgyár	153	1995	2005
Erdészet	356	1995	2003

Ey utóbbi táblát kapcsoló táblának is nevezik.

1.5.1.2. Az egy-több kapcsolat átalakítása

Egy-több kapcsolat esetén a több kapcsolat entitás tábláját kiegészítjük az egy-entitás kulcsával, ha a kapcsolatnek nincs paramétere. (Ha a kapcsolatnak van attribútuma, akkor a kapcsolathoz is egy kapcsoló táblát rendelünk), így jelöljük az öszszetartozást. Lásd az 1.8 ábrát és az alábbi táblákat.



1.8. ábra. A egy-több kapcsolat átalakítása

Az OKTATÓ entitás táblája kiegészítve a tanszék kódjával:
OKTATÓ (O_kod, Név, Fizetés, t_kod)

O_kod	**Név**	**Fizetés**	**t_kod**
932664	Nagy Tibor	134000	201
935122	Kiss István	162000	201
940312	Fábián Márton	134000	205

A TANSZÉK entitás táblája:
TANSZÉK (t_kod, Név, Cím)

t_kod	Név	Cím
201	Automatizálási és Alkalmazott Informatikai Tanszék	1111 Budapest Goldmann Gy. tér 3
205	Irányítástechnika és Informatika Tanszék	1111 Budapest Magyar Tudósok körútja 2

Ha a fenti példában a kapcsolatnak attribútumai is vannak, akkor kapcsoló táblát hozunk létre. Például, ha az attribútum a dátum, amikor a megadott fizetést kapta a dolgozó (a fizetés is a kapcsolat attribútuma lesz), akkor a dolgozik kapcsolatból is egy táblát kell készíteni

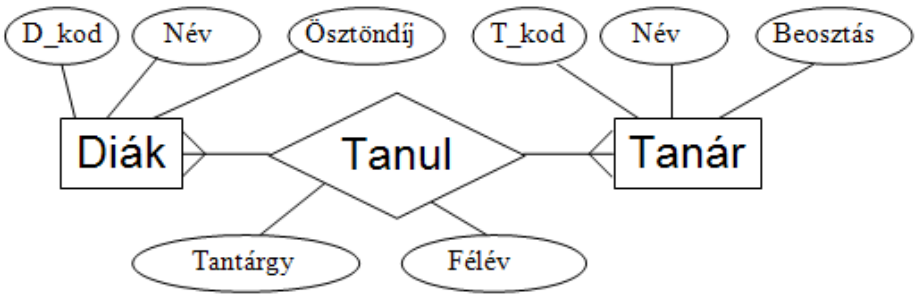
DOLGOZIK (O_kod, t_kod, Tól, Ig, Fizetés)

míg az OKTATÓ táblája, amelyből a fizetés kimarad

OKTATÓ (O_kod, Név) lesz.

1.5.1.3. A több-több kapcsolat átalakítása

Több-több kapcsolat esetén a kapcsolathoz is létrehozunk egy kapcsoló táblát, melybe felvesszük oszlopoknak a kapcsolatban résztvevők kulcsait, valamint a kapcsolat attribútumait (lásd az 1.9. ábrát és az alábbi táblákat.



1.9. ábra. A több-több kapcsolat átalakítása

A DIÁK entitás táblája:

DIÁK (D_kod, Név, Ösztöndíj)

D_kod	Név	Ösztöndíj
AR02ST	Nagy Tamás	12000
DK7H86	Kiss Péter	24300
KBCAS2	Oravecz Pál	17500

A TANÁR entitás táblája:

TANÁR (T_kod, Név, Beosztás)

T_kod	**Név**	**Beosztás**
HKR145	Kenderes Béla	Docens
S12BHG	Maklári Iván	Adjunktus
LJ87TG	Pados Gergely	Tanár

A Tanul kapcsolat táblája:

TANUL (D_kod, T_kod, Tantárgy, Félév)

D_kod	**T_kod**	**Tantárgy**	**Félév**
AR02ST	HKR145	Digitális technika 1	2005/06_1
AR02ST	S12BHG	Informatika1	2006/07_2
AR02ST	LJ87TG	Matematika	2006/07_1
DK7H86	HKR145	Digitális tervezés	2005/06_1
DK7H86	LJ87TG	Analízis	2006/07_2
KBCAS2	S12BHG	Informatika 2	2005/06_2

1.5.2. Tervezés sémadekompozícióval

A relációk tetszőleges számú tulajdonságokból építhetők fel. Elvileg a rendszerben található valamennyi adatot beépíthetjük egyetlen relációba. Ekkor egy tábla írja le az egész rendszert. Ezt univerzális relációnak nevezzük. Ez a reláció általában sok redundanciát tartalmaz, ezért a rendszert több relációból alakítjuk ki, biztosítva ez által a redundancia mentességet.

Azon adatokat, melyek valamely egyed jellemzője, alapadatnak nevezzük. Pl.: valakinek a születési dátuma. Azokat az adatokat, amelyek alapadatok alapján meghatározhatóak, származtatott adatoknak nevezzük. Pl.: milyen napra esik a születésnap. Redundancia: A feleslegesen többször tárolt alapadatok, vagy a tárolt származtatott adatok.

A redundancia részben az adatbázis méretét növeli meg, másrészt következetlenné, inkonzisztenssé teheti az adatbázist. (Pl.: a többször tárolt adatok egyikét már megváltoztattuk, amikor rendszerhiba miatt a tranzakció megszakad.) Nem minden többször tárolt adat jelent redundanciát.

Az univerzális relációra példát az alábbi táblázat mutat.

NÉV	TANSZÉK	TSZ_TELEFON	TSZ_VEZETŐ
TÓTH ISTVÁN	MATEMATIKA	1664-555	BOROS Z.
KISS JÁNOS	MATEMATIKA	1664-555	BOROS Z.
NAGY TIBOR	GÉPTAN	1345-233	KOVÁCS I.
VERES PÉTER	GÉPTAN	1345-233	KOVÁCS I.
NOVÁK BÉLA	HŐTAN	2313-454	ERDŐS G.

Itt a tanszék nevének tárolása szükséges, nem jelent redundanciát, azonban a többször tárolt tanszékvezető már igen.

1.5.2.1. Anomáliák

Relációinkat kénytelenek vagyunk függőlegesen felbontani (dekomponálni), mert minden logikailag összetartozó adatnak egyetlen sémában történő ábrázolása (univerzális reláció) problémákat vet fel.

Tárolási anomália

Mivel egy adat többször is szerepelhet, szükségtelenül pazaroljuk a tárat, pl.: tanszék-tanszékvezető több sorban is szerepel.

Módosítási anomália

Ugyanazt az adatot több helyen kell módosítani. Ez veszélyforrás, mert pl. ha megváltozik a GÉPTAN tanszék telefonja, ezt a 3. és a 4. sorban is javítani kell.

Beszúrási anomália

Nem tudunk olyan adatot nyilvántartásba venni, amely kell ugyan, de jelenleg nincs meg a hozzátartozó összes adat. Pl.: ha a egy új tanszéknek még nincs tanszékvezetője, vagy telefonja, akkor egyetlen munkatársat sem tudunk felvinni a relációba (feltéve, hogy ezen adatok megadása kötelező).

Törlési anomália

Mivel csak egész sorok törölhetők, elveszíthetünk olyan adatot is, amelyre még szükségünk lehet, de már nem marad belőle példány. Pl.: NOVÁK BÉLA kilép, eltünik a HŐTAN tanszék is.

1.5.3. Funkcionális függőség

Annak érdekében, hogy olyan felbontásokat tudjunk kialakítani, melyek használatával az előbb említett anomáliák csökkenthetők, vagy megszüntethetők, az attribútumok kapcsolatainak mélyebb vizsgálata szükséges.

1.5.3.1. A funkcionális függőség meghatározása

Legyen adott az $R(A_1, A_2, ...A_n)$ reláció, ahol A_i az attribútumokat jelöli. Legyen X és Y a relációk attribútumainak egy-egy részhalmaza.
$X \subseteq A_1, A_2, ...A_n$ és $Y \subseteq A_1, A_2, ...A_n$.

Jelöljük $R[XY]$-nal az R relációnak azon vetületét, amely csak az $X \cup Y$ attribútumokat tartalmazza. Az $R(A_1, A_2, ...A_n)$ reláción akkor és csak akkor áll fenn az $X \rightarrow Y$ funkcionális függőség, ha az idő minden pillanatában érvényes $R[XY]$-ban az $R[X] \rightarrow R[Y]$ leképezés. Más szavakkal a funkcionális függőség azt jelenti, hogy a tulajdonságok által meghatározott rendszerben egy (vagy több) tulajdonság egyértelműen meghatároz egy vagy több másik tulajdonságot.

Általánosan Y akkor függ funkcionálisan X-től, ha X minden azonos értékéhez hozzárendelhető Y egy és csakis egy értéke. A funkcionális függőségek meghatározása koncepcionális (modellezési) kérdés.

Alapvetően a funkcionális függőségek nem az adatbázis tervezőjétől függenek, az ő feladata ezen függőségek felderítése, elemzése. Maguk a függőségek biológiai, fizikai kapcsolatokon, törvényeken, jogszabályokon, rendeleteken alapulnak.
Pl. ha ismerjük egy személy adóazonosítóját(A_azns) , akkor tudjuk a születési dátumát(Sz_dat), azaz $A\_azns \rightarrow Sz\_dat$ fennáll.

Ha ismerjük egy gépkocsi rendszámát(R_szam) akkor tudhatjuk a tipuát($Tipus$) is, azaz $R\_szam \rightarrow Tipus$ fennáll.

Ha ismerjük egy anyag nevét(A_nev) akkor tudhatjuk a fajsulyát($Fajs$) is, azaz $A\_nev \rightarrow Fajs$ fennáll.

Ismerjük egy növény nevét(N_nev), ebből adódik, hogy milyen termése($Termes$) van ($N\_nev \rightarrow Termes$).

Az adatbázis tervezője is hozhat olyan "rendelkezést", feltételt, amely következtében kialakulhat ilyen funkcionális függőség. Problémát okoz azonban, ha az adatbázis használója ezt nem veszi figyelembe, ami könnyen előfordulhat. Ugyan-

igy problémát okoz pl. ha egy gépkocsi rendszámot többször is kiadnak, hiszen eredeti elképzelés szerint a rendszám (R) azonosítja a gépkocsit(G), azaz $R \rightarrow G$, viszont ha egy rendszám több gépkocsin is azonos, akkor ez a függőség már nem igaz. Pl.: TANSZÉK $\rightarrow$ TSZ.TELEFON (feltéve, hogy a tanszéknek csak egy telefonja van, illetve a modellalkotásnál eldöntjük, hogy csak egyet tárolunk, akkor a funkcionális függőség igaz, egyébként nem). Viszont TSZ.TELEFON $\rightarrow$ TANSZÉK mindig igaz (Ugyanis egy telefonszám nem rendelhető több tanszékhez, amit a modellezésnél kihasználhatunk, mert ez postai előírás, ugyanazt a számot nem lehet kiadni két előfizetőnek).

Ha $X \rightarrow Y$ fennáll és Y nem függ funkcionálisan X egyetlen részhalmazától sem, akkor X-et Y determinánsának nevezzük.

Nézzük meg a függőségeket egy konkrét reláció esetén:

Pl.: R(SZEMÉLYISZÁM, NÉV, CÍM, VÁROS, IRÁNYÍTÓSZÁM, TELEFON) relációs sémában a valóságot jól modellező funkcionális függőségek az alábbiak: (a SZEMÉLYISZÁM igazából egy személyt azonosít)

- SZEMÉLYISZÁM $\rightarrow$ NÉV
- SZEMÉLYISZÁM $\rightarrow$ CÍM
- SZEMÉLYISZÁM $\rightarrow$ VÁROS
- SZEMÉLYISZÁM $\rightarrow$ IRÁNYÍTÓSZÁM
- SZEMÉLYISZÁM $\rightarrow$ TELEFON
- (VÁROS, CÍM) $\rightarrow$ IRÁNYÍTÓSZÁM
- IRÁNYÍTÓSZÁM $\rightarrow$ VÁROS

1.5.3.2. Kulcsok

A fizikai modelleknél már használtunk egy kulcs fogalmat. Ott azt mondtuk, kulcs minden, ami szerint keresni tudunk. Most megadjuk egy reláción értelmezett kulcs matematikai definícióját.

Legyen adott az $R(A_1, A_2, ...A_n)$ reláció, ahol A_i az attribútumokat jelöli. Legyen X a relációk attribútumainak részhalmaza: $X \subseteq A_1, A_2, ...A_n$. X-et akkor és csak akkor nevezzük kulcsnak az R reláción, ha

- X meghatározza R valamennyi attribútumát, vagyis $X \rightarrow A_i$ fennáll, ahol $i = 1, 2, ...n$ és

- X-nek nincs olyan valódi részhalmaza, amely meghatározza R valamennyi attribútumát, vagyis X-ből bármit elhagyva már nem teljesíti az első feltételt, vagyis nem létezik $X' \subset X$; $X' \rightarrow A_j$, $j = 1, 2, ...n$. Más szavakkal X minimális kulcs.

Szuperkulcs, kulcs

X-et szuperkulcsnak nevezzük az R relación, ha a kulcsokra vonatkozó két kritérium közül csak az elsőt teljesíti, azaz lehet olyan részhalmaza amely kulcs.
Definíció: Legyen adott az R reláció a T tulajdonságok halmaza felett. A tulajdonságok egy olyan K részhalmazát, melynek értékei az R reláció egy sorát egyértelműen meghatározzák, a reláció szuperkulcsának nevezzük. Ha K szuperkulcs, de $K' \subset K$ már nem az, akkor K minimális kulcs, vagy egyszerűen kulcs.
Ha K egy tulajdonságból áll, akkor **egyszerű kulcs**, egyébként **összetett kulcs**.
Egy kulcs az alábbi tulajdonságokkal kell, hogy rendelkezzen:

- a kulcs a relációnak egy és csakis egy sorát határozza meg
- a kulcsnak nincs olyan részhalmaza, amely szintén kulcs lenne
- a kulcs tulajdonságok nem lehetnek NULL-értékűek
- minden relációnak van kulcsa
- egy relációnak több kulcsa is lehet.

Minden relációnak van kulcsa

Legyen adott a korábban látott R reláció. Válasszuk X-nek az attribútumok teljes halmazát. Ez a kulcsokra vonatkozó első feltételnek eleget tesz, hiszen nincs olyan attribútum, amit ne vettünk volna figyelembe. (A relációnak nincs két azonos sora, egy sor saját magát pedig egyértelműen azonosítja.) Ha a második feltétel teljesül, akkor kulcs, ha pedig nem, akkor szuperkulcs, tehát tartalmaz kulcsot.

Elsődleges kulcs

Ha egy relációnak több kulcsa is van, kiválasztunk egyet, amelyet használni fogunk, ezt elsődleges kulcsnak nevezzük. A többi kulcsot kulcsjelöltnek hívjuk.
Pl. egy személy esetén kulcs lehet a tajszám, az adószám, a személyi szám, a név, születési dátum, anyja neve, cím együttes is. Ebből pl. az adóhatósgnál elkészíett rendszerben elsődleges kulcs lesz az adószám, a többi kulcsjelölt. A

név, születési dátum, anyja neve, cím együttes sehol nem lesz elsődleges kulcs, mert egyedi ugyan, de túl sok helyet foglal, mindenütt bevezetnek helyette egy rövidebb "kód"-ot, amit kulcsként fognak használni.

Idegen kulcs

Egy relációban lehetnek olyan tartományok, amelyek másik relációban a sorokat egyértelműen azonosítják, tehat ott kulcsok, de ebben a relációban nem. Ezeket idegen kulcsoknak nevezzük. Ezekkel tudjuk az egyik reláció adatait a másikkal összekapcsolni. Ezek értelemszerűen mindkét relációban megjelennek, ezért lényeges, hogy az elsődleges kulcsok lehetőleg egyszerűek legyenek.

1.5.3.3. Funkcionális függőségek tulajdonságai

Armstrong három axiómája van a funkcionális függőségről.

Reflexivitás

Ha ugyanazon R reláción

$Y \subseteq X$, *akkor* $X \rightarrow Y$.

Szokás triviális függőségnek nevezni. Pl.: Ha valakinek ismerem a nevét (vezetéknév, keresztnév) akkor ismerem a keresztvnevét.

(vezetéknév, keresztnév) $\rightarrow$ keresztnév

Tranzitivitás

Ha ugyanazon R reláción

$X \rightarrow Y$ *és* $Y \rightarrow Z$, akkor $X \rightarrow Z$. Pl.:

osztály $\rightarrow$ osztályfőnök

osztályfőnök $\rightarrow$ osztályfőnök_telefonszáma,

akkor

osztály $\rightarrow$ osztályfőnök_telefonszáma,

szavakban: ha ismerem az osztályhoz tartozó osztályfőnököt, az osztályfőnöknek pedig ismerem a telefonszámát, akkor az osztályhoz hozzá tudom rendelni az osztályfőnök telefonszámát.

Bővíthetőség

Ha ugyanazon R reláción

$X \to Y$, *akkor* $XZ \to YZ$

Ha a személyi szám alapján tudom, hogy az illető neve Tóth István, akkor a személyi szám és (pl.) a szeme színe alapján tudom a nevét és a szeme szinét.

A fentiek axiómák, nem kell őket bizonyítani, de logikailag, a példákkal illusztrálva könnyen érthetőek.

1.5.3.4. A tulajdonságok következményei

A funkcionális függőségek tulajdonságaiból további összefüggések vezethetők le. Más szavakkal az axiómákból tételek vezethetők le.

Egyesítési szabály

Ha	$X \to Y$ *és* $X \to Z$,	akkor $X \to YZ$
Bizonyítás:	$X \to Y \Rightarrow XX \to XY$	bővíthetőség alapján
és	XX=X miatt $X \to XY$	
és	$X \to Z \Rightarrow XY \to ZY$	bővíthetőség alapján,
ezután	$X \to XY$ és	
	$XY \to ZY \Rightarrow X \to YZ$	a tranzitivitás alapján.

Pszeudotranzitivitás

Ha	$X \to Y$ *és* $YW \to Z$,	akkor $XW \to Z$
Bizonyítás:	$X \to Y \Rightarrow XW \to YW$	bővíthetőség alapján,
ezután	$XW \to YW$ és	
	$YW \to Z \Rightarrow XW \to Z$	a tranzitivitás alapján.

Dekompozíciós szabály

Ha	$X \to Y$ *és* $Z \subseteq Y$,	akkor $X \to Z$
Bizonyítás:	$Z \subseteq Y \Rightarrow Y \to Z$	a reflexivitás miatt,
majd	$X \to Y$ és	
	$Y \to Z \Rightarrow X \to Z$	a tranzitivitás alapján.

1.5.3.5. A függőséghalmaz lezárása

A relációs adatbázis tervezésénél meg kell adni az egyes relációkhoz tartozó függőségeket. Ezeket a reláció függőséghalmazának (F) nevezzük.

A megadott függőségekből az Armstrong axiómák felhasználásával további függőségeket tudunk levezetni. Ha az összes levezethető függőséget is belevesszük a függőséghalmazba, akkor a **függőséghalmaz lezártjáról** beszélünk. Ezt F^+-al jelöljük.

Formálisan: $F^+ = \{X \to Y | F \Rightarrow X \to Y\}$, azaz F^+-nak eleme minden olyan függőség, amely F-ből következik az Armstrong axiómák alapján. Az F^+ igen nagy lehet.

Pl.: $F = \{X \to Y,\ Y \to Z\}$

akkor $F^+ = \{X \to Y,\ Y \to Z,\ X \to X, Y \to Y,\ Z \to Z,\ X \to Z,\ X \to XY,$
$Y \to YZ,\ X \to XZ,\ XY \to Z, XY \to XY,\ XYZ \to XYZ ...\}$

A függőséghalmaz lezártjával lehetőségünk van eldönteni két függőséghalmaz azonosságát. Két függőséghalmaz akkor azonos, ha lezártjai megegyeznek.

Sokszor szükségünk van arra, hogy egy függőséghalmazból a lehető legkisebb halmazt határozzuk meg, amely még egyenlő az eredeti függőséghalmazzal. Ezt minimális függőséghalmaznak nevezzük.

Egy **függőséghalmaz minimális**, ha

- nem hagyható el belőle függőség,
- a függőségek jobb oldalán csak egyetlen attribútum van és
- a függőségek bal oldaláról nem hagyható el attribútum.

A minimális függőséghalmaz nem egyértelmű, egy függőséghalmazhoz tartozhat több minimális függőséghalmaz is. Pl.:

$F = \{A \to B,\ B \to A,\ A \to C\ B \to C\}$ függőséghalmaz két különböző minimális függőséghalmaza:

$F1 = \{A \to B,\ B \to A,\ A \to C\}$ és $F2 = \{A \to B,\ B \to A,\ B \to C\}$

1.5.3.6. Az attribútumhalmaz lezárása

Két függőséghalmaz ekvivalenciáját úgy is megállapíthatjuk, hogy az egyikben létező bármelyik függőség a másikban is benne van, és fordítva, azaz veszünk egy függőséget az egyik halmazból, és ez benne van a másikban, vagy az ott lévő

függőségekből levezethető. Ennek egyszerű meghatározásához használhatjuk az attribútum halmaz lezártját.

Az X attribútum halmaz lezártja az a $Z \subseteq \Omega$ attribútumhalmaz (Ω a teljes attribútum halmaz), amelyre igaz, hogy az $X \rightarrow Z$ benne van az F függőséghalmazban, vagy abból levezethető az Armstrong axiómák felhasználásával. Ezt X^+-al jelöljük.

Egy attribútumhalmaz lezártja viszonylag gyorsan meghatározható.
Induláskor:
$X^{+(0)}$=X, ahol a felső index az X^+ meghatározásához tartozó lépésszámot jelöli. A következő lépésekben vegyünk egy olyan függőséget, melynek bal oldala részhalmaza az $X^{+(i)}$-nak, azaz $X_R \subseteq X^{+(i)}$, a föggőség jobb oldalát pedig jelöljük Y-nal, azaz F-ben létezik az $X_R \rightarrow Y$ függőség. Vegyük hozzá az eddigi $X^{+(i)}$ lezárthoz Y-t, azaz $X^{+(i+1)} = X^{+(i)} \cup Y$. Ezt addig folytassuk, amig bármelyik X_R -hez van olyan függőség, melynek jobb oldala még nincs bent az $X^{+(i)}$ -ben.
Formálisan:
$X^{+(i+1)} = X^{+(i)} \cup \{Y \mid X_R \subseteq X^{+(i)}$, és $X_R \rightarrow Y \subseteq F\}$
Példa: Legyen az R(A,B,C,D,E,F) reláció, ennek függőséghalmaza F={A→B, B→C, DE→F}. Nézzük meg, mi az egyes attribútumok és az egyes függőségek bal oldalának lezártjai!

$A^{+(0)}$=A

$A^{+(1)}$=AB , az A→B miatt.

$A^{+(2)}$=ABC, a B→C miatt.

Mivel az ABC egyetlen további részhalmaza sem szerepel a függőségek bal oldalán, így készen vagyunk, azaz A^+=ABC.

$B^{+(0)}$=B

$B^{+(1)}$=BC , a B→C miatt.

Mivel az BC egyetlen további részhalmaza sem szerepel a függőségek bal oldalán, így készen vagyunk, azaz B^+=BC.

$C^{+(0)}$=C , nem szerepel a függőségek bal oldalán, így készen vagyunk.

$D^{+(0)}$=D , nem szerepel a függőségek bal oldalán, így készen vagyunk.

$E^{+(0)}$=E , nem szerepel a függőségek bal oldalán, így készen vagyunk.

$F^{+(0)}$=F , nem szerepel a függőségek bal oldalán, így készen vagyunk.

Nézzük meg, mi lesz az DE lezártja.

$DE^{+(0)}$=DE

$DE^{+(1)}$=DEF , az DE→F miatt.

Mivel az DEF egyetlen további részhalmaza sem szerepel a függőségek bal oldalán, így készen vagyunk.

A továbbiakban nézzük meg, mi lesz az ADE lezártja.

$ADE^{+(0)}$=ADE

$ADE^{+(1)}$=ADEB , az A→B miatt.

$ADE^{+(2)}$=ADEBC, a B→C miatt.

$ADE^{+(3)}$=ADEBCF , a DE→F miatt.

Itt $ADE^{+(3)}$=ABCDEF tartalmazza a reláció összes attribútumát, készen vagyunk.

Azt is megállapíthatjuk, hogy ADE a reláció szuperkulcsa, ami itt egyben kulcs is, mert ADE bármelyik részhalmaza nem kulcs, azaz ADE minimális. (az, hogy a részhalmazok nem kulcsok, láttuk, hiszen A^+ =ABC nem kulcs és D^+ =D, E^+=E, AD^+=ABCD, AE^+=ABCE, DE^+=DEF szintén nem kulcsok).

A reláció kulcsát az egyes attribútumok és a függőségek bal oldalának lezártjaiból könnyen megtalálhatjuk. Kell venni azoknak a lezártaknak az unióját, melyek jobb oldalának unió jában az összes attribútum megtalálható, akkor ez biztosan szuperkulcs. Ebben az esetben a bal oldalak uniójának lezártja XY+ egyenlő a jobb oldalak uniójával $X^+ \cup Y^+$. (Általában, ha az attribútumok és a determinánsok lezártjaiból indulunk ki, igaz, hogy $X^+ \cup Y^+ = XY^+$ feltéve, hogy nem létezik olyan $Z \subseteq XY$ és $Z \not\subseteq X$ és $Z \not\subseteq Y$, amely determináns, egyébként $X^+ \cup Y^+ \subseteq XY^+$)

Példa: R(A,B,C,D,E,F) F={AB→D, AC→E, BC→F}

Ekkor

AB^+=ABD

AC^+=ACE

BC^+=BCF

$AB^+ \cup AC^+$=ABCDE

ABC^+=ABCDEF (BC→F miatt)

Szintén gyakori feladat, annak eldöntése, hogy egy függőség következik-e az eredeti függőséghalmazból, pl.: az $X \rightarrow Y$ benne van-e az F-ben. Az eldöntés

egyik módja, hogy előállítjuk F^+-t(ez nagyon munkaigényes), és megnézzük, hogy ez tartalmazza-e a keresett függőséget. Másik lehetőség, hogy vesszük a bal oldal lezártját, azaz meghatározzuk X^+-t. Ha ennek jobb oldalán szerepel Y, akkor $X \rightarrow Y$ benne van az F^+-ban.

1.5.4. Relációk felbontása

Legyen adott egy $R(A)$ relációs séma, valamint az $X_1, X_2, ...X_n$ attribútumok halmaza, ahol $X_i \subset A$ és $X_i \cap X_j \neq 0$ és $\cup X_i = A$, akkor az $R(A)$ relációs sémának függőleges felbontásai a $\rho(R_1(X_1), R_2(X_2), ...R_n(X_n))$ részsémák.

Egy reláció tetszőleges függőleges felbontásakor információt veszíthetünk, ami abban nyilvánulhat meg, hogy új sorok is keletkezhetnek a relációk újra egyesítéskor.

Az $R(A)$ reláció séma $\rho(R_1(X_1), R_2(X_2), ...R_n(X_n))$ felbontását veszteségmentesnek mondjuk, ha
$R_1 \bowtie R_2 \bowtie ... \bowtie R_n$ = R,
azaz természetes illesztéssel újraegyesítve a részrelációkat, az eredeti relációt kapjuk vissza, tehát új sor nem keletkezik. A séma felbontásnál az egyes részsémáknak vannak közös attribútuma, különben nem tudnánk egyesíteni ezeket.

1.5.4.1. Veszteségmentes dekompozíció

Tétel: Az $R(A)$ reláció $\rho(R_1(X_1), R_2(X_2))$ ($X_1 \subset A$ és $X_2 \subset A$) dekompozíciója akkor és csak akkor veszteségmentes, ha a reláció sémáira az
$(X_1 \cap X_2) \rightarrow (X_1 - X_2) \subseteq F$ vagy
$(X_1 \cap X_2) \rightarrow (X_2 - X_1) \subseteq F$ teljesül.
Az $(X_1 \cap X_2)$ nem lehet üres, azaz kell lennie közös attribútumnak a két részreláció sémájában.

1.5.4.2. Példa dekompozíciókra

Legyen adott R(X,Y,Z) reláció, melynek egyetlen függősége Z→X, és képezzük az R_1(X,Y)-t és R_2(Y,Z)-t felbontást, majd egyesítsük R'(X,Y,Z)-vé.

Látni fogjuk, hogy könnyű olyan példát találni, hogy R(X,Y,Z) $\neq R'$(X,Y,Z), azaz a felbontás veszteséges:

R(X,Y,Z) R'(X,Y,Z)

(Z → X) R_1(X,Y) R_2(Y,Z) (Z → X) nem igaz

X	**Y**	**Z**
a	c	e
a	d	f
b	c	g
b	d	h

X	**Y**
a	c
a	d
b	c
b	d

Y	**Z**
c	e
d	f
c	g
d	h

X	**Y**	**Z**
a	c	e
a	c	g
a	d	f
a	d	h
b	c	e
b	c	g
b	d	f
b	d	h

A következő felbontás veszteségmentes, azaz R(X,Y,Z) = R'(X,Y,Z).

R(X,Y,Z) R'(X,Y,Z)

(Z→X) R_3(X,Z) R_4(Y,Z) (Z→X) igaz

X	**Y**	**Z**
a	c	e
a	d	f
b	c	g
b	d	h

X	**Z**
a	e
a	f
b	g
b	h

Y	**Z**
c	e
d	f
c	g
d	h

X	**Y**	**Z**
a	c	e
a	d	f
b	c	g
b	d	h

Ugyanis R(X,Y,Z) felbontása R_3(X,Z) és R_4(Y,Z), akkor XZ∩YZ = Z és XZ-YZ = X és fennáll a Z→X.

Példa:

Legyen egy relációs séma az R(A,B,C) és a hozzá tartozó függőségi halmaz F={A→B, B→C}. Nézzük meg, hogy a ρ_1(AB, AC), a ρ_2(AB, BC) iletve a ρ_3(AC, BC) felbontások veszteségmentesek-e!

A ρ_1(AB, AC) felbontás esetén AB∩AC=A és az AB-AC=B, az A→B szerepel F -ben, tehát a felbontás veszteségmentes.

A ρ_2(AB, BC) felbontás esetén AB∩BC=B és az AB-BC=A, az B→A nem szerepel F -ben, ugyanakkor a BC-AB=C és a B→C szerepel F -ben, tehát a felbontás veszteségmentes.

A ρ_3(AC, BC) felbontás esetén AC∩BC=C és az AC-BC=A, az C→A nem szerepel F -ben, valamint a BC-AC=B és a C→B szintén nem szerepel F -ben, tehát a felbontás nem veszteségmentes.

Ha a fenti példát megfigyeljük, megállapíthatjuk, hogy ha a felbontásban a közös attribútum valamelyik függőség determinánsa, akkor a felbontás lehet veszteségmentes, ha legalább az egyik irányú különbség a kiválasztott függőség jobb oldalát tartalmazza. Ezt úgy is megfogalmazhatjuk, hogy a **felbontás veszteségmentes**, ha

- *az egyik részreláció attribútumai éppen egy függőség attribútumai,*
- *a másik részreláció tartalmazza a függőség determinánsát, de nem tartalmazza annak jobb oldalát.*

Formálisan: R(A,B,C,..) F={A→B} felbontása A→B szerint

R_1(A,B), R_2(Ω-B),

ekkor $R_1 \cap R_2$=A és R_1-R_2=B és A→B $\subseteq F$ teljesül.

1.5.4.3. Függőségőrző felbontások

Egy relációs séma veszteségmentes felbontása eredményezheti, hogy a részsémákban nem tudjuk többé az eredeti sémában érvényes függőségeket alkalmazni. Ennek következtében a rész-relációinkba nem megengedett adatok is bekerülhetnek. Célszerű ezért olyan sémákat konstruálni, amelyekre a funkcionális függéseket vetítve, a vetített függésekből az eredeti függőségek (az Armstrong axiómák segítségével) helyreállíthatók. Ekkor a felbontás függőségőrző.

Nézzünk meg egy példát!

Legyen a relációnk R(A,B,C), ahol a függőségek F = {AB→C, C→B}. Ahhoz, hogy veszteségmentes legyen a felbontás, az előzőek szerint egy függőséget célszerű belevenni az egyik részsémába (C→B), azaz R_1(B,C) legyen. Ekkor R_2(A,C) adódik (a függőség jobb oldala ne legyen benne). Veszteségmentes, mert

(BC∩AC = C)→ (BC-AC=B).

Ugyanakor egyik részrelációban sem szerepel az AB→C függőség. Igy a rendszerben olyan adat bevitelét közvetlenül nem tudjuk ellenőrizni, amely esetleg ezt a függőséget megsérti.

Erre egy gyakorlati példa lehet az R(CÍM, VÁROS, IRÁNYÍTÓSZÁM) , ahol
f_1 = (CÍM, VÁROS) → IRÁNYÍTÓSZÁM
f_2 = IRÁNYÍTÓSZÁM → VÁROS
R_1 (IRÁNYÍTÓSZÁM, VÁROS) , f_2 alapján ellenőrizhető
R_2 (IRÁNYÍTÓSZÁM, CÍM) , nincs ellenőrzési lehetőségünk, csak, ha minden bevitelnék (ideiglenesen) előállítjuk az eredeti relációt, és abban ellenőrzünk.

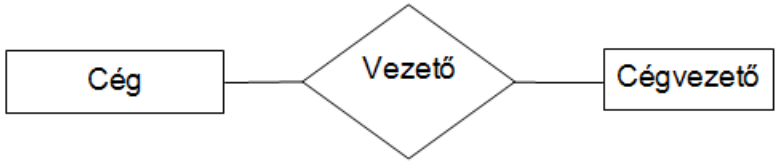
Összefoglalásként megállapíthatjuk, hogy egy felbontás lehet:

- veszteségmentes és függőségőrző,
- veszteségmentes és nem függőségőrző,
- nem veszteségmentes(veszteséges) és függőségőrző,
- nem veszteségmentes és nem függőségőrző.

Az utóbbi kettő nem használható, mert újraegyesítésnél új sorok jönnek be.

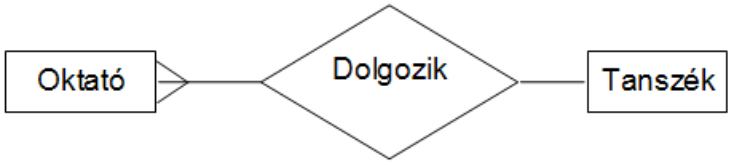
1.5.4.4. A függőségek kapcsolata az ER diagrammal

Az egy-egy kapcsolat esetén mindkét irányú függőség fennáll. Pl:



Cég → Cégvezető
Cégvezető → Cég

Több-egy kapcsolat esetén csak a *több* → *egy* függőség áll fenn. Pl.:



Oktató → Tanszék.

Több-több kapcsolat esetén nem lehet függőséget meghatározni.

1.5.5. A relációk normal formái

A tervezés során a relációk különböző felbontásban lehetnek.

1.5.5.1. A 0. és 1. normál forma (0NF, 1NF)

A relációk normál alakjai közül a nulladiknak és elsőnek nincs gyakorlati jelentősége.

0NF alakúnak kell tekintenünk minden olyan relációt, amelyben az attribútumok nem atomiak, vagy ismétlődő csoport van az attribútumok között.

1NF alakú, ha csak egyszerű tulajdonság-értékek szerepelnek benne.

Az alábbi relációk 0NF alakúak.

egyetem
rektor
kar
dékán
tanszék
tanszékvezető

R1 (E,R(K,D,(T,V)*)*)

A * jelű csoportok ismétlődnek.

R2 (személyi szám, név, cim), ahol a cím tartalmazza a város, irányitószám utcanév és házszám információt. Amennyiben a cím részletei is érdekelnek bennünket, akkor a cím nem atomi attribútum, tehát a reláció 0NF-ben van.

1.5.5.2. A 2. normál forma (2NF)

Definíció: Az R reláció A_i attribútuma elsődleges, ha A_i eleme a reláció valamely K kulcsának, egyébként a A_i másodlagos.

Definíció: Egy relációs séma 2NF alakú, ha benne minden másodlagos attribútum a reláció valamely kulcsától teljesen függ, azaz nincs benne függőség részkulcstól.

Tekintsük azt a relációt, amelyben egy hallgatóról a következő adatokat tartjuk nyilván:

a hallgató nevét, a félévet, amelyben a hallgató egy tanárnál - aki egy tanszéken dolgozik - egy tárgyat hallgat, és ebből osztályzatot is kap.

Legyen ez az R reláció, és a felsorolt attribútumok rendre HALLGATÓ, FÉLÉV, TANÁR, TANSZÉK, TANTÁRGY, OSZTÁLYZAT . Ekkor a reláció sémája a következő:

R(HALLGATÓ, FÉLÉV, TANÁR, TANSZÉK, TANTÁRGY, OSZTÁLYZAT)

A reláció függőségei:

- (HALLGATÓ, FÉLÉV, TANTÁRGY) → OSZTÁLYZAT
- TANTÁRGY → TANÁR (egy tantárgyat egy tanár tanít)
- TANÁR →TANSZÉK (egy tanár egy tanszéken dolgozik)

A reláció nincs 2NF alakban, mert a reláció kulcsa a HALLGATÓ, FÉLÉV, TANTÁRGY attribútum hármas, ugyanakkor a TANTÁRGY -tól függ a TANÁR , ami részkulcstól való függést jelent.

Ha felbontjuk a relációt két relációra, R_1 és R_2 -re, azaz

R_1(HALLGATÓ, FÉLÉV, TANTÁRGY, OSZTÁLYZAT)

R_2(TANTÁRGY, TANÁR, TANSZÉK)

Ekkor az R_1 reláció már 2NF alakú, mert az egyetlen másodlagos attribútuma a teljes kulcstól függ. R_2 is 2NF alakú, mert egyszerű kulcsa van (TANTÁRGY), így részkulcstól nem lehet benne függőség.

1.5.5.3. A 3. normál forma (3NF)

Definíció: Egy R relációs séma 3NF alakú, ha minden $X \to A$ függőség esetén

- vagy X szuperkulcsa R-nek,
- vagy A elsődleges attribútum.

Más megfogalmazásban: Egy reláció 3NF alakú, ha benne egyetlen másodlagos attribútum sem függ tranzitíven valamelyik kulcstól.

Tekintsük az előbbi, már felbontott relációkat.

R_1(HALLGATÓ, FÉLÉV, TANTÁRGY, OSZTÁLYZAT)

R_2(TANTÁRGY, TANÁR, TANSZÉK) , az előbbiek szerint ennek függőségei:

- TANTÁRGY → TANÁR
- TANÁR →TANSZÉK

Itt R_1 már 3NF alakú, mert egyetlen másodlagos attribútuma van. R_2 viszont nem 3NF alakú, mert a TANSZÉK függ a TANÁR -tól, így tranzitíven függ a TANTÁRGY -tól, ami kulcs.

Bontsuk fel R_2-t két relációra, R_3 és R_4-re.

R_3(TANTÁRGY, TANÁR)

R_4(TANÁR, TANSZÉK) .

Itt R_3 és R_4 is 3NF alakú.

1.5.5.4. A Boyce-Codd normál forma (BCNF)

Definició: Egy reláció BCNF alakú, ha benne nem triviális függőség csak szuperkulcstól van, azaz minden $X \to A$ nemtriviális függőség esetén X szuperkulcs.

Nézzük az R (CÍM, VÁROS, IRÁNYÍTÓSZÁM)relációt, ahol a függőséghalmaz F ={(CÍM, VÁROS) → IRÁNYÍTÓSZÁM, IRÁNYÍTÓSZÁM → VÁROS }. Ez nem BCNF alakú, mert IRÁNYÍTÓSZÁM → VÁROS determinánsa nem szuperkulcs.

Eszerint az R (CÍM, VÁROS, IRÁNYÍTÓSZÁM) reláció 3NF de nem BCNF. Ugyanakkor a fenti R_1, R_3, R_4 relációk 3NF és BCNF alakúak is.

Bontsuk fel az R relációt veszteségmentesen IRÁNYÍTÓSZÁM → VÁROS függőség szerint. Ekkor

R_1(IRÁNYÍTÓSZÁM, VÁROS), F_1={IRÁNYÍTÓSZÁM → VÁROS}

R_2(IRÁNYÍTÓSZÁM, CÍM), F_2={} relációkat kapjuk.

Ezek már BCNF alakúak, ugyanakkor (CÍM, VÁROS) → IRÁNYÍTÓSZÁM függőség nem szerepel a részsémákban.

1.5.6. Sémadekompozíciók

Tétel: Minden normalizált relációnak létezik veszteségmentes és függőségőrző felbontása, amely 3NF.

Tétel: Minden normalizált relációnak létezik veszteségmentes felbontása, amely BCNF.

1.5.6.1. A 3NF alakra hozás

A 3NF alakra hozás mindig veszteségmentes és függőségőrző felbontást jelent. A módszere a következő:

- Képezzük a függőséghalmaz minimális lefedését, legyen ez G (lásd: 1.5.3.5. fejezet).
- Minden $X \to Y \subset G$ függőséghez készítsünk egy $R_i(XY)$ részsémát;

- Egy K kulcs attribútumaiból is képezzünk egy további részsémát, ha K összetett kulcs és nincs benne valamelyik $\mathrm{R}_i(XY)$ részsémában.

Példa:

Egy relációs séma:

R(C,T,H,R,S,G), ennek függőséghalmaza F={ C→T, HR→C, HT→R, CS→G, HS→R}.

Hozzuk 3NF alakra!

F minimális, ezért a felbontás függőségekhez rendelt részsémákkal $\rho(R_1(CT), R_2(HRC), R_3(HTR), R_4(CSG), R_5(HSR))$. A reláció kulcsa HS, ($HS^+$=HSRCTG, az 5,2,1,4 függőségek miatt). Ezt a HS részsémát még hozzá kell venni a felbontáshoz, ha ez még nem szerepel valamelyik részsémában. Viszont ez az utolsó részsémában(HSR) már szerepel, így ezt nem vesszük hozzá, a megoldás a már előállított felbontás.

Példa:

Legyen adott az alábbi reláció:

R_1(TANSZÉKKÓD, TANSZÉKNÉV, TANSZÉKVEZETŐ, SZEMÉLYISZÁM, SZEMÉLYNÉV, FOKOZATKÓD, TÉMASZÁM, TÉMANÉV, ELFOGLALTSÁG)

Ha egy személy több témán is dolgozhat, akkor az R_1 reláció kulcsa SZEMÉLYISZÁM, TÉMASZÁM lehet, ez ugyanis egyértelműen meghatároz egy sort a relációban. A reláció függőségei ekkor

- (SZEMÉLYISZÁM, TÉMASZÁM) → ELFOGLALTSÁG
- TÉMASZÁM → TÉMANÉV
- SZEMÉLYISZÁM → TANSZÉKKÓD
- TANSZÉKKÓD → TANSZÉKNÉV, TANSZÉKVEZETŐ
- SZEMÉLYISZÁM → SZEMÉLYNÉV, FOKOZATKÓD

A második normál alakban az oszlopok csak a teljes elsődleges kulcstól függenek, tehát a relációk:

R_2(SZEMÉLYISZÁM, TÉMASZÁM, ELFOGLALTSÁG)

R_3(TÉMASZÁM, TÉMANÉV)

R_4(SZEMÉLYISZÁM, TANSZÉKKÓD, TANSZÉKNÉV, TANSZÉKVEZETŐ, SZEMÉLYNÉV, FOKOZATKÓD)

Harmadik normál alakot akkor kapunk, ha az R_4-ből kivesszük a tranzitív függőséget, azaz
személyiszám → tanszékkód → (tanszéknév, tanszékvezető) figyelembevételével

R_5(<u>személyiszám</u>, tanszékkód, személynév, fokozatkód)

R_6(<u>tanszékkód</u>, tanszéknév, tanszékvezető).

A kapott 3NF alakú reláció az R_2, R_3, R_5, R_6 lesz.

1.5.6.2. A BCNF alakra hozás

BCNF alakra tudjuk hozni a relációkat veszteségmentes dekompozícióval. Itt a függőségőrzés nem biztosított.
A BCNF-re hozás menete a következő:

- 1. Megszerkesztjük az univerzális relációt.
- 2. Meghatározzuk a funkcionális függőségeket.
- 3. Eldöntjük, hogy a reláció BCNF-e, ha igen, kész.
- 4. Ha valamely $X \to A$ miatt nem az, akkor a relációt két relációra bontjuk szét veszteségmentes dekompozícióval $X \to A$ szerint
- 5. Megismételjük a 3. és 4. lépéseket az új relációkra.

Példa:

r(<u>diakkod</u>, diaknev, ofonok, ofotel, <u>nyelv</u>, <u>felevkod</u>, osztalyzat)

Tekintettel arra, hogy a fenti reláció univerzális, ezért a feladat megoldását a második lépéssel kezdhetjük, azaz meghatározzuk a függőségeket.
2. lépés
A függőségek

- diakkod → diaknev
- diakkod → ofonok
- diakkod → ofotel
- ofonok → ofotel
- ofotel → ofonok
- (diakkod, nyelv, felevkod) → osztalyzat

3. lépés
A relációnak három nem kulcsjelölt determinánsa van

- diakkod

- OFONOK
- OFOTEL

Az R reláció nem BCNF pl. az OFONOK → OFOTEL függőség miatt.
Bontsuk fel e szerint két relációra:

R_1(<u>OFONOK</u>, OFOTEL)

R_2(<u>DIAKKOD</u>, DIAKNEV, OFONOK, <u>NYELV</u>, <u>FELEVKOD</u>, OSZTALYZAT)

R_1 BCNF, R_2-t tovább elemezzük a 2. lépéstől folytatva.

2. lépés

R_2 reláció függőségei:

- DIAKKOD → DIAKNEV
- DIAKKOD → OFONOK
- (DIAKKOD, NYELV, FELEVKOD) → OSZTALYZAT

Nem BCNF a DIAKKOD → DIAKNEV és a DIAKKOD → OFONOK függőségek miatt.

3. lépés:

Ezt alakítsuk át DIAKKOD → DIAKNEV, OFONOK alakú függéségre, és ezt vegyük ki R_2 -ből, ekkor

R_3(<u>DIAKKOD</u>, DIAKNEV, OFONOK)

R_4(<u>DIAKKOD</u>, <u>NYELV</u>, <u>FELEVKOD</u>, OSZTALYZAT)

részsémákat kapjuk.

Itt már R_3 és R_4 is BCNF alak, tehát készen vagyunk, a megoldás az R_1, R_3 és az R_4 relációk együttese lesz.

1.6. Az SQL nyelv

1.6.1. Bevezetés

1974-75-ben kezdték az SQL nyelv kifejlesztését az IBM-nél, az "eredeti" neve SEQUEL (Structured English QUEry Language).
1979-től több cég (pl. IBM,ORACLE Corp.) kereskedelmi forgalomban kapható termékeiben már szerepel. 1987-től ANSI szabvány lett.

1.6.1.1. Jelentősége

A nyelv jelentőségét, főbb jelemzőit az alábbiakban soroljuk fel:

- Szabvány, amelyet jelenleg csaknem minden relációs adatbáziskezelő alkalmaz (kisebb-nagyobb módosításokkal).
- Tömör, felhasználó közeli nyelv, alkalmas hálózatokon adatbáziskezelő szerver és kliensek közötti kommunikációra.
- Nem procedurális programozási nyelv.

1.6.1.2. A nyelv definíciója

A nyelv utasításait a következő csoportokra oszthatjuk:

- adatleíró (DDL Data Definition Language)
- adatmódosító (DML Data Manipulation Language)
- lekérdező (Queries)
- adatelérést vezérlő (DCL Data Control Language)

A nyelvben a szöveg literálok kivételével a kis- és nagybetűket nem különböztetjük meg. A megadott példáknál a könnyebb érthetőség miatt a nyelv alapszavait csupa nagy betűvel, míg a programozó által használt egyéb neveket kis betűkkel írjuk. A parancsok több sorba is átnyúlhatnak, a sorokra tördelésnek nincs szemantikai jelentősége. Az SQL parancsokat a pontosvessző zárja le.

1.6.1.3. A példákban szereplő táblák

A leírás az ORACLE adatbáziskezelő SQL dialektusát ismerteti, ez többé-kevésbé megfelel az egyéb termékekben található nyelv variációknak. A nyelv termék- illetve hardver specifikus elemeit nem, vagy csak futólag ismertetjük. Az utasítások ismertetésénél a következő táblákat használjuk.(Lásd még a 1.11.1. fejezetet.)

Az EMP tábla az alkalmazottak adatainak tárolására szolgál.

empno	**ename**	**job**	**mgr**	**hiredate**	**sal**	**comm**	**deptno**
7329	SMITH	CLERK	7902	17-Dec-80	800		20
7499	ALLEN	SALESMAN	7698	20-FE8-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FE8-81	1250	500	30
7566	JONES	MANAGER	7839	02-APA-81	2975		20
7654	MARTIN	SALESMAN	7698	28-5EP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-May-81	2850		30
7782	CLARK	MANAGER	7839	09-Jun-81	2450		10
7788	SCOTT	ANALYST	7566	09-Dec-82	3000		20
7839	KING	PRESIDENT		17-Nov-81	5000		10
7840	TURMER	SALESMAN	7698	08-Sep-81	1500		30
7876	ADAMS	CLERK	7788	12-Jun-83	1100		20
7900	JAMES	CLERK	7698	03-Dec-81	950		30
7902	FORD	ANALYST	7566	03-Dec-81	3000		20
7934	MILLER	CLERK	7782	23-Jan-82	1300		20

A DEPT tábla a cég részlegeinek adatait tartalmazza .

deptno	**dname**	**loc**
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

A további táblák az 1.11 Függelék részben találhatóak.

1.6.2. Objektumok létrehozása, törlése, módosítása

Az SQL nyelvben többek köyött objektumok a táblák, a néyetek és az indexek. Mivel a nézetek létrehozásához már kellenek a lekérdezéssel kapcsolatos ismeretek, ezért ezzel csak a lekérdezések ismertetése után foglalkozunk. A következőkben a táblák létrehozásával, majd az ojektumok törlésével, módosításával fogunk foglalkozni.

Az alkalmazott jelölések: a ⟨ ⟩ jelek között egy azonosító találahtó. A [ ] jeleken belüli rész opcionális, nem kötelező. A ...n az előtte lévő rész ismételhetőségét jelenti. A{ } jelek között egy felsorolás található, melyek között a | jel a felsoroltak kizáró vagy kapcsolatát jelenti.

1.6.2.1. Táblák létrehozása

Új táblákat a

```
CREATE TABLE ⟨táblanév⟩
(
⟨oszlopdefinicio1⟩ [, ...n]
[,táblaszintű megkötések]
);
```

paranccsal lehet létrehozni.

Az oszlopdefinició pedig:

```
⟨oszlopnév⟩ ⟨adattipus⟩ [ DEFAULT ÉRTÉK ] [mezőszintű megkötések]
```

A neveknek betűvel kell kezdődniük, tartalmazhatnak számokat, valamint a '_', '#', és '$' karaktert. A kis és nagy betűk azonosnak számítanak. Amennyiben a neveket idézőjelek között adjuk meg, abban az esetben az idézőjelen kívül minden betű használható, és ilyenkor a kis- és nagybetűk is különbözőnek számítanak. Nem használhatók névként a foglalt kulcsszavak. A tábla neveknek egyedieknek kell lenniük, hosszuk legfeljebb 30 karakter lehet(verzió függő).

A lehetséges adattípusok implementációkként változhatnak, általában a következő adattípusok megtalálhatók:

- CHAR (n) legfeljebb n byte vagy karakter hosszú szöveg, n hosszan tárol. n maximális értéke 32767 lehet;
- LONG (n) mint CHAR , de hosszára általában nincs (nagyon nagy) felső korlát;

- VARCHAR (n) legfeljebb n byte vagy karakter hosszú szöveg, csak a tényleges adatokat tárolja. n maximális értéke 32767 lehet;
- NUMBER (w) az előjellel együtt legfeljebb w karakter széles egész szám;
- NUMBER (w,d) w a teljes szám, d a törtrész szélessége;
- DATE dátum (és általában időpont).

A NUMBER típus legfeljebb 40 számjegy szélességű lehet. A DATE típusban i.e. 4712 január l. és i.sz. 9999 december 31. közötti dátumok tárolhatók. A CHAR tipus kivételével definiált szélességtől függetlenül a mező csak annyi tárolóhelyet foglal, amennyi az adat tárolásához szükséges.

Ha valamelyik oszlop definíciója a mező szintű megkötésekben tartalmazza a NOT NULL megkötést, a megfelelő mezőben mindig érték kell, hogy szerepeljen. Az egyéb megkötésekkel az 1.7.1 fejezetben foglalkozunk.

A felhasznált emp tábla definíciója a következő lehet:

```
CREATE TABLE emp
    (empno NUMBER (4) NOT NULL ,
    ename CHAR (10),
    job CHAR (9),
    mgr NUMBER (4),
    hiredate DATE ,
    sal NUMBER (7,2),
    comm NUMBER (7,2),
    deptno NUMBER (2) NOT NULL );
```

A bővebb megkötésekkel kiegészítve: (lásd még a 1.7.1 fejezetet)

```
CREATE TABLE emp
    (empno NUMBER (4) CONSTRAINT pk_emp PRIMARY KEY ,
    ename VARCHAR2 (10) CONSTRAINT nn_ename NOT NULL
    CONSTRAINT upper_ename CHECK (ename = UPPER (ename)),
    job VARCHAR2 (9),
    mgr NUMBER (4) CONSTRAINT fk_mgr REFERENCES scott.emp(empno),
    hiredate DATE DEFAULT SYSDATE ,
    sal NUMBER (10,2) CONSTRAINT ck_sal CHECK (sal > 500),
    comm NUMBER (9,0) DEFAULT NULL ,
    deptno NUMBER (2) CONSTRAINT nn_deptno NOT NULL
    CONSTRAINT fk_deptno REFERENCES dept(deptno));
```

A felhasznált dept tábla definíciója pedig a következő lehet:

```
CREATE TABLE dept
    (deptno NUMBER (2) NOT NULL ,
    dname CHAR (14),
    loc CHAR (13));
```

A dept tábla definíciója a megkötésekkel:

```
CREATE TABLE dept
    (deptno NUMBER (2) CONSTRAINT pk_dept PRIMARY KEY ,
    dname CHAR (14) CONSTRAINT unique_dname UNIQUE ,
    loc CHAR (13));
```

1.6.2.2. Objektumok törlése

A fenti adatbázis objektumokat a DROP paranccsal lehet törölni.

```
DROP [TABLE | VIEW | INDEX ] ⟨név⟩;
```

1.6.2.3. Tábla definíciók módosítása

Már létező táblákat módosítani az

```
ALTER TABLE ⟨táblanév⟩ [ ADD | MODIFY ] ⟨oszlopdefinició⟩ ⟨típus⟩;
```

paranccsal lehet, ahol ADD egy új, NULL vagy DEFAULT értékű oszlopot illeszt a táblához, míg MODIFY paranccsal egy létező oszlop szélességét növelhetjük.
Az ALERT TABLE parancs csak a táblák formátumának módosítására szolgál, nem változtatja meg a tábla adatait (egy oszlop szélességének növelése, illetve egy új oszlop hozzáadása a táblához).

```
ALTER TABLE proj MODIFY ( budget NUMBER (9,2));
ALTER TABLE emp ADD ( projno NUMBER ) ;
```

1.6.2.4. Új sorok bevitele

Új sorokat egy meglévő táblába az

```
INSERT INTO ⟨táblanév⟩ [(⟨oszlopnév⟩, ...)] VALUES (⟨kif1⟩, ...);
```

illetve az

```
INSERT INTO ⟨táblanév⟩ [(⟨oszlopnév⟩, ...)] SELECT ...;
```

utasítások valamelyikével lehetséges.

Míg az első szerkezet egyetlen sort, addig a második a lekérdezés által előállított összes sort beilleszti. (Figyelem: a táblákban az egyes rekordok sorrendje tetszőleges, így a beillesztés sem feltétlenül a tábla "végére" történik.)

Amennyiben nem adtuk meg az oszlopok nevét, akkor a tábla deklarálásánál megadott sorrendben minden mezőnek értéket kell adni, esetleg NULL -t, ha viszont megadtuk az egyes oszlopok neveit, akkor csak azoknak adunk értéket, mégpedig a felsorolásuk sorrendjében, a többi mező NULL vagy DEFAULT értékű lesz.

Nem fut le az INSERT, ha a NOT NULL megkötésű mezőnek nem adunk értéket. Az adatbáziskezelő ellenőrzi, hogy az egyes mezőkbe ne kerülhessen a tábla definíciójával ellentétben NULL érték, és ellenőrzi az integritási feltételeket is.
Példa az INSERT parancsra:

INSERT INTO emp VALUES (7954, ´CARTER´, 'CLERK', 7698. '7-APR-84', 1000, NULL , 30);

A fenti esetben az értékeket olyan sorrendben kell megadni, amilyen sorrendben a mezők a tábla létrehozásakor definiálva voltak. Ha ettől el akarunk térni, akkor a mezők neveit is meg kell adni.

INSERT INTO emp (empno, ename, hiredate, deptno, sal) VALUES (7955, 'WlLSON', TO DATE ('1993-FEB-19 9:30', 'YYYY-MON-DD HH:Ml'), 30,1500);

A táblába beszúrandó sorok a SELECT paranccsal is előállíthatók. Így több sor is beszúrható egyszerre.

INSERT INTO bonus (ename, job, sal, comm)
SELECT ename, job, sal, comm FROM emp
WHERE job ='MANAGER' OR comm > 0.25 * SAL;

A tábla létrehozása és a sorok feltöltése akár egyetlen paranccsal is megoldható.

CREATE TABLE bonus(ename, job, sal, comm) AS
SELECT ename, job, sal, comm FROM emp
WHERE job = 'MANAGER' OR comm > 0.25 * SAL;

1.6.2.5. Sorok törlése

Sorokat törölni a

DELETE [FROM] ⟨táblanév⟩
[WHERE ⟨logikai kifejezés⟩];

paranccsal lehet. Ha a WHERE hiányzik, a tábla valamennyi sorát, egyébként a logikai kifejezés által kiválasztott sorokat töröljük (ahol a kifejezés igaz).
Példa a DELETE parancsra:

```
DELETE FROM bonus
WHERE job IN
(SELECT job FROM emp WHERE ename = 'JONES' );
```

1.6.2.6. Adatok módosítása

Sorokban mező értékeit módosítani, a mezőknek új értéket adni az

```
UPDATE ⟨táblanév⟩ SET ⟨oszlopnév⟩ = ⟨kifejezés⟩, ...
[WHERE ⟨logikai kifejezés⟩];
```

paranccsal lehet.

Ha a WHERE hiányzik, a parancs a tábla valamennyi sorában módosít, egyébként csak a kiválasztott sorokban.
Például adjunk az üzletkötőknek 20% fizetésemelést:

```
UPDATE emp SET sal = 1.2 * sal WHERE job = 'SALESMAN';
```

A SELECT parancs eredménye itt is használható, az kap fizetés emelést, akinek a neve a bonus táblában szerepel.

```
UPDATE emp SET sal = 1.5 * sal WHERE ename IN
(SELECT ename FROM bonus);
```

A parancsban a kifejezésben a szokásos operátorok és függvényeken felül egyes implementációk akár lekérdezést is megengednek.

A fentiek hatására az adatbázis módosulását még mások nem látják és a módosítás még nem végleges. A változtatásokat a COMMIT paranccsal kell véglegesíteni. A ROLLBACK parancs segítségével azonban még visszaállítható a legutolsó COMMIT parancs utáni állapot. Ezek részletes ismertetésére a tranzakcióknál kerül sor.

1.6.3. Lekérdezések

A lekérdezések általános szintaxisa a következő:

```
SELECT ⟨jellemzők⟩
FROM ⟨táblák⟩
[WHERE ⟨logikai kifejezés⟩]
[GROUP BY ⟨csoportosítási jellemzők⟩]
[HAVING ⟨csoportra vonatkozó feltétel⟩]
[ORDER BY ⟨rendezési jellemzők⟩];
```

A lekérdezés művelete eredményül egy újabb táblát állít elő (nem tárolt tábla, csak tábla struktúrájú adathalmaz), persze lehet, hogy az eredmény táblának csak egy oszlopa lesz. Az oszlopok száma a SELECT után felsorolt jellemzőktől függ, míg az sorok száma a WHERE után megadott logikai kifejezéstől függ, és akár 0 is lehet, ha ez a logikai kifejezés sohasem igaz.

Az eredmény tábla a lekérdezés után rendelkezésre áll, listázható vagy egyéb módon felhasználható (pl. az adatmódosító utasításokban).

A ⟨jellemzők⟩ definiálják az eredmény tábla oszlopait, a ⟨táblák⟩ adják meg a lekérdezésben résztvevő táblák nevét, a ⟨logikai kifejezés⟩ segítségével "válogathatunk" az eredmény sorai között. A ⟨csoportosítási jellemzők⟩ az eredmény tábla sorait rendezik egymás mellé, illetve a ⟨rendezési jellemzők⟩ a megjelenő sorok sorrendjét határozzák meg.

Nézzük meg, hogy a lekérdezés műveletével hogyan lehet megvalósítani a relációs algebra alapműveleteit.

1.6.3.1. Vetítés (projection)

A vetítés művelete egy táblából adott oszlopokat válogat ki. A ⟨jellemzők⟩ között kell felsorolni a kívánt oszlopokat a megjelenésük sorrendjében. Például az alkalmazottak neve és fizetése:

SELECT ename, sal FROM emp ;

Az oszlopneveket listaszerűen kell megadni. Az oszlopok a megadás sorrendjében jelennek meg. Ha valamennyi oszlopot meg akarjuk jeleníteni, akkor a nevek helyébe *-ot kell ími.

SELECT * FROM emp ;

A ⟨jellemzők⟩ közé nem csak a FROM mögött megadott tábla oszlopainak nevét lehet megadni, hanem használhatjuk az SQL beépített műveleteit, függvényeit, pl. egyszerű aritmetikai kifejezéseket új érték előállítására.

A dolgozók egész éves fizetése:

SELECT ename, 12 * sal FROM emp;

Amennyiben a dolgozó által kézhez kapott teljes összeget fizetést és prémiumot együtt akarjuk megkapni, hasonlóan járhatunk el. Az jelent csak problémát, hogy a comm érték nincs mindenhol kitöltve, az üres mezőket nem tudjuk hozzáadni a fizetéshez (ami üres, az nem 0)! Ilyenkor használhatjuk az NVL (oszlop, érték) függvényt*(oracle) vagy az ISNULL (oszlop, érték) függvényt**(SQL 2000), amely az üres (NULL) mező helyett a megadott értéket adja vissza (részletesen lásd a 1.6.3.4 fejezetben).

A dolgozó által felvett pénz:

SELECT ename, 12*sal +NVL (comm, 0) FROM emp;

A kiválasztott oszlopokat tartalmazó eredmény táblákban lehetnek azonos sorok, ami ellentmond a relációs táblák egyik alapvető követelményének. Ennek ellenére a SELECT utasítás nem szűri ki automatikusan az azonos sorokat, mert ez túlságosan időigényes művelet. A programozónak kell tudnia, hogy az előállított táblában lehetnek-e (zavarnak-e) ilyen sorok. Ha kell, a ezeket a DISTINCT kulcsszóval szűrhetjük ki.

Az összes különböző munka megnevezése:

SELECT DISTINCT job FROM emp;

JOB
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN

Az oszlopokhoz hivatkozási név (alias) is megadható. Az oszlopok kiírásakor a fejrészben a hivatkozási név jelenik meg. Ha aposztrofok között adjuk meg a nevet, akkor a kis és nagybetűk különböznek, egyébként a fejjrészben csupa nagybetűvel

írja, mint az oszlopneveket egyébként is.

SELECT dname department, deptno FROM dept;

DEPARTMENT	DEPTNO
ACCOUNTING	10
RESEARCH	20
SALES	30
OPERATIONS	40

1.6.3.2. Kizárás (restriction)

A kizárás műveleténél a tábla sorai közül válogatunk. A WHERE után álló ⟨logikai kifejezés⟩ igaz értékeinek megfelelő sorok kerülnek be az eredmény táblába. Pl. azok a dolgozók, akik fizetése 2000 dollárnál több:

SELECT ename, sal FROM emp

WHERE sal > 2000;

ENAME	SAL
JONES	2975
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
FORD	3000

Vizsgáljuk meg a logikai kifejezések szerkezetét.

A kifejezések elemi összetevői:

- literálok különböző típusú értékekre: számok, szöveg, dátum;
- oszlopok nevei;
- a fenti elemekből elemi adatműveletekkel képzett kifejezések
  - számoknál: aritmetikai műveletek, aritmetikai függvények;
  - szövegeknél: SUBSTR() , INSTR() , UPPER() , LOWER() , SOUNDEX() , ...;
  - dátumoknál: +, -, konverziók;
- halmazok: pl.: (10,20,30);
- zárójelek között egy teljes SELECT utasítás (ld. később).

A fenti műveletekkel képzett adatokból logikai értéket a következő műveletekkel állíthatunk elő:

- relációk: <, <=, =, !=, >=, >;
- intervallumba tartozás: BETWEEN .. AND ..; A kezdő és a végső érték is benne van az intervallumban.
- NULL érték vizsgálat: IS NULL , IS NOT NULL ;
- halmaz eleme: IN <halmaz>;
- szöveg vizsgálat: mintával összevetés ..LIKE <minta>, ahol a mintán belül két megkülönböztetett karakter van, a % tetszőleges, akár 0 hosszúságú karaktersorozat, az _ a tetszőleges karakter jelzésére; Ha a %-ra, mint karakterre szeretnénk keresni, akkor egy ugynevezett ESCAPE karaktert kell választanunk, pl: a \-t, majd a LIKE '%\%_' ESCAPE '\' az összes olyan szöveget megtalálja, aminek az utolsó előtti karaktere %.

Pl. a LIKE -ra:

SELECT ename, job FROM emp

WHERE ename LIKE 'M%',

azokat a dolgozókat válogatja ki, akiknek a neve M betűvel kezdődik.

ENAME	JOB
MARTIN	SALESMAN
MILLER	CLERK

Végül a logikai értékeket a zárójelezéssel illetve az AND , OR és NOT műveletekkel lehet tovább kombinálni.

Példák:

1. A 82 .. 83-es években felvett dolgozók:

   SELECT ename, hiredate FROM emp

   WHERE hiredate BETWEEN '01-JAN-82' AND '31-DEC-89';

ENAME	HIREDATE
SCOTT	09-DEC-82
ADAMS	12-JUN-83
MILLER	23-JAN-82

2. Azon dolgozók neve és fizetése, akik havonta 2000 dollárnál kevesebbet keresnek és nem kaptak prémiumot:
   SELECT ename, sal FROM emp
   WHERE sal < 2000 AND comm IS NULL ;

ENAME	SAL
SMITH	800
TURMER	1500
ADAMS	1100
JAMES	950
MILLER	1300

3. Azoknak a dolgozóknak az összes adata, akik a 30-as osztályon dolgoznak, vagy a munkakörük 'clerk' vagy 'salesman', és a fizetésük nem 1000 és 2000 dollár között van:
   SELECT * FROM emp
   WHERE deptno = 30 OR job IN ('CLERK', 'SALESMAN')
   AND sal NOT BETWEEN 2800 AND 3000 ;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CLERK	7902	17-DEC-80	800		20
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7900	JAMES	CLERK	7698	03-DEC-81	950		30

A feltételekben használható operátorok precedencia sorrendje :

- 1. =, !=, ^=, <>, <, <=, >, >=, BETWEEN ... AND ..., IN (list), LIKE, IS NULL, NOT BETWEEN, NOT IN, NOT LIKE, IS NOT NULL
- 2.NOT
- 3.AND
- 4.OR

1.6.3.3. Összekapcsolás (join)

Az természetes összekapcsolás műveleténél 2 vagy több tábla soraiból hozunk létre egy-egy új sort akkor, ha a két tábla megfelelő sorának azonos nevű mezőinek értéke megegyező.
Az SQL nyelvben erre két megoldás van.

Az egyik lehetőség, hogy a SELECT kifejezésben a FROM utáni ⟨táblák⟩-ban kell megadni az érintett táblák neveit, a WHERE mögötti logikai kifejezés definiálja azokat az oszlopokat, amely értékei szerint történik meg az összekapcsolás.
Pl. az egyes osztályok neve, székhelye és a hozzájuk tartozó dolgozók:

SELECT dept.dname, dept.loc, emp.ename
FROM dept, emp
WHERE dept.deptno = emp.deptno;

DEPT.DNAME	DEPT.LOC	EMP.ENAME
ACCOUNTING	MEW YORK	CLARK
ACCOUNTING	MEW YORK	KING
ACCOUNTING	MEW YORK	MILLER
RESEARCH	DALLAS	SMITH
RESEARCH	DALLAS	JONES
RESEARCH	DALLAS	SCOTT
RESEARCH	DALLAS	ADAMS
RESEARCH	DALLAS	FORD
SALES	CHICAGO	ALLEN
SALES	CHICAGO	WARD
SALES	CHICAGO	MARTIN
SALES	CHICAGO	BLAKE
SALES	CHICAGO	TURMER
SALES	CHICAGO	JAMES

Látható, hogy mivel mindkét felhasznált táblában azonos az összekapcsolást megvalósító oszlop neve, a WHERE -t követő logikai kifejezésben az oszlop neve mellé meg kell adni a tábla nevét is.
Hasonló helyzet előfordulhat a SELECT -et követő ⟨jellemzők⟩ között is.

A másik lehetőség az összekapcsolásra, hogy a FROM után a <táblanév1> JOIN <táblanév2> ON <tábla1.mező1> = <tabla2.mező2> alakban adjuk meg az összekapcsolás feltételét.
Az előbbi **példa:**

SELECT dept.dname, dept.loc, emp.ename
FROM dept JOIN emp ON dept.deptno = emp.deptno;

Ha megvizsgáljuk a fenti példában kapott eredmény, láthatjuk, hogy a 40-es osztály nem szerepel a listában. Ennek az az oka, hogy enenk az osztálynak

nincs egyetlen dolgozója sem, tehát az egyesítésnél nem találtunk az emp táblában egyetlen olyan sort sem, amelyet ehhez az osztályhoz kapcsolhattunk volna. Ez lehet kívánatos eredmény, azonban az SQL-ben lehetőség van arra is, hogy ezeket a sorokat is egyesítsük, ilyenkor az egyesítésben az emp táblához hozzáképzelünk egy üres sort is, amely azzal a sorral egyesíthető, ami más sorokkal nem volt összevonható. Ezt külső egyesítésnek hívjuk (*outer join*). A módosított példa a következőképpen néz ki:

```
SELECT dept.dname, dept.loc, emp.ename
FROM dept, emp
WHERE dept.deptno = emp.deptno (+);
```

Illetve

```
SELECT dept.dname, dept.loc, emp.ename
FROM dept LEFT OUTER JOIN emp
ON dept.deptno = emp.deptno;
```

vagy

```
SELECT dept.dname, dept.loc, emp.ename
FROM emp RIGHT OUTER JOIN dept
ON dept.deptno = emp.deptno;
```

A (+) jel jelzi azt a táblát, amelyikhez az egyesítés előtt az üres mezőket tartozó sort hozzá kell venni. (Jelen esetben ez fog a dept tábla 40-es deptno-jú sorával összekapcsolódni.) A (+) hatására a rendszer úgy kezeli az EMP táblát, mintha lenne egy csupa NULL értékkel rendelkező sora. Ezt a sort kapcsolja hozzá a másik tábla minden olyan sorához, amelyhez nem sikerült a feltételnek megfelelő sort találni. Ez a módszer használható arra is, hogy kiválasszuk azokat a sorokat, melyeket nem lehet a másik táblához kapcsolni.

```
SELECT DISTINCT dept.deptno, dname, loc
FROM dept, emp
WHERE dept.deptno = emp.deptno (+) AND empno IS NULL ;
```

A LEFT OUTER JOIN azt jelenti, hogy a bal oldalon megadott tábla összes sorát szeretnénk látni, akkor is, ha nincs hozzá kapcsolódó sor a másik táblában. (RIGHT OUTER JOIN -nál a jobb oldalra vonatkoznak a fentiek.)

Az összekapcsoláson kívül további feltételek is megadhatók a sorok kiválasztására.

Pl. az egyes osztályok neve, székhelye és a hozzájuk tartozó dolgozók, ha a dolgozó fizetése nagyobb 2000 dollárnál:

```
SELECT dept.dname, dept.loc, emp.ename
FROM dept, emp
WHERE dept.deptno = emp.deptno AND emp.sal>2000;
```

Illetve:

```
SELECT dept.dname, dept.loc, emp.ename
FROM dept
JOIN emp ON dept.deptno = emp.deptno
WHERE emp.sal>2000;
```

DEPT.DNAME	DEPT.LOC	EMP.ENAME
ACCOUNTING	MEW YORK	CLARK
ACCOUNTING	MEW YORK	KING
RESEARCH	DALLAS	JONES
RESEARCH	DALLAS	SCOTT
RESEARCH	DALLAS	FORD
SALES	CHICAGO	BLAKE

Az egyesítésnél lehet egy táblát önmagával is összekapcsolni. Pl. azon dolgozók, akik többet keresnek a főnöküknél:

```
SELECT e.ename, e.sal, m.ename, m.sal
FROM emp e, emp m
WHERE e.mgr = m.empno AND e.sal > m.sal;
```

E.ENAME	E.SAL	M.ENAME	M.SAL
SCOTT	3000	JONES	2975
FORD	3000	JONES	2975

A fenti példában ugyanazt a táblát kétszer is használjuk, a két tábla oszlopainak megkülönböztetésére a táblákat a FROM részben lokális névvel (alias) is el kell látni. Lokális neveket természetesen különböző táblák esetén is használhatunk. Látható, hogy az egyesítés mellett egyidejűleg más logikai kifejezéseket is használhatunk.

A fizetések fenti vizsgálatát felfoghatjuk úgy is, mint az egyesítés műveletének általánosított esetét, ahol nem csak az egyenlőség művelete használható, valamint úgy is, hogy a már egyesített táblából zárjuk ki a fizetésekre szabott követelményeknek meg nem felelő sorokat.

A kapcsolat a táblák között nem-egyenlőség típusú is lehet. Ekkor az egyik tábla sora a másik tábla több sorához is hozzákapcsolódhat.
Az alábbi példában azt keressük, ki keres többet 'JONES'-nál.

SELECT X.ename, X.sal. X.job, Y.ename, Y.sal, Y.job
FROM emp X, emp Y
WHERE X.sal > Y.sal AND Y.ename = 'JONES';

X.ENAME	X.SAL	X.JOB	Y.ENAME	Y.SAL	Y.JOB
SCOTT	3000	ANALYST	JONES	2975	MANAGER
KING	5000	PRESIDENT	JONES	2975	MANAGER
FORD	3000	ANALYST	JONES	2975	MANAGER

1.6.3.4. Oszlopfüggvények

A lekérdezés eredményeként előálló táblák egyes oszlopaiban lévő értékeken végrehajthatunk olyan műveleteket, amelyeket a szokásos programozási nyelveken ciklussal oldahtók meg, amelyek egyetlen értéket állítanak elő. Ezek paraméterként az eredménytábla összes sorának megadott oszlopát megkapják és ezen végzik el e kijelölt műveletet. Ilyen oszlopfüggvények a következők, amelyeket most a fizetés(sal) illetve a prémium(comm) oszlopra alkalmazunk:

- AVG (sal) az oszlopbeli értékek átlaga
- COUNT (comm) az oszlopban értékkel rendelkező sorok darabszáma
- MAX (sal) maximális érték az oszlopban
- MIN (sal) minimális érték az oszlopban
- SUM (sal) az oszlop elemeinek az összege

A count-nál a paraméter lehet *, ilyenkor az eredménytábla sorait számolja meg.

Az üzletkötők átlagfizetése:

SELECT AVG (sal)
FROM emp
WHERE job = 'SALESMAN';

AVG(SAL)
1400

Hány dolgozó van a táblában:

SELECT COUNT (*)

FROM emp;

COUNT(*)
14

Hány különböző beosztás van:

SELECT COUNT (DISTINCT job) 'job'

FROM emp;

job
5

Itt az eredmény tábla a DISTINCT job-nak megfelelően minden job-ot csak egyszer tartalmaz, azaz annyi sora lesz, ahány különböző job van, a COUNT ezeket számolja össze. Az oszlopfüggvényekről részletesen szó lesz még a 1.9.1.2

NULL értékek kezelése:

Az oszlopfüggvények nem veszik figyelembe a NULL értékeket

COUNT (*) megszámolja az összes sort, hiszen a sor valamelyik értéke biztosan nem NULL .

Átlagos prémium:

SELECT AVG (comm)

FROM emp;

AVG(COMM)
733.33

Illetve:

SELECT AVG (NVL (comm, 0)) 'prmium átlag'

FROM emp;

prmium átlag
157.14

Az utolsó példa az NVL függvényt használva az összes dolgozóra átlagolja a kifizetett prémiumot, mert itt a NULL értékű mezőt 0-val helyettesítjük és a számításnál figyelembe vesszük. Az előző példában, ahol NVL nélkül adtuk meg a lekérdezést, csak azokra átlagolja a prémiumot, akik egyáltalán kaptak prémiumot, mert a számításánál a NULL értékű rekordok kimaradnak.

Ha az oszlopfüggvények 0 sorra hajtódnak verge, (pl. a tábla üres, vagy a WHERE feltétel soha nem teljesül, vagy a teljesült sorokra a függvény argumentuma NULL értékű) akkor NULL értékkel térnek vissza, kivéve a COUNT (*)-ot, mert az ilyenkor 0-t ad vissza. Ezért ügyelni kell arra, hogy ha az oszlopfügvénnyel további műveleteket akarunk végezni, ez problémát okozhat.

Pl. szeretnénk kiíratni osztályonként a legnagyobb és a legkisebb fizetések arányát, akkor a 40-es osztály esetén NULL /NULL (nincs dolgozója az osztálynak) jönne ki, de a NULL értékkel aritmetika nem végezhető.(A megoldást lásd lejjebb).

Mivel az oszlopfüggvény eredménye egyetlen értéket állít elő, az oszlopfüggvény mellé vagy más oszlopfüggvényeket írhatunk, vagy olyan értéket írhatunk, amelyik az összes kiválasztott sorban azonos. Például írhatjuk:

SELECT COUNT (*), AVG (sal)

FROM emp;

COUNT(*)	AVG(SAL)
14	2073

de hibás a

SELECT COUNT (*), ename

FROM emp;.

Szintén nem működik a

SELECT job, AVG (sal)

FROM emp

WHERE job = 'SALESMAN';,

mert a WHERE kiértékelése előtt a lekérdezés feldolgozó nem tudja, hogy csak egy érték lesz soronként, ezért nem fogadja el a lekérdezést (szintaktikailag helytelen a lekérdezés). Ilyenkor a csoportosítás használatával oldhatjuk meg a feladatot, azaz:

SELECT job, AVG (sal)

FROM emp
WHERE job = 'SALESMAN'
GROUP BY job;

JOB	AVG(SAL)
SALESMAN	1400

(A GROUP BY -t később, a 1.6.3.5 fejezetben részletesen tárgyaljuk).
Egy érdekes példa az NVL és a SUM használatára:

SELECT deptno, NVL (SUM (sal),0)/NVL (SUM (comm), **1**) 'A fizetés és prémium aránya'
FROM emp
GROUP BY deptno
HAVING SUM (comm) IS NOT NULL ;

DEPTNO	A fizetés és prémium aránya
30	4.2727

Itt a SUM (comm) értékét az NVL függvényben 1-el helyettesítettük, mert a hányados kiszámítása előbb történik, mint a HAVING kiértékelése, és így elkerüljük a 0-val való osztást.

1.6.3.5. Csoportosítás

Az oszlopfüggvények a teljes kiválasztott táblára minden sorra lefutnak. Gyakran célszerű lenne a kiválasztott sorokat valamilyen szempont szerint csoportosítani és az oszlopfüggvényeket az egész tábla helyett ezekre a csoportokra alkalmazni. Foglalkozásonkénti átlagfizetés:

SELECT job, AVG (sal)
FROM emp
GROUP BY job;

JOB	AVG(SAL)
ANALYST	3000.00
CLERK	1037.50
MANAGER	2758.33
PRESIDENT	5000.00
SALESMAN	1400.00

A fenti parancs az emp tábla sorait a job oszlop azonos értékei alapján csoportosítja. Természetesen az oszlopfügvények korábbi használatához hasonlóan a SELECT ⟨jellemző⟩-k között csak a csoportosítás alapját képező oszlop neve illetve a csoportokra alkalmazott oszlopfüggvények szerepelhetnek. Ezt ugy is megfogalmazhatjuk, hogy a csoportosításnál minden olyan oszlopnevet fel kell sorolni, amit a lekérdezésnél használunk, akár a SELECT jellemzői között, akár a rendezés jellemzői között.

A csoportosítás után az eredményből bizonyos csoportok kihagyhatók. Foglalkozásonkénti átlagfizetés a 1000 és 3000 dollár közötti tartományban:

```
SELECT job, AVG (emp)
FROM emp
GROUP BY job
HAVING AVG (sal) BETWEEN 2000 AND 3000;
```

JOB	AVG(SAL)
ANALYST	3000.00
MANAGER	2758.33

A HAVING mögötti logikai kifejezésben természetesen csak egy-egy csoport közös jellemzőire vonatkozó értékek a csoportosítás alapját képező oszlop értéke, vagy oszlopfüggvények eredménye szerepelhet. Természetesen a csoportosítás előtt azért a WHERE feltételek használhatók. Célszerű gyorsabb WHERE feltételeket használni mindenhol, ahol csak lehet, a HAVING szerkezetet csak akkor alkalmazni, amikor a teljes csoporttól függő értékeket akarjuk vizsgálni. Ay adatbáziskezelő a lekérdezésnél először az alapadatokból a (where) feltételeinek megfelelően kiválogatja a szükséges adatokat, azután ezt a csoportosításnak megfelelően csoportokba rendezi, kiszámítja az oszlopfüggvényeket, végül ezen értékek alapján kiértékeli a HAVING -ban megadott feltételeket.

Több szempont szerinti csoportosításnál a
GROUP BY GROUPING SET ⟨jellemzők⟩
szerkezetet használhatjuk. Itt a GROUPING SET mindegyik csoportjára kükönkülön kiértékeli a csoport függvényeket. pl.:

SELECT deptno, job, SUM (sal)
FROM emp
GROUP BY GROUPING SETS ((deptno, job), deptno, job, ( ))

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	CLERK	1900
20	ANALYST	6000
20	MANAGER	2975
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600
10		8750
20		10875
30		9400
	ANALYST	6000
	CLERK	4150
	MANAGER	8275
	PRESIDENT	5000
	SALESMAN	5600
		29025

Itt a mintapéldában a

GROUP BY GROUPING SETS ((deptno, job), deptno, job, ( ))

esetén csoportok voltak a (depptno, job) együttes, a deptno, a job, valamint a (). Ez utóbbi az egész táblára számítja ki az oszlopfüggvény értékét.

A hasonló eredményt kaphatunk a ROLLUP használatával is.

SELECT deptno, job, count(*), sum(sal)
FROM emp
GROUP BY ROLLUP (deptno,job);

DEPTNO	JOB	COUNT(*)	SUM(SAL)
10	CLERK	1	1300
10	MANAGER	1	2450

DEPTNO	JOB	COUNT(*)	SUM(SAL)
10	PRESIDENT	1	5000
10		4	8750
20	ANALYST	2	6000
20	CLERK	2	1900
20	MANAGER	1	2975
20		5	10875
30	CLERK	1	950
30	MANAGER	1	2850
30	SALESMAN	4	5600
30		6	9400
15		14	29025

A ROLLUP (a, b, c) megfelel a GROUPING SETS ((a, b, c), (a, b), (a), ())-nek. jelen esetben a

GROUP BY ROLLUP (deptno, job) megfelel a

GROUP BY GROUPING SETS ((deptno, job), deptno, ()) -nek, csak a eredmény sorrendje különbözik.

Még finomíthajuk a csoportosítást a

GROUP BY CUBE (jellemzők)

használatával. Ebben az esetben a jellemzők sorrendjében csoportokra és alcsoportokra számolja ki az oszlopfüggvényeket. pl:.

```
SELECT deptno, job, count(*), sum(sal)
FROM emp
GROUP BY CUBE (deptno, job);
```

Az eredmény:

DEPTNO	JOB	COUNT(*)	SUM(SAL)
10	CLERK	1	1300
10	MANAGER	1	2450
10	PRESIDENT	1	5000
10		3	8750
20	ANALYST	2	6000
20	CLERK	2	1900
20	MANAGER	1	2975
20		5	10875
30	CLERK	1	950

DEPTNO	JOB	COUNT(*)	SUM(SAL)
30	MANAGER	1	2850
30	SALESMAN	4	5600
30		6	9400
	ANALYST	2	6000
	CLERK	4	4150
	MANAGER	3	8275
	PRESIDENT	1	5000
	SALESMAN	4	5600
		14	29025

A GROUP BY CUBE (a, b, c) megfelel a GROUP BY GROUPING SETS ((a, b, c), (a, b), (b, c), (a, c), (a), (b), (c), ())-nek.

jelen példánál a

GROUP BY CUBE (deptno, job) megfelel a

GROUP BY GROUPING SETS ((deptno, job), deptno, job, ()) -nek, csak az eredmény sorrendje különbözik.

1.6.3.6. Rendezés

Az eddig tárgyalt lekérdezések eredményében a sorok "véletlenszerű" a programozó által nem megadható sorrendben szerepeltek (amilyen sorrendben az adatokhoz az adatbáziskezelő hozzáfér, ez általában a tárolás sorrendje, ami induláskor a bevitel sorrendje, majd a törlések és ujabb bevitelek miatt teljesen követhetetlen). A sorrendet lehet az ORDER BY által megadott rendezéssel szabályozni. A rendezés több oszlop értékei szerint is történhet, ilyenkor az először megadott oszlop szerint rendezünk, majd az itt álló azonos értékek esetében használjuk a következőnek megadott oszlop(ok) értékét. Minden egyes oszlop esetében külön meg lehet adni a rendezés "irányát", amely alap esetben emelkedő ASC , de a DESC módosítóval csökkenő rendezés írható elő. Az alapértelmezés szerinti irány a növekvő ASC .

SELECT ename, job, sal

FROM emp

ORDER BY job, sal DESC ;

ENAME	JOB	SAL
SCOTT	ANALYST	3000
FORD	ANALYST	3000
MILLER	CLERK	1300
ADAMS	CLERK	1100
JAMES	CLERK	950
SMITH	CLERK	800
BLAKE	MANAGER	2850
CLARK	MANAGER	2450
JONES	MANAGER	2975
KING	PRESIDENT	5000
ALLEN	SALESMAN	1600
TURMER	SALESMAN	1500
MARTIN	SALESMAN	1250
WARD	SALESMAN	1250

Minden esetben a lista elejére kerülnek azok a sorok, amelyekben a rendező oszlopbeli mező értéke NULL . A 30-as osztály dolgozói (rendezve a job szerint növekvő és a sal szerint csökkenő sorrendbe:

```
SELECT *
FROM emp
WHERE deptno = 30
ORDER BY job, sal DESC ;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7499	ALLEN	SALESMAN	7698	20-FE8-81	1600	300	30
784	TURMER	SALESMAN	7698	O8-SEP-81	1500		30
7654	MARTIN	SALESMAN	7698	28-5EP-81	1250	1400	30
7521	WARD	SALESMAN	7698	22-FE8-81	1250	500	30

Osztályok szerinti átlagfizetés növekvő sorrendben:

```
SELECT deptno, AVG (sal)
FROM emp
GROUP BY deptno
ORDER BY AVG (sal);
```

DEPTNO	AVG(SAL)
30	1566,67
20	2175,00
10	2916,67

Az ORDER BY esetében az oszlopokra az alias névvel vagy a sorszámukkal is lehet hivatkozni.

1.6.3.7. Egymásba ágyazott lekérdezések

A FROM után állhat egy SELECT utasítás, ami egy táblát ad vissza, és ezt alias névvel ellátva ugyanúgy használhatjuk a lekérdezés további részeiben mint egy igazi táblát.

A WHERE mögött álló logikai kifejezésben is állhat egy teljes SELECT utasítás is. Itt azonban csak egy oszlopa lehet a lekérdezésnek, és ezt mint halmazt kezelhetjük. Ha csak egy sorral tér vissza a lekérdezés, akkor ezt egyszerű értékként is használhatjuk. Az allekérdezést mindig zárójelek között kell megadni.
Például a következő lekérdezésben egy halmazt kapunk vissza. A nem New Yorkban dolgozók listája:

```
SELECT ename
FROM emp
WHERE deptno IN
      (SELECT deptno FROM dept
      WHERE loc != 'NEW YORK');
```

ENAME
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKEv
SCOTT
TURMER
ADAMS
JAMES
FORD

Azaz először kiválasztjuk azon osztályok azonosítóját, amelyek nem New Yorkban vannak, majd azt vizsgáljuk, hogy az ezekből képzett halmazban található-e az adott dolgozó osztályának azonosítója.
Ugyanezt a listát megkaphatjuk az egyesítés műveletével is:

```
SELECT ename
FROM dept, emp
WHERE dept.deptno = emp.deptno AND dept.loc !='NEW YORK';
```

Mint említettük, a beágyazott lekérdezés vagy egyetlen értéket ad vissza, azért mert egyetlen megfelelő sor egyetlen oszlopát választottuk ki illetve oszlopfüggvényt használtunk, vagy több értéket, több sort állít elő. Az előbbi esetben a SELECT értékét az elemi értékekkel azonos módon használhatjuk. Több érték egy halmazt jelent, tehát a halmazműveleteket használhatjuk. A korábban megismert IN () eleme művelet mellett használható az ANY () illetve az ALL () műveletek, ahol a kívánt reláció a halmaz legalább egy, illetve valamennyi értékére igaz. A legmagasabb fizetésű dolgozók (lehet, hogy több van!):

```
SELECT ename, sal
FROM emp
WHERE sal >= ALL
      (SELECT sal FROM emp);
```

ENAME	SAL
KING	5000

Ugyanez a példa oszlopfüggvény felhasználásával (az allekérdezés csak egy értéket ad vissza):

```
SELECT ename, sal
FROM emp
WHERE sal =
      (SELECT MAX (sal)
      FROM emp);
```

Az = ANY helyett az IN , a != ALL helyett a NOT IN használható.

ORACLE esetén az alkérdés több oszlopot is visszaadhat eredményként. A feltételeket ennek megfelelően kell megfogalmazni.

```
SELECT ename, job, sal
```

```
FROM emp
WHERE (job, sal ) =
        (SELECT job, sal
        FROM emp
        WHERE ename = 'FORD' );
```

ENAME	JOB	SAL
SCOTT	ANALYST	3000
FORD	ANALYST	3000

Vagy akár több visszatérési sor esetén

```
SELECT ename, job, sal
FROM emp
WHERE (job, sal ) IN
        (SELECT job, sal
        FROM emp
        WHERE sal>3000 );
```

ENAME	JOB	SAL
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
FORD	ANALYST	3000

Az alkérdések egymásba ágyazhatók, logikai (AND , OR) és halmazműveletekkel (UNION, INTERSECT, MINUS) összekombinálhatók (lásd a 1.6.3.9 fejezetet). Ha a főkérdésben és az alkérdésben ugyanaz a tábla szerepel, és a főkérdésbeli táblára akarunk hivatkozni, akkor a főkérdésben hivatkozási nevet kell a táblához rendelni.

Ha csak arra vagyunk kíváncsiak, hogy az allekérdezésben létezik-e legalább egy sor, akkor az EXISTS függvényt használhatjuk.

```
SELECT job, ename, empno, deptno
FROM emp X
WHERE EXISTS
        (SELECT * FROM emp
        WHERE X.empno = mgr )
        ORDER BY empno;
```

1.6.3.8. Fa struktúrába rendezett táblák

A tábla sorait precedencia szabály megadásával fa struktúrába lehet rendezni. A gyökérelem kijelölése után a fa csomópontjai a mélységi bejárás sorrendjében kiírhatók, ha szükséges, akkor a mélységi szintszám feltüntetésével. Az alábbi példa szerint, ha egy sorban az EMPNO érték megegyezik egy másik sorban található MGR értékkel, akkor az első sor szülője a másodiknak. A fa bejárása a kezdő elemtől, mint gyökértől a levelek irányába halad, és a gyökér valamennyi leszármazottját érinti.

```
SELECT LEVEL , ename, empno, job, mgr
FROM EMP
CONNECT BY PRIOR empno = mgr
START WITH ename = 'KING'
```

A PRIOR kulcsszó áthelyezésével megváltoztatható a bejárási irány. A következő példában a bejárás a kezdő elemtől mint levéltől a gyökér felé halad. Ebben az esetben a bejárás csak a levél őseit érinti.

```
SELECT LEVEL ,ename, empno, job, mgr
FROM emp
CONNECT BY empno = PRIOR mgr
START WITH ename= 'SMITH';
```

A WHERE szakaszban megadott feltételek nem befolyásolják a fa bejárását. Ha le akarunk vágni bizonyos ágakat, akkor a CONNECT szakaszban kell a feltételeket megadni.

```
SELECT ename, job
FROM emp
CONNECT BY PRIOR empno = mgr AND ename != 'SCOTT'
START WITH ename = 'JONES';
```

A műveleteket az SQL*Plus a következő sorrendben hajtja végre:

- Megkeresi a kezdő csomópontot
- Létrehozza a fa struktúrát a kapcsolatnak megfelelően
- Bejárja a fát a megadott irányban
- Levágja a feltétel szerinti ágakat
- Kiválogatja a WHERE szakaszbeli feltételeknek megfelelő sorokat
- Sorbarendezi a sorokat

1.6.3.9. Az unió, a metszet és a különbség

Az egyes lekérdezések által előállított táblák halmazok, az SQL nyelv ezen táblák kombinálására tartalmaz szokásos halmazműveleteket is. Ezek:

UNION	unió
UNION ALL	unió, de nincs szűrés az azonos sorokra
INTERSECT	metszet

MINUS különbség

A műveleteket két SELECT utasítás közé kell írni.

Nem tartoznak szorosan az SQL nyelvhez, de a legtöbb rendszer tartalmaz utasításokat, amelyekkel a lekérdezések által előállított táblázatok megjelenését pl. az oszlopok neveit, szélességét, adatformátumát, illesztését, tördelését definiálhatjuk.

1.6.4. Nézet és index létrehozása

1.6.4.1. Nézet létrehozása

A nézetek olyan virtuális táblák, amelyek a fizikai táblákat felhasználva a tárolt adatok más és más logikai modelljét, csoportosítását tükrözik. Nézetet a

CREATE VIEW ⟨nézetnév⟩ [(⟨oszlopnév1⟩, ...)] AS ⟨lekérdezés⟩;

paranccsal lehet létrehozni. A lekérdezésre az egyedüli megkötés, hogy rendezést nem tartalmazhat, nem lehet benne ORDER BY szakasz. A rendezést a nézet lekérdezésekor kell megadni.

Amennyiben nem adunk meg oszlopneveket, a nézet oszlopai a SELECT után felsorolt oszlopok neveivel azonosak. Meg kell viszont adni az oszlopneveket, ha a SELECT számított értéket is előállít. Például:

```
CREATE VIEW dept_sal (deptno, avg_salary) AS
SELECT deptno, AVG (sal)
FROM emp
GROUP BY deptno;
```

Az alias neveket betehetjük a SELECT utasításba is, azaz:

```
CREATE VIEW dept_sal AS
SELECT deptno, AVG (sal) avg_salary
FROM emp
GROUP BY deptno;
```

A nézetek a lekérdezésekben a táblákkal megegyező módon használhatók, azonban nem tárolnak adatot, az eredeti tábla adatait használják.

Jelentőségük, hogy az adatok más modelljét fejezik ki, felhasználhatók a tárolt információ részeinek elrejtésére, pl. különböző felhasználók más nézeteken keresztül szemlélhetik az adatokat. Ez úgy valósul meg, hogy a létrehozáshoz megadott

SELECT utasítás minden hozzáférés esetén lefut. A nézet általában csak olvasható, az adatmódosító műveletekben csak akkor szerepelhet, hagy egyetlen táblából keletkezett és nem tartalmaz számított értékeket. (Triggerek segítségével szükség esetén megoldható az adatmódosítás.)

A nézet-tábla egy ablakhoz hasonlít, amelyen keresztül a táblákban levő adatok megvizsgálhatók és módosíthatók. Használatukkal a felhasználó számára elkerülhetjük a bonyolult kérdések használatát. Ezen kívül fontos szerepet játszanak a biztonság szempontjából.

Mint említettük, a lekérdezések pontosan úgy végezhetők mint a közönséges táblákon.

```
SELECT ename, job
FROM emp10
WHERE empno > 7800 ;
```

Az adatok a nézet-táblán keresztül közvetlenül módosíthatók, ha a nézet egy táblára vonatkozik.

```
UPDATE emp10
SET job = 'CLERK'
WHERE ename = 'MlLLER';
```

A nézet-táblát több tábla összekapcsolásával is létrehozhatjuk.

```
CREATE VIEW projects ( project, employee, emp_number, location ) AS
SELECT pname, ename, empno, loc
FROM proj, emp, dept
WHERE emp.deptno = dept.deptno AND emp.projno = proj.projno;
```

Kifejezések vagy függvények segítségével úgynevezett virtuális oszlopok, vagy mezők is definiálhatók.

```
CREATE VIEW pay (name, monthly_sal, sal, deptno ) AS
SELECT ename, sal, 12*sal, deptno FROM emp;
```

Ennek használata:

```
SELECT *
FROM pay
WHERE deptno=30;
```

A nézet nincs külön tárolva, lekérdezéskor a nézetben megadott SELECT automatikusan végrehajtódik.

1.6.4.2. Index létrehozása

Az indexek a táblákban keresést gyorsítják meg.
Létrehozásuk:

CREATE [UNIQUE] INDEX ⟨indexnév⟩ ON ⟨táblanév⟩ (⟨oszlopnév1⟩, ...);

paranccsal lehetséges.

Az indexeket az adatbáziskezelő a táblák minden módosításánál frissíti. Ha valamelyik indexet az UNIQUE kulcsszóval definiáltuk, a rendszer biztosítja, hogy az adott oszlopban minden mező egyedi értéket tartalmazzon.

Lehetséges több oszlopot egybefogó, kombinált indexek létrehozása (pl. összetett kulcs esetén). A lekérdezésekben nem jelenik meg, hogy a táblához tartozik-e index, az indexek a létrehozásuk után a felhasználó számára láthatatlanok, csak a lekérdezéseket gyorsítják.

Indexeket azokra az oszlopokra érdemes definiálni, amelyek gyakran szerepelnek keresésekben. Szerencsés esetben az adattáblában nem kell egyáltalán keresni, a kívánt rekord az index alapján közvetlen kiválasztható.
Pl. a

```
SELECT *
FROM emp
WHERE ename = 'JONES';
```

az emp táblában keresés nélkül kiválaszthatja JONES rekordját, ha az ename oszlopra definiáltunk indexet.

1.6.5. Adatelérések szabályozása

Az adatbáziskezelő rendszereket tipikusan több felhasználó használja, ezzel kapcsolatban újabb problémák merülnek fel.

1.6.5.1. Jogosultságok definiálása

A egyes felhasználók részint az adatbáziskezelő rendszerrel, részint az egyes objektumaival különböző műveleteket végezhetnek. Ezeknek a megadására szolgálnak a GRANT utasítások. A

```
GRANT [DBA | CONNECT | RESOURCES ] TO ⟨felhasználó⟩, ...
IDENTIFIED BY ⟨jelszó, ...⟩;
```

paracs az egyes felhasználókat a megadott csoporthoz rendeli, s ezen keresztül szabályozza az adatabázishoz a hozzáférési jogot.

- A DBA jogosultság az adatbázis adminisztrátorokat (DataBase Administrator) definiálja, akiknek korlátlan jogai vannak az összes adatbázis objektum felett, nem csak létrehozhatja, módosíthatja illetve törölheti, de befolyásolhatja az objektumok tárolásával, hozzáférésével kapcsolatos paramétéreket is.
- A RESOURCE jogosultsággal rendelkező felhasználók létrehozhatnak, módosíthatnak ill. törölhetnek új objektumokat,
- míg a CONNECT jogosultság csak az adatbáziskezelőbe belépésre jogosít.

A táblákhoz illetve a nézetekhez a hozzáférést a

```
GRANT ⟨ jogosultság⟩, ...
ON ⟨tábla vagy nézetnév⟩ TO ⟨felhasználó⟩ [WITH GRANT OPTION ];
```

parancs határozza meg.

A ⟨jogosultág⟩ az objektumon végezhető műveleteket adja meg.

Lehetséges értékei:

- ALL
- SELECT ⟨oszlopnev⟩, ...
- INSERT ⟨oszlopnev⟩, ...
- UPDATE ⟨oszlopnév⟩, ...
- DELETE
- ALTER
- INDEX

Az utolsó két művelet nézetekre nem alkalmazható. A felhasználó neve helyett PUBLIC is megadható, amely bármelyik felhasználóra vonatkozik. A PUBLIC egy olyan csoport, amihez minden felhasználó hozzátartozik.

A WITH GRANT OPTION -nal megkapott jogosultságokat a felhasználók tovább is adhatják.

1.6.6. Tranzakciók

Az adatbázisok módosítása általában nem történhet meg egyetlen lépésben, hiszen legtöbbször egy módosítás során több táblában tárolt információn is változtatni akarunk, illetve egyszerre több rekordban akarunk módosítani, több rekordot akarunk beilleszteni. Előfordulhat, hogy módosítás közben valami hibát követünk el, vagy ami még súlyosabb következményekkel jár, hogyha az adatbáziskezelő leáll. Ilyenkor a tárolt adatok inkonzisztens állapotba kerülhetnek, hiszen egyes módosításokat már elvégeztünk, ehhez szorosan hozzátartozó másokat viszont még nem.

1.6.6.1. A tranzakció jellemző tulajdonságai

A tranzakció az adatbázis módosításainak egy sorozata, a munka egy logikai egysége. Ennek négy tulajdonsággal kell rendelkeznie, hogy tranzakciónak nevezhessük. Ezek az atomitás (atomicity), a konzisztencia (consistency), az izoláció (isolation) és a tartósság (durability).

- Atomitás: a tranzakciónak atominak kell lennie abban az értelemben, hogy vagy minden műveletnek végre kell hajtódnia, vagy egynek sem.
- Konzisztencia: a tranzakció befejezése után az adatoknak konzisztens állapotban kell lenniük, azaz az adatok ellentmondás mentesek, csak helyes értékeket tartalmaznak.
- Izoláció: konkurensen futó tranzakciók módosításait el kell különíteni egymástól, tehát egy tranzakción belül olyan adatot kell látni, amit még a konkurensen futó tranzakció elindulása előtt volt, vagy olyat, ami a másik tranzakció lefutása után. Feltételezzük, hogy egy tranzakció elvárt, helyes eredménye az, amit akkor kapunk, ha a tranzakció úgy fut, hogy közben más tranzakció nem fut.
- Tartósság: a tranzakció sikeres befejezése után az adatoknak véglegesnek kell lenniük.

1.6.6.2. A tranzakció érvényesítése és visszaállítása

Az adatbáziskezelők biztosítják, hogy mindig vissza lehessen térni az utolsó teljes egészében végrehajtott tranzakció utáni állapothoz. Egy folyamatban lévő tranzakciót vagy a COMMIT utasítással zárhatjuk le, amely a korábbi COMMIT óta

végrehajtott összes módosítást véglegesíti, vagy a ROLLBACK utasítással törölhetjük a hatásukat, visszatérve a megelőző COMMIT kiadásakor érvényes állapotba. Beállítható, hogy bizonyos műveletek automatikusan COMMIT műveletet hajtsanak végre.

```
SET AUTOCOMMIT [ IM | OFF] ;
```

Az IMM állapotban az ALTER , CREATE , DROP , GRANT és EXIT utasítások sikeres végrehajtása COMMIT -ot is jelent.

A rendszer hardver hiba utáni újraindulásakor illetve hibás INSERT, UPDATE vagy DELETE parancs hatására automatikusan ROLLBACK -et hajt végre. Érdemes tehát minden tranyakció végén COMMIT parancsot kiadni, nehogy egy hibásan kiadott parancs visszavonja a korábbi módosításokat.

Az adatbázist több felhasználó egyidejűleg is használhatja. Ilyenkor az egyes táblákhoz a párhuzamos hozzáférést az izolációs szintnek megfelelő zárolásokkal, hozzáférések tiltásával a rendszer automatikusan beállítja. A párhuzamos hozzáférést az izolációs szintek állításával, vagy egyedileg a táblák, sorok zárolásával tudjuk befolyásolni.

1.6.6.3. Izolációs szintek

Amennyiben a rendszerben nincsenek zárolási mechanizmusok, és egyidejűleg ugyanazt a táblát többen is használják, akkor a következő problémák léphetnek fel:

- **Elveszett módosítások**: ha több tranzakció ugyanazt a sort olvassa, és ez alapján módosítja, akkor a későbbi módosítás felülírja az előzőeket, csak az utolsó marad meg, a többi elvész. A probléma úgy oldható meg, hogy amíg egy tranzakció be nem fejezte a módosítást, a többi nem módosíthat.
- **Piszkos olvasás**: egy tranzakció olyan adatot olvas, és ez alapján akar dolgozni, amit egy másik tranzakció módosított, de még nem véglegesített (nem kommitált), ezért értéke még változhat. A probléma megoldható, ha a második tranzakció csak az első tranzakció által már véglegesített adatot olvashat.
- **Nem ismételhető olvasás**: Egy tranzakció egy sort többször beolvas, miközben egy másik tranzakció ezen adatokat megváltoztatja. Annyiban külön-

bözik a piszkos olvasástól, hogy itt kommittált adatokat olvas. Kiküszöbölhető a probléma, ha a második tranzakció csak akkor olvashatná az adatot, ha az első az írást már befejezte.

- **Fantom olvasás**: akkor fordulhat elő, ha egy tranzakció kitöröl vagy beszúr sorokat egy táblába, miközben egy másik tranzakció ezt olvassa. Ebben az esetben a második tranzakció olyan adattal dolgozik, ami már nem is létezik(közben már kitörölték), illetve olyan adatokhoz nem fér hozzá, amit a másik most hozott létre. Nem lehet ilyen gond, ha a második tranzakció csak az első befejezése után indul el.

A fentieknek megfelelően 4 izolációs szintet definiáltak:

1. **read uncommitted**: Ez a leggyengébb izolációs szint, olvasásra nem kerülnek zárak kiosztásra, igy az utolsó 3 probléma előfordulhat.
2. **read committed**: Olvasásra osztott zárakat használ, igy a piszkos olvasás elkerülhető, csak kommitált adatokat lehet olvasni. Nem ismételhető olvasás illetve fantom olvasás előfordulhat.
3. **repeteable read**: A tranzakció által használt adatokra zárak kerülnek, igy más tranzakció abba nem módosíthat.
4. **serializable**: A legszigorúbb izolációs szint, tábla szintű zárat használ, kiküszöbölve ezzel a beszúrás és törlés okozta fantom olvasást is.

Az alábbi táblázat megmutatja, hogz az egyes izolációs szinteknél milyen probléma léphet fel.

Izolációs szint	**Piszkos olvasás**	**Nem ismételhető olvasás**	**Fantom olvasás**
Read uncommitted	Igen	Igen	Igen
Read committed	Nem	Igen	Igen
Repeteable read	Nem	Nem	Igen
Serializable	Nem	Nem	Nem

1.6.6.4. Táblák, sorok zárolása

Ay izolációs szinteket a felhasználó a feladat igényének megfelelően beállíthatja. Előfordulhat azonban, hogy valamely részfeladatnál a beállított izolációs szintnél szigorúbb zárolást kell alkalmazni, de az egész tranzakciót nem lenne indokolt erre a szigorúbb izolációs szintre tenni, akkor egyedi zárolásokat lehet alkalmazni.

Az ORACLE adatbázis-kezelő rendszerben a táblákhoz való hozzáférést a LOCK paranccsal lehet szabályozni.

```
LOCK TABLE ⟨ táblanév⟩, ... IN
[[ROW ] SHARE | [[SHARE ] ROW ] EXCLUSIVE | SHARE UPDATE ]
MODE [NOWAIT ];
```

A LOCK parancsal egy felhasználó megadhatja, hogy az egyes táblákhoz más felhasználóknak milyen egyidejű hozzáférést engedélyez. Az utasítás végrehajtásánál a rendszer ellenőrzi, hogy a LOCK utasításban igényelt felhasználási mód kompatibilis-e a táblára érvényben lévő kizárással (mások által kiadott LOCK -kal). Amennyiben megfelelő, az utasítás visszzatér és egyéb utasításokat lehet kiadni. Ha az igényelt kizárás nem engedélyezett, az utasítás várakozik amíg az érvényes kizárást megszüntetik, ha csak a parancs nem tartalmazza a NOWAIT módosítót. Ebben az esetben a LOCK utasítás mindig azonnal visszatér, de visszaadhat hibajelzést is.

A táblához hozzáférést az első sikeres LOCK utasítás definiálja. A SHARE módban megnyitott táblákat mások olvashatják, a SHARE UPDATE módban mások módosíthatják is, ilyenkor automatikusan kölcsönös kizárás teljesül egy-egy sorhoz való hozzáférésnél. Az EXCLUSIVE mód kizárólagos hozzáférést biztosít a táblához.

A SELECT ... FOR UPDATE parancs esetén a kiválasztott sorokra kerülnek zárak. A zárolás az UPDATE és a DELETE utasítások előtti időre zárolja a sorokat. A zárak megszünnek egy COMMIT vagy ROLLBACK parancs kiadása esetén is.

A Microsoft SQL adatbáziskezelő esetén a zárolást a SELECT, INSERT, UPDATE vagy DELETE utasításokkal lehet kiadni, a tábla neve után a kulcsszó után. Pl:

```
SELECT ⟨oszlopnév⟩ FROM ⟨táblanév⟩ WITH ⟨zárolási mód⟩ WHERE ...;
```

1.7. Bővítések

1.7.1. Konzisztencia feltételek

A táblák definíciójánál eddig csak azt adtuk meg, hogy milyen adattípusba tartozó értékeket lehet az egyes oszlopokban használni, illetve mely oszlopokban kell feltétlenül értéknek szerepelnie. Célszerű lenne a táblákhoz olyan feltételeket rendelni, amelyek szigorúbb feltételeket definiálnak az egyes adatokra, amelyeket aztán a rendszer a tábla minden módosításánál ellenőriz. Amennyiben a táblák módosításánál valamelyik feltételt megsértenénk, a rendszer hibajelzést ad. Ezeket a feltételeket megkötéseknek (CONSTRAINT) nevezzük, és hogy később tudjunk rájuk hivatkozni, nevük is lehet. Ezeket foglaljuk össze az alábbiakban. Az oszlopszintű megkötések általános formája:

```
[ CONSTRAINT ⟨megkötésnév⟩ ]
{   [ NULL | NOT NULL ]
    [ {PRIMARY KEY  | UNIQUE  } ]
    [ [ FOREIGN KEY ] REFERENCES  < hivatkozott tablanév>
      [ (<hivatkozott oszlopnév>) ]
      [ ON DELETE  {CASCADE | NO ACTION  } ]
      [ ON UPDATE  {CASCADE | NO ACTION  } ]
    ]
    CHECK  (<logikai kifejezés>)
}
```

Itt a logikai kifejezésben csak az adott oszlopra lehet hivatkozni.
A táblaszintű megkötés általános formája:

```
[ CONSTRAINT ⟨megkötésnév⟩ ]
{   [ {PRIMARY KEY  | UNIQUE  }
      { (<oszlopnév> [,...n] ) } ]
    [ FOREIGN KEY [ (<OSZLOPNÉV> [,...] ) ] REFERENCES
      < hivatkozott tablanév> [ (<hivatkozott oszlopnév> [,...n] ) ]
      [ ON DELETE  {CASCADE | NO ACTION  } ]
      [ ON UPDATE  {CASCADE | NO ACTION  } ]
    ]
```

```
    CHECK (<logikai kifejezés>)
}
```

Ebben az esetben a logikai kifejezésben a tábla bármely oszlopára lehet hivatkozni.

1.7.1.1. Elsődleges kulcs és egyedi érték definiálása

Amennyiben az oszlop elsődleges kulcs, azaz PRIMARY KEY -nek definiáljuk, akkor a tábla minden sorában különböző az értéke, és nem lehet NULL értékű. Ha egy mező nem kulcs, szintén előírhatjuk, hogy minden értéke különböző legyen. Ezt az UNIQUE -val tehetjük meg. Ez a mező felvehet NULL értéket is. Pl.:

- Elsődleges kulcs:
  anyag_id INT CONSTRAINT anyag_primary_key PRIMARY KEY , ...
  Ha a megkötésnek nem akarunk nevet adni, akkor
  anyag_id INT PRIMARY KEY , ...
- Egyedi érték előírása:
  anyag_nev VARCHAR2 (100) NOT NULL
  CONSTRAINT anyag_nev_unique UNIQUE , ...
  Itt az anyag_nev a NOT NULL miatt nem vehet fel NULL értéket.

Amennyiben a tábla kulcsa több mezőből áll (összetett kulcs), vagy azt szeretnénk, hogy több mező együttes értéke legyen egyedi, akkor ezt a táblaszintű megkötésben írhatjuk elő.

- Összetett elsődleges kulcs:
  CONSTRAINT tábla_primary_key PRIMARY KEY
  (kulcsmezo1,kulcsmezo2), ...
- Mező együttesre egyedi érték előírása:
  CONSTRAINT mezo1_mezo2_unique UNIQUE (mezo1,mezo2) ...
  Itt, ha szeretnénk, hogy a mező együttes ne lehessen NOT NULL , akkor ezt a mező szintű megkötésben kell megadnunk.

1.7.1.2. Idegen kulcs

Az oszlop idegen kulcs, azaz értéke meg kell, hogy egyezzen egy másik tábla elsődleges kulcs oszlopának valamelyik létező elemével, akkor ezt a REFERENCES táblanév(mezőnév) megkötéssel írhatjuk elő. Pl.:

anyag_egyseg INT CONSTRAINT anyag_egyseg_foreign_key
REFERENCES egyseg(egyseg_id).

Ezel az idegen kulccsal lehet a két táblát összekapcsolni.

Összetett idegen kulcs esetén a táblaszintű megkötésben kapcsolhatjuk össze a két táblát, azaz:

CONSTRAINT összetett_foreign_key
FOREIGN KEY (oszlop1, oszlop2)
REFERENCES hivatkozott_tábla(hivatkozott_oszlop1, hivatkozott_oszlop2).

Idegen kulcs megadásakor opcionálisan megadható a

ON DELETE {CASCADE | NO ACTION }

vagy

ON UPDATE {CASCADE | NO ACTION }

kitétel is. Ez azt jelenti, hogy a hivatkozott táblában elvégzendő törlés vagy módosítás esetén az összekapcsolt tábla hogy viselkedjen.

CASCADE esetn a hivatkozó tábla megfelelő sora törlődik, ha a hivatkozott mező törlésre kerül, illetve a hivatkozó mező megváltozik, ha a hivatkozott mezőt módosítjuk.

NO ACTION esetében, ha a hivatkozott mezőt törölni vagy módosítani akarjuk, hibaüzenetet kapunk, és a törlés illetve a módosítás vissza lesz görgetve mindaddig, amíg van a mezőre hivatkozás.

1.7.1.3. Tartományi ellenőrzés

Az egyes adatok értékészletének az általános adattípusnál pontosabb definícióját adhatjuk meg Ez lehet egy adott intervallumba tarozás, adott halmazba tartozás, ahol a hallmaz lehet egy másik tábla egyik oszlopának értékei. Ezt a tábla definíciójánál egy CHECK feltételben adhatjuk meg. Ha ez egy oszlopra vonatkozik, akkor az oszlop definíciójakor, ha több oszlopot érint, akkor az egyes oszlopok definíciója után kerülhet rá sor.

Pl:

- Egy oszlopra a definició és a megkötés:
  anyag_raktardb FLOAT
  CONSTRAINT anyag_raktardb_check
  CHECK (anyag_raktardb >= 0).
- Több mezőre vonatkozóan (tábla szintű megkötés):
  CONSTRAINT anyag_minmax_check
  CHECK (anyag_minraktar < anyag_maxraktar).

1.7.1.4. Példa a megkötésekre

```
CREATE TABLE egyseg
      ( egyseg_id INT
          CONSTRAINT egyseg_primary_key PRIMARY KEY ,
      egyseg_nev VARCHAR2 (20)
          CONSTRAINT egyseg_nev_unique UNIQUE NOT NULL );
```

```
CREATE TABLE anyag
      anyag_id INT
          CONSTRAINT anyag_primary_key PRIMARY KEY ,
      anyag_nev VARCHAR2 (100)
          NOT NULL CONSTRAINT anyag_nev_unique UNIQUE ,
      anyag_egyseg INT
          CONSTRAINT anyag_egyseg_foreign_key
          REFERENCES egyseg(egyseg_id),
      anyag_egysegar FLOAT (24)
          CONSTRAINT anyag_egysegar_check CHECK (anyag_egysegar>0),
      anyag_raktardb FLOAT
          CONSTRAINT anyag_raktardb_check CHECK (anyag_raktardb >= 0),
      anyag_minraktar FLOAT
          CONSTRAINT  anyag_min_check CHECK (anyag_minraktar >= 0),
      anyag_maxraktar FLOAT
          CONSTRAINT anyag_minmax_check
          CHECK (anyag_minraktar < anyag_maxraktar));
```

1.8. Procedurális adatbázis kezelés

1.8.1. A PL/SQL nyelv

A PL/SQL (Procedural Language/SQL) az Oracle adatbáziskezelő- rendszer procedurális nyelve. Segítségével olyan programokat írhatunk, amelyek tartalmazzák az algoritmikus programozás elemeit, mint programvezérlő utasításokat (feltéte-

lek, ciklusok), alprogramokat (függvények, eljárások) valamint különböző modulokat (blokkokat, csomagokat és objektumokat). Mivel kifejezetten az adatbázis kezeléshez kifejlesztett nyelv, így természetesen nagyon nagymértékben támogatja azt.

A PL/SQL programban a PL/SQL nyelvbeli elemeken kívül használhatunk nagyon sok adatmanipuláló SQL utasítást is, mint pl. a SELECT, INSERT, DELETE, UPDATE stb. utasításokat.

A PL/SQL blokk-orientált nyelv, ami azt jelenti, hogy a programot logikai blokkokra lehet bontani. Egy blokk lehet anonim blokk eljárás vagy függvény. A blokkok természetesen egymásba is ágyazódhatnak, így alakítva ki a program struktúráját Jellemzően egy logikai blokk egy probléma megoldására szolgál. Egy blokk szerkezete:

```
[⟨blokk-fejléc⟩]
[DACLARE
        ⟨konstansok;⟩
        ⟨változók;⟩
        ⟨kurzorok;⟩
        ⟨felhasználó által definiált kivételek;⟩]
BEGIN
        ⟨PL/SQL futtatható utasítások;⟩
[EXCEPTION
        ⟨hibakezelés;⟩]
END;
```

A PL/SQL-ben futtatható utasítások az SQL utasítások, a programvezérlő utasítások, kurzorkezelés, eljárás és függvényhívások, kivétel kezelés. Az SQL SELECT utasítása több előre nem látható számú sort állít elő. A PL/SQL bevezette a kurzor fogalmát, amely segítségével egy lekérdezés által előállított sorokon lehet végiglépkedni. Hosszas magyarázatok helyett nézzünk egy példát. A következő PL/SQL programrészlet kiszámolja a 20-as osztályban kifizetett összes pénzt (fizetés és prémium), megszámolja, hogy hányan kapnak 2000 nál nagyobb fizetést, illetve hány dolgozónak nagyobb a prémiuma, mint a fizetése. Az erdményeket egy átmeneti (temp) táblába rakja.

```
DECLARE
        totalwages NUMBER := 0;
        highpaid NUMBER := 0;
        highercomm NUMBER := 0;
        CURSOR c1 IS SELECT sal, comm
        FROM emp
        WHERE deptno = 20;
BEGIN
        FOR emprecord IN c1 LOOP
                emprecord.comm := NVL (emprecord.comm, 0);
                totalwages := totalwages + emprecord.sal + emprecord.comm;
                IF emprecord.sal > 2000 THEN
                        highpaid := highpaid + 1;
                ENDIF ;
                IF emprecord.comm > emprecord.sal THEN
                        highercomm := highercomm + 1;
                ENDIF ;
        ENDLOOP ;
        INSERT INTO temp VALUES (highpaid, highercomm,
        'Total wages:' || TO_CHAR (totalwages));
        COMMIT ;
END ;
```

Bár a fenti értékeket elő lehetne állítani 3 független lekérdezéssel, de a fenti program csak egyszer halad végig az emp táblán, nem háromszor, ami nagy táblák esetén jelentős futási időkülönbséget jelenthet.

PL/SQL legfontosabb elemei a programvezérlő utasítások, adattípusok, összetett adattípusok, kurzorok, alprogramok, hibák kezelése, triggerek, tranzakciók , csomagok (packages), objektum típusok, dinamikus SQL.

1.8.2. A Transact-SQL nyelv

A Transact-SQL, hasonlóan mint a PL/SQL az oracle adatbáziskezelő- rendszerben, egy programozási lehetőséget teremt a Microsoft SQL Server kezelésében. Ezen túlmenően a T-SQL segítségével kibővültek a szabványos SQL utasítások is. Pl.: az ALTER TABLE utasítás, ami a szabványban csak az oszlop típusának a módosítását és új oszlop hozzáadását engedélyezi, addig a T-SQL-ben a tábla bármilyen tulajdonságát megváltoztathatjuk.
T-SQL segítségével lehetőség van változók definiálására, feltételes utasítások, ciklusok írására valamint tárolt eljárások, triggerek, függvények, indexek készítésére. A T-SQL-ben, ellentétben a PL/SQL-lel nem szükséges az egyes sorokat

pontosvesszővel lezárni, viszont minden változónak @-al kell kezdődnie.
Az alábbi példa az anyag tábla sorait sorra kiírja a kimenetre.

```
declare
        @id INT,
        @nev varchar(20),
        @egyseg char(2),
        @ar FLOAT
DECLARE c1 CURSOR FOR SELECT * FROM anyag;
open c1
fetch c1 into @id, @nev, @egyseg, @ar
WHILE @@FETCH_STATUS = 0
        begin
                print convert(varchar,@id)+': '+@nev+' –
                '+convert(varchar,@ar)+'Ft/'+@egyseg
                fetch c1 into @id, @nev, @egyseg, @ar
        end
close c1
deallocate c1
```

A T-SQL legfontosabb összetevői a programvezérlő utasítások, adattípusok, kurzorok, tárolt eljárások, függvények, triggerek, tranzakciók, hibakezelés.

1.9. Beépített függvények, adatok kezelése

Az alábbiakban az ORACLE rendszer fontosabb beépített függvényeivel és adatkezelésével foglaslkozunk, néhány esetben azonban megemlítjük a Microsoft SQL rendszer sajátságait is. Ezt mindig külön említjük.

1.9.1. Számok kezelése

Aritmetikai kifejezések használhatók a SELECT, a WHERE, és az ORDER BY szakaszban is.

```
SELECT ename, comm/sal, comm, sal
FROM emp
WHERE job = 'SALESMAN'
ORDER BY comm/sal DESC ;
```

1.9.1.1. Az aritmetikai függvények

Az aritmetikai függvények minden sorban kiszámításra kerülnek, és minden sorban külön eredményt adnak.

ABS (balance)	abszolút érték,
GREATEST (sal, comm)	nagyobb érték,
LEAST (sal, comm)	kisebb érték,
ROUND (sal, 2)	kerekítés két tizedes jegyre,
TO_NUMBER ('34.5')	szöveg átalakítása számmá,
TRUNC (sal, 2)	csonkítás két tizedes jegyre,

1.9.1.2. A csoportfüggvények (oszlopfüggvények)

A csoportfüggvények több sorról adnak egyetlen összegző információt. A függvények a NULL értékű mezőket nem veszik figyelembe.

AVG (comm)	az oszlopbeli értékek átlaga,
COUNT (sal)	a sorok száma az oszlopban,
MAX (sal)	a legmagasabb érték az oszlopban,
MIN (sal)	a legkisebb érték az oszlopban,

SUM (comm) az oszlopok elemeinek az összege.

A COUNT (*) az összes sort megszámolja. Ha egyetlen oszlopban sincs NULL -tól különböző érték, az oszlopfüggvények visszatérési értéke NULL, kivéve a COUNT függvényt, ez ilyenkor 0-t ad vissza.
Mód van arra, hogy csak azokat a sorokat számoljuk össze, melyben különböző értékek vannak.

```
SELECT COUNT (DISTINCT job)
FROM emp
WHERE deptno = 30;
```

A csoportfüggvényekkel együtt nem használható olyan kifejezés, amelyik minden sorra külön eredményt ad.

```
SELECT ename, AVG (sal) rossz !!!
```

Ez csak akkor lehetséges, ha csoportosítást alkalmazunk a hivatkozott mezőre, pl:

```
SELECT deptno, AVG (sal)
FROM emp
GROUP BY deptno;
```

Míg a sorok közül a WHERE szakaszban, a csoportok közül a HAVING szakaszban lehet válogatni. Számoljuk meg, hogy egy munkakörben hányan dolgoznak, de csak azokat a munkaköröket vegyük figyelembe, ahol több mint ketten dolgoznak.

```
SELECT job, COUNT (*)
FROM emp
GROUP BY job
HAVING COUNT (*) > 2 ;
```

JOB	count(*)
clerk	4
manager	3
salesman	4

Az adatbáziskezelő a következő sorrendben végzi a feldolgozást :

- l. A WHERE szakaszban szereplő feltételek szerint kiválogatja a sorokat.
- 2. Kialakítja a csoportokat, kiszámítja a csoportfüggvényeket.
- 3. A HAVING szakaszban szereplő feltételek szerint kiválogatja a csoportokat.

A csoportfüggvények oszlopait is el lehet látni névvel. Pl:

```
SELECT job, AVG (sal) "Average Salary"
FROM emp
GROUP BY job;
```

1.9.2. Szövegek kezelése

A szövegek összefűzhetők a || operátorral.

```
SELECT name || '-' || loc departsment
FROM dept;
```

A szövegkonstansokat aposztrófok (') közé kell írni. A szövegben előforduló aposztrófot a jel megduplázásával kell jelölni ('Tom''s diner').

1.9.2.1. Szöveg függvények

DECODE (GRADE,'A',4,'B',3,'C',2,0)	szöveg dekódolása, lásd még a 1.9.4.5. fejezetet.
INITCAP (ename)	nagy kezdőbetű,
INSTR (loc,'YORK')	tartalmazás vizsgálata,
LENGTH (ename)	szöveg hossza,
LOWER (ename)	kisbetűssé alakít,
SOUNDEX ('MAIN')	hangzás szerint alakít át,
SUBSTR (str, 1,2)	részszöveget képez,
UPPER (ename)	nagybetűssé alakít,
LPAD (str,length[,str2])	baloldalon kiegészít(vág) a megadott hosszra [az adott karakterrel]
RPAD (str,length[,str2])	jobboldalon kiegészít(vág) a megadott hosszra [az adott karakterrel]
LTRIM (str,[str2])	a baloldali ures[az adott] karakter[eke]t elhagyja
RTRIM (str)	a jobboldali ures[az adott] karakter[eke]t elhagyja

Példák:

a függvény	az eredmény
lpad('tech', 7);	' tech'
lpad('tech', 2);	'te'
lpad('tech', 8, '0');	'0000tech'
rpad('tech', 7);	'tech '
rpad('tech', 2);	'te'
rpad('tech', 8, '0');	'tech0000'
ltrim(' tech');	'tech'
ltrim('000123', '0');	'123'
ltrim('6372Tech', '0123456789');	'Tech'
rtrim('tech ');	'tech'
rtrim('Techxyxzyyy', 'xyz');	'Tech'

1.9.3. Dátumok kezelése

A dátum standard formátuma: 'DD-MON-YY', 17-FEB-93
A dátum típus az időpontot is magában foglalja (default 12AM). A dátumokkal számolni is lehet. Napokat lehet hozzáadni, illetve levonni. Két dátum kivonható egymásbó1. Ilyenkor az eredmény tört nap is lehet, erre vigyázni kell.

1.9.3.1. Dátum függvények

ADD_MONTHS (D,N) N	hónapot hozzáad a dátumhoz
GREATEST (D1,D2)	kiválasztja a későbbi dátumot,
LEAST (D1,D2)	kiválasztja a korábbi dátumot,
LAST_DAY (D)	a dátum hónapjának utolsó napja,
MONTHS_BETWEEN (D1,D2)	a dátumok közötti hónapok száma,
NEXT_DAY (D,'FRIDAY')	a dátum utáni első péntek,
ROUND (hiredate)	egész napra kerekít,
TO_CHAR (D,'MM/DD/YY')	dátumot szöveggé alakít,
	lásd meg a 1.9.4.3.fejezetet.
TO_DATE (chdate,'MM/DD/YY')	
	szöveget dátummá alakít,
	lásd meg a 1.9.4.2. fejezetet
SYSDATE	napi dátum,

TRUNC (D , [format]) az adott formára csonkol

a függvény	az eredmény
trunc(to_date('22-AUG-03'), 'YEAR')	'01-JAN-03'
trunc(to_date('22-AUG-03'), 'Q')	'01-JUL-03'
trunc(to_date('22-AUG-03'), 'MONTH')	'01-AUG-03'
trunc(to_date('22-AUG-03'), 'DDD')	'22-AUG-03'
trunc(to_date('22-AUG-03'), 'DAY')	'17-AUG-03'

A rendszer nyilvántart egy minden táblából kiválasztható pszeudooszlopot is, amely az aktuális rendszeridőt tartalmazza.

```
SELECT SYSDATE
FROM DUAL ;
```

Ahol a DUAL egy rendszertábla.

1.9.4. Konverziós függvények

NVL

TO_DATE

TO_CHAR

TO_NUMBER

1.9.4.1. A NULL érték kezelése

Két esetben kaphat egy mező, vagy változó NULL értéket.

- ha értéke ismeretlen

- ha az adott sorban nincs értéke

Ha meg akarjuk vizsgálni, hogy egy mező értéke NULL -e, akkor az IS NULL illetve az IS NOT NULL feltételeket kell használni. Az = NULL feltétel nem használható (az összehasonlítás logikai értéke NULL , és ilyenkor egyetlen sort sem kapunk vissza).

```
SELECT ename, sal, comm, job
FROM emp
```

WHERE comm IS NOT NULL ;

A rendezések során, függetlenül az iránytól, mindig a NULL értéket tartalmazó sorok kerülnek a sor elejére. Ha egy kifejezésben egy tag NULL értékű, akkor az egész kifejezés értéke NULL lesz. Az NVL függvénnyel rendelhetünk értéket a NULL értékű mezőkhöz.

```
SELECT ename, job, sal, comm, sal+NVL (comm,0)
FROM emp
WHERE deptno = 30;
```

1.9.4.2. A TO_DATE használata

TO_DATE (string1, [format_mask,] [nls_language])

Szöveg konvertálása dátum adattípussá, formátum string alapján.

Néhány tipikus alkalmazás:

```
TO_DATE ('2001-05-18','yyy-mm-dd')
TO_DATE ('January 15, 1989, 11:00 A.M.', 'Month dd, YYYY, HH:MI A.M.')
```

A legfontosabb formátum paraméterek:

YY, YYY, YYYY:	Év (86, 986, 1986)
MM:	Hónap sorszámmal (1-12)
MON:	Hónap rövidítve
MONTH:	Hónap teljes neve
D:	A hét napja(1-7)
DD:	A hónap napja(1-31)
DDD:	Az év napja (1-366)
HH:	Óra (1-12)
HH24	Óra (0-23)
MI:	Perc (0-59)
SS:	Másodperc

1.9.4.3. A TO_CHAR használata

TO_CHAR (adat, [formátum,] [nyelv])

Az adatmezőben szereplő adatot konvertálja VARCHAR2 adattípusúvá, a formátumnak megfelelően. Dátum, és szám típusú adatok konvertálása VARCHAR2 adattípussá.
Dátum konverziója esetén a formátum megegyezik a TO_DATE függvény formátumával.
Szám konvertálása:

- 0999
- 999.99
- S999G999G999D99L

ahol

- S: Előjel
- L: Pénznem
- G: Csoport elválasztó
- D: Tizedes jegy határoló

1.9.4.4. TO_NUMBER használata

```
TO_NUMBER (szám, [formátum,] [nyelv])
```

Szöveg konvertálása számmá.
A fromátum megegyezik a TO_CHAR függvény formátumával.

1.9.4.5. A Decode használata

```
DECODE (expr,
        value1, retval1,
        value2, retval2, ...
        default value)
```

Szöveg dekódolása, a megadott párokat kódoljuk, ha egy kódolandó érték nem szerepel, utolsó értékre kódoljuk. Pl:

```
DECODE (deptno,
        10, 'ACCOUNTING' ,
```

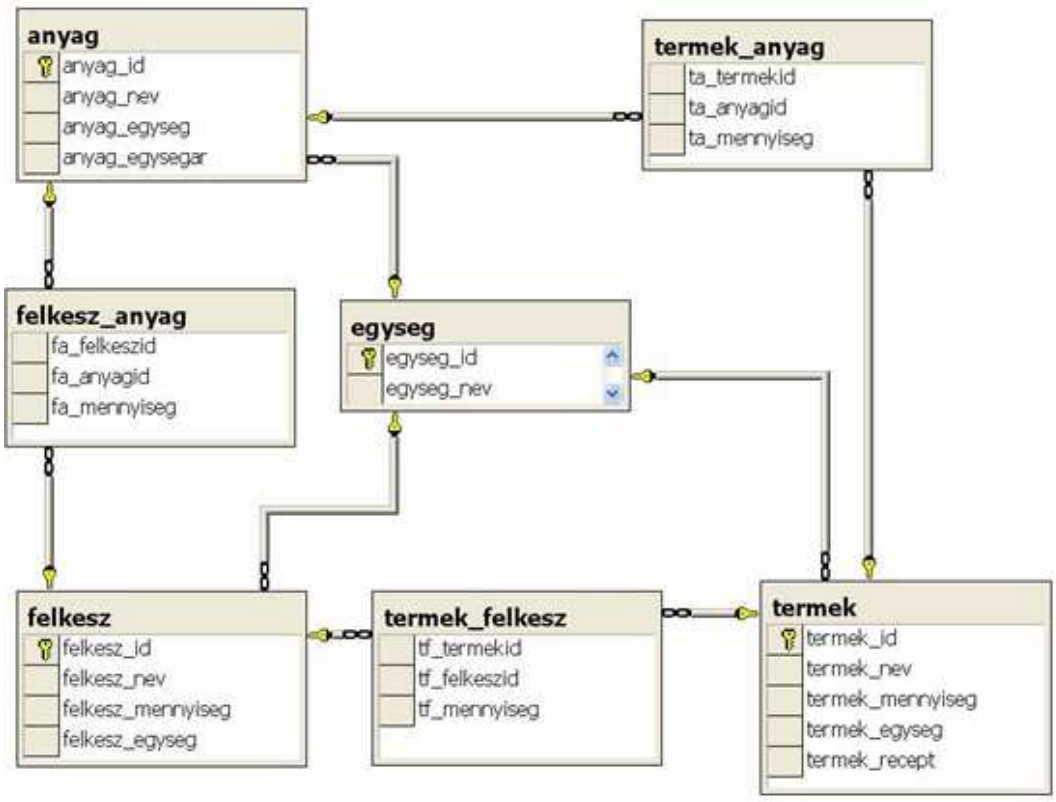
20, 'research' ,
30, 'sales' ,
40, 'operation' ,
'none').

1.10. Példák az SQL-re

1.10.1. A cukrász adatbázis

1.10.1.1. Az adatbázis sémája

A cukrasz adatbázis sémája az 1.10. ábrán látható.



1.10. ábra. A cukrász adatbázis sémája

1.10.1.2. A cukrász adatbázis tábláinak létrehozása

A régi táblák kitörlése

```
DROP VIEW rendeles_lista;
DROP TABLE rendeles;
DROP VIEW termek_recept;
DROP VIEW felkesz_recept;
DROP VIEW anyag_raktar;
DROP VIEW felkesz_raktar;
DROP VIEW termek_lista;
DROP TABLE termek_felkesz;
DROP TABLE termek_anyag;
DROP TABLE termek;
DROP TABLE felkesz_anyag;
DROP TABLE felkesz;
DROP TABLE anyag;
DROP TABLE egyseg;
```

Az új táblák létrehozása

```
CREATE TABLE egyseg (
        egyseg_id INT CONSTRAINT egyseg_primary_key PRIMARY KEY,
        egyseg_nev VARCHAR2 (20)
          CONSTRAINT egyseg_nev_unique UNIQUE NOT NULL );
```

```
CREATE TABLE anyag (
        anyag_id INT
          CONSTRAINT anyag_primary_key PRIMARY KEY,
        anyag_nev VARCHAR2 (100) NOT NULL
          CONSTRAINT anyag_nev_unique UNIQUE,
        anyag_egyseg INT
          CONSTRAINT anyag_egyseg_foreign_key REFERENCES egyseg(egyseg_id),
        anyag_egysegar FLOAT (24)
          CONSTRAINT anyag_egysegar_check CHECK (anyag_egysegar>0),
        anyag_raktardb FLOAT
          CONSTRAINT anyag_raktardb_check CHECK (anyag_raktardb >=0),
        anyag_minraktar FLOAT
          CONSTRAINT anyag_min_check CHECK (anyag_minraktar >= 0),
        anyag_maxraktar FLOAT
          CONSTRAINT anyag_minmax_check CHECK
          (anyag_minraktar < anyag_maxraktar) );
```

```
CREATE TABLE felkesz (
        felkesz_id INT CONSTRAINT felkesz_primary_key PRIMARY KEY,
        felkesz_nev VARCHAR2 (100) NOT NULL
          CONSTRAINT felkesz_nev_unique UNIQUE,
        felkesz_mennyiseg INT,
        felkesz_egyseg INT
          CONSTRAINT felkesz_egyseg_foreign_key REFERENCES egyseg(egyseg_id),
        felkesz_raktardb FLOAT
          CONSTRAINT felkesz_raktardb_check CHECK (felkesz_raktardb >=0),
        felkesz_minraktar FLOAT
          CONSTRAINT felkesz_min_check CHECK (felkesz_minraktar >= 0) );
```

```
CREATE TABLE termek (
        termek_id INT CONSTRAINT termek_primary_key PRIMARY KEY,
        termek_nev VARCHAR2 (100) NOT NULL
          CONSTRAINT termek_nev_unique UNIQUE, termek_mennyiseg INT,
        termek_egyseg INT
          CONSTRAINT termek_egyseg_foreign_key REFERENCES egyseg(egyseg_id),
        termek_recept VARCHAR2 (2000) );
```

```
CREATE TABLE felkesz_anyag (
        fa_felkeszid INT NOT NULL
          CONSTRAINT fa_felkesz_foreign_key REFERENCES felkesz(felkesz_id),
        fa_anyagid INT NOT NULL
          CONSTRAINT fa_anyag_foreign_key REFERENCES anyag(anyag_id),
        fa_mennyiseg FLOAT NOT NULL,
          CONSTRAINT fa_UNIQUE UNIQUE (fa_felkeszid, fa_anyagid) );
```

```
CREATE TABLE termek_anyag (
        ta_termekid INT NOT NULL
          CONSTRAINT ta_termek_foreign_key REFERENCES termek(termek_id),
        ta_anyagid INT NOT NULL
          CONSTRAINT ta_anyag_foreign_key REFERENCES anyag(anyag_id),
        ta_mennyiseg FLOAT NOT NULL
          CONSTRAINT ta_UNIQUE UNIQUE (ta_termekid, ta_anyagid));
```

```
CREATE TABLE termek_felkesz (
        tf_termekid INT NOT NULL
          CONSTRAINT tf_termek_foreign_key REFERENCES termek(termek_id),
        tf_felkeszid INT NOT NULL
          CONSTRAINT tf_felkesz_foreign_key REFERENCES felkesz(felkesz_id),
        tf_mennyiseg FLOAT NOT NULL
          CONSTRAINT tf_unique UNIQUE (tf_termekid, tf_felkeszid) );
```

1.10.1.3. A cukrász adatbázis feltöltése

```
INSERT INTO egyseg values (1,'kg');
INSERT INTO egyseg values (2,'g');
INSERT INTO egyseg values (3,'l');
INSERT INTO egyseg values (4,'dl');
INSERT INTO egyseg values (5,'db');
INSERT INTO egyseg values (6,'lap');
INSERT INTO egyseg values (7,'tekercs');
INSERT INTO egyseg values (8,'szelet');
```

```
INSERT INTO anyag values (1,'cukor',1,186,100,10,200);
INSERT INTO anyag values (2,'liszt',1,42,100,10,200);
INSERT INTO anyag values (3,'vaniliascukor',2,1,100,10,200);
INSERT INTO anyag values (4,'konyak',3,3400,100,10,200);
INSERT INTO anyag values (5,'tojas',5,19,100,10,200);
INSERT INTO anyag values (6,'barack lekvar',1,870,100,10,200);
INSERT INTO anyag values (7,'porcukor',1,216,100,10,200);
INSERT INTO anyag values (8,'sutopor',2,1,100,10,200);
INSERT INTO anyag values (9,'sajt',1,1250,100,10,200);
INSERT INTO anyag values (10,'so',1,56,100,10,200);
INSERT INTO anyag values (11,'kakao',2,2,100,10,200);
INSERT INTO anyag values (12,'vaj',2,1.5,100,10,200);
INSERT INTO anyag values (13,'viz',3,0.05,100,10,200);
INSERT INTO anyag values (14,'zselepor',2,2,100,10,200);
INSERT INTO anyag values (15,'barack kompot',1,250,100,10,200);
INSERT INTO anyag values (16,'banan',1,250,100,10,200);
INSERT INTO anyag values (17,'megy befott',1,450,100,10,200);
INSERT INTO anyag values (18,'margarin',1,650,100,10,200);
INSERT INTO anyag values (19,'vanilia',2,3,100,10,200);
INSERT INTO anyag values (23,'dio',1,350,100,10,200);
INSERT INTO anyag values (24,'tej',3,156,100,10,200);
INSERT INTO anyag values (25,'tejszin',4,98,100,10,200);
```

```
INSERT INTO felkesz values (1,'piskota',1,6,10,3);
INSERT INTO felkesz values (2,'csoki krem',1,3,10,3);
INSERT INTO felkesz values (3,'vanilia krem',1,3,10,3);
INSERT INTO felkesz values (4,'gyumolcstorta alap',2,5,10,3);
INSERT INTO felkesz values (5,'tejszinhab',4,4,10,3);
```

```
INSERT INTO felkesz_anyag values (1,5,6);
INSERT INTO felkesz_anyag values (1,2,0.25);
INSERT INTO felkesz_anyag values (1,1,0.06);
INSERT INTO felkesz_anyag values (4,1,0.25);
INSERT INTO felkesz_anyag values (4,5,5);
INSERT INTO felkesz_anyag values (4,3,12);
INSERT INTO felkesz_anyag values (4,2,0.25);
INSERT INTO felkesz_anyag values (4,8,12);
INSERT INTO felkesz_anyag values (4,17,0.5);
INSERT INTO felkesz_anyag values (2,12,1750);
INSERT INTO felkesz_anyag values (2,11,750);
INSERT INTO felkesz_anyag values (2,24,1);
INSERT INTO felkesz_anyag values (2,1,0.5);
INSERT INTO felkesz_anyag values (5,25,4);
INSERT INTO felkesz_anyag values (5,1,0.1);
```

```
INSERT INTO termek values (1,'Piskotatekercs',2,7,' ');
INSERT INTO termek values (2,'Csokitorta',12,8,' ');
INSERT INTO termek values (3,'Kepviselofank',20,5,' ');
INSERT INTO termek values (4,'Gyumolcstorta',8,8,' ');
```

```
INSERT INTO termek_anyag values (1,6,0.3);
INSERT INTO termek_anyag values (1,7,0.05);
INSERT INTO termek_anyag values (4,15,0.5);
INSERT INTO termek_anyag values (4,6,0.1);
INSERT INTO termek_anyag values (4,14,12);
INSERT INTO termek_anyag values (4,16,0.2);
INSERT INTO termek_anyag values (2,7,0.02);
INSERT INTO termek_anyag values (2,17,0.15);
```

```
INSERT INTO termek_felkesz values (1,1,2);
INSERT INTO termek_felkesz values (4,4,1);
INSERT INTO termek_felkesz values (2,1,3);
INSERT INTO termek_felkesz values (2,2,1.2);
INSERT INTO termek_felkesz values (2,5,1);
```

```
COMMIT ;
```

1.10.1.4. Feladatok és megoldások a cukrász adatbázishoz

1. feladat

A táblák létrehozása és feltöltése.

Futtassa le a fenti utasításokat, ami létrehozza a cukrász adatbázis táblákat, és

feltölti egységes minta adatokkal!

2. feladat

Listázás rendezéssel.

Listázza ki, megnevezés szerint fordított ABC sorrendben, hogy milyen félkész termékek vannak (minden adatot)!

Megoldás:

```
COLUMN felkesz_nev FORMAT A20 ;
SELECT *
FROM felkesz
ORDER BY felkesz_nev DESC ;
```

FELKESZ_ID	FELKESZ_NEV	FELKESZ_MENNYISEG	FELKESZ_EGYSEG	FELKESZ_RAKTARDB	FELKESZ_MINRAKTAR
3	vanilia krem	1	3	10	3
5	tejszinhab	4	4	10	3
1	piskota	1	6	10	3
4	gyumolcstorta alap	2	5	10	3
2	csoki krem	1	3	10	3

3. feladat

Listázás WHERE feltétellel.

Listázza ki, hogy a gyümölcstorta alap milyen alapanyagokból áll, és melyikből mennyi kell! (név, mennyiség, egység)

Megoldás:

```
COLUMN anyag_nev FORMAT A20 ;
SELECT fa_mennyiseg, anyag_egyseg, anyag_nev
FROM anyag, felkesz_anyag, felkesz
WHERE anyag_id=fa_anyagid AND felkesz_id=fa_felkeszid
  AND LOWER (felkesz_nev) LIKE 'gyumolcstorta alap';
```

FA_MENNYISEG	ANYAG_EGYSEG	ANYAG_NEV
0.25	1	cukor
0.25	1	liszt
12.00	2	vaniliascukor
5.00	5	tojas
12.00	2	sutopor
0.50	1	megy befott

4. feladat

Az UNION alkalmazása.

Listázza ki azokat a termékeket, amihez kell vaníliáscukor!

Megoldás:

```
COLUMN termek_nev FORMAT A20 ;
SELECT termek_nev
FROM termek, anyag, termek_anyag
WHERE ta_anyagid=anyag_id AND ta_termekid=termek_id
 AND LOWER (anyag_nev)='vaniliascukor'
UNION
SELECT termek_nev
FROM termek, felkesz, termek_felkesz, anyag, felkesz_anyag
WHERE tf_termekid=termek_id AND tf_felkeszid=felkesz_id
 AND fa_anyagid=anyag_id AND fa_felkeszid=felkesz_id
AND LOWER (anyag_nev)='vaniliascukor';
```

A termékben egyrészt lehet közvetlen vaníliáscukor, ezt a termek_anyag tábla alapján találhatjuk meg, de lehet, hogy egy termékben azért van vaníliás cukor, mert a termék által tartalmazott félkész termékben van vaníliáscukor. Ezt a termek_felkesz tábla alapján tudjuk meghatározni. A két lekérdezés uniója lesz a feladat eredménye.

TERMEK_NEV
Gyumolcstorta

5. feladat

A NOT IN használata.

Listázza ki azokat a termékeket, amikhez nem kell vaníliáscukor!

Megoldás:

```
SELECT termek_nev
FROM termek
WHERE termek_nev NOT IN
( SELECT termek_nev
FROM termek, anyag, termek_anyag
WHERE ta_anyagid=anyag_id AND ta_termekid=termek_id
 AND LOWER (anyag_nev)='vaniliascukor'
UNION
SELECT termek_nev
FROM termek, felkesz, termek_felkesz, anyag, felkesz_anyag
WHERE tf_termekid=termek_id AND tf_felkeszid=felkesz_id
 AND fa_anyagid=anyag_id AND fa_felkeszid=felkesz_id
 AND LOWER (anyag_nev)='vaniliascukor' );
```

TERMEK_NEV
Piskotatekercs
Csokitorta
Kepviselofank

vagy

```
SELECT termek_nev
FROM termek
MINUS
SELECT termek_nev
FROM termek, anyag, termek_anyag
WHERE ta_anyagid=anyag_id AND ta_termekid=termek_id
 AND anyag_nev='vaniliascukor'
MINUS
SELECT termek_nev
FROM termek, felkesz, termek_felkesz, anyag, felkesz_anyag
WHERE tf_termekid=termek_id AND tf_felkeszid=felkesz_id
 AND fa_anyagid=anyag_id AND fa_felkeszid=felkesz_id
 AND LOWER(anyag_nev)='vaniliascukor'
```

TERMEK_NEV
Csokitorta
Kepviselofank
Piskotatekercs

6. feladat

Oszlopfüggvények használata.

Listázza ki a legolcsóbb anyagot!

Megoldás:

```
SELECT anyag_egysegar, anyag_nev
FROM anyag
WHERE anyag_egysegar=
(SELECT MIN(anyag_egysegar) FROM anyag);
```

vagy

```
SELECT anyag_egysegar, anyag_nev
FROM anyag
WHERE anyag_egysegar<= ALL (SELECT anyag_egysegar FROM anyag);
```

ANYAG_EGYSEGAR	ANYAG_NEV
.05	viz

7. feladat

Az UNION és az ORDER BY használata.

Listázza ki, az alapanyag neve szerint sorrendben, hogy a gyümölcstorta milyen alapanyagokból készül (név, mennyiség)!

Megoldás:

```
SELECT anyag_nev, ta_mennyiseg as mennyiseg, anyag_egyseg
FROM anyag, termek_anyag, termek
WHERE anyag_id=ta_anyagid AND ta_termekid=termek_id
 AND UPPER (termek_nev)=UPPER ('gyumolcstorta')
UNION ALL
SELECT anyag_nev, tf_mennyiseg/felkesz_mennyiseg, anyag_egyseg
FROM anyag, termek_felkesz, termek, felkesz, felkesz_anyag
WHERE felkesz_id=tf_felkeszid AND tf_termekid=termek_id
 AND UPPER (termek_nev) = UPPER ('gyumolcstorta')
```

```
  AND felkesz_id=fa_felkeszid AND fa_anyagid=anyag_id
ORDER BY anyag_nev;
```

ANYAG_NEV	MENNYISEG	ANYAG_EGYSEG
banan	0.20	1
barack kompot	0.50	1
barack lekvar	0.10	1
cukor	0.50	1
liszt	0.50	1
megy befott	0.50	1
sutopor	0.50	2
tojas	0.50	5
vaniliascukor	0.50	2
zselepor	12.00	2

8. feladat

A SUM és a GROUP BY használata.

Számolja ki, hogy egy gyümölcstorta alap mennyibe kerül!

Megoldás:

```
SELECT SUM ((fa_mennyiseg*anyag_egysegar))/felkesz_mennyiseg AS ar
FROM felkesz, felkesz_anyag, anyag
WHERE anyag_id=fa_anyagid AND fa_felkeszid=felkesz_id
  AND UPPER (felkesz_nev)=UPPER ('gyumolcstorta alap')
GROUP BY felkesz_mennyiseg;
```

vagy

```
SELECT sum((fa_mennyiseg*anyag_egysegar)/felkesz_mennyiseg) AS ar
FROM felkesz, felkesz_anyag, anyag
WHERE anyag_id=fa_anyagid AND fa_felkeszid=felkesz_id
  AND LOWER (felkesz_nev)='gyumolcstorta alap';
```

AR
200.5

9. feladat

A SUM és a GROUP BY használata.

Számolja ki, hogy az egyes félkész termékek hány forintba kerülnek!

Megoldás:

```
SELECT felkesz_nev, SUM ((fa_mennyiseg*anyag_egysegar)/felkesz_mennyiseg) AS
ar
FROM felkesz, felkesz_anyag, anyag
WHERE anyag_id=fa_anyagid AND fa_felkeszid=felkesz_id
GROUP BY felkesz_nev;
```

FELKESZ_NEV	AR
csoki krem	4374
gyumolcstorta alap	200.5
piskota	135.66
tejszinhab	102.65

10. feladat

A HAVING használata.
Listázza ki, hogy mely félkész termékek olcsóbbak 150 Ft-nál!
Megoldás:

```
SELECT felkesz_nev, SUM ((fa_mennyiseg*anyag_egysegar)/felkesz_mennyiseg) AS
ar
FROM felkesz, felkesz_anyag, anyag
WHERE anyag_id=fa_anyagid AND fa_felkeszid=felkesz_id
GROUP BY felkesz_nev
HAVING SUM ((fa_mennyiseg*anyag_egysegar)/felkesz_mennyiseg) < 150;
```

FELKESZ_NEV	AR
piskota	135.66
tejszinhab	102.65

11. feladat

Átmeneti táblák használata.
Számolja ki, hogy egy gyümölcstorta mennyibe kerül!
Megoldás:

```
SELECT tabla1.ar+tabla2.ar
FROM
(SELECT SUM ((fa_mennyiseg*anyag_egysegar)/felkesz_mennyiseg) AS ar
FROM felkesz, felkesz_anyag, anyag, termek_felkesz, termek
WHERE anyag_id=fa_anyagid AND fa_felkeszid=felkesz_id AND
 tf_felkeszid=felkesz_id AND tf_termekid=termek_id
 AND termek_nev='Gyumolcstorta' ) tabla1,
(SELECT SUM (ta_mennyiseg*anyag_egysegar) AS ar
FROM anyag, termek_anyag, termek
WHERE anyag_id=ta_anyagid AND ta_termekid=termek_id AND
 termek_nev='Gyumolcstorta' ) tabla2;
```

TABLA1.AR+TABLA2.AR
486.5

vagy

xxx

```
SELECT SUM (ar)
      FROM
      (SELECT ((fa_mennyiseg*anyag_egysegar)/felkesz_mennyiseg)AS ar
      FROM felkesz, felkesz_anyag, anyag, termek_felkesz, termek
      WHERE anyag_id=fa_anyagid AND fa_felkeszid=felkesz_id AND
            tf_felkeszid=felkesz_id AND tf_termekid=termek_id AND
            termek_nev='Gyumolcstorta'
UNION ALL
SELECT (ta_mennyiseg*anyag_egysegar) AS ar
      FROM anyag, termek_anyag, termek
      WHERE anyag_id=ta_anyagid AND ta_termekid=termek_id AND
            termek_nev='Gyumolcstorta');
```

xxx

SUM (**AR**)
486.5

12. feladat

Átmeneti táblák használata.
Számolja ki, hogy az egyes termékek hány forintba kerülnek!
Megoldás:

```
COLUMN termek_nev FORMAT A20 HEADING 'Termék';
SELECT tabla1.termek_nev, tabla1.ar+tabla2.ar as "össz ár(Ft)"
FROM
(SELECT SUM ((fa_mennyiseg*anyag_egysegar)/felkesz_mennyiseg) AS ar,
termek_nev
FROM felkesz, felkesz_anyag, anyag, termek_felkesz, termek
WHERE anyag_id=fa_anyagid AND fa_felkeszid=felkesz_id AND
 tf_felkeszid=felkesz_id AND tf_termekid=termek_id
GROUP BY termek_nev ) tabla1,
(SELECT SUM (ta_mennyiseg*anyag_egysegar) AS ar, termek_nev
FROM anyag, termek_anyag, termek
WHERE anyag_id=ta_anyagid AND ta_termekid=termek_id
GROUP BY termek_nev ) tabla2
WHERE tabla1.termek_nev=tabla2.termek_nev;
```

Termék	**össz ár(Ft)**
Csokitorta	4616.63
Gyumolcstorta	486.5
Piskotatekercs	407.46

1.11. Függelék

1.11.1. Minta táblák

EMP - a személyek adatait tartalmazza

DEPT - az egyes osztályok adatait tartalmazza

BONUS - a jutalmakat tartalmazza

SALGRADE - a fizetési csoportokat tartalmazza.

A **DEPT** tábla

DEPTNO	DNAME	LOC
10	ACCOUNTING	MEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

A **BONUS** tábla

EMAME	JOB	SAL	COMM
SMITH	CLERK	800.00	300.00
ALLEN	SALESMAN	1,600.00	300.00
WARD	SALESMAN	1,250.00	500.00
JONES	MANAGER	1,875.00	
MARTIN	SALESMAN	1,250.00	1,l00.00

A **SALGRADE** tábla

GRADE	OSAL	HISAL
1	700	1200
2	1201	1100
3	1101	2000
4	2001	3000
5	3001	8995

Az **EMP** tábla

EMP NO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPT NO
7329	SMITH	CLERK	7902	17-DEC-80	800.00		20
7499	ALLEN	SALESMAN	7698	20-FE8-81	1.600.00	300.00	30
7521	WARD	SALESMAN	7698	22-FE8-81	1,250.00	500.00	30
7566	JONES	MANAGER	7839	02-APR-81	2.975.00		20
7654	MARTIN	SALESMAN	7698	28-5EP-81	1,250.00	1,400.00	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2,850.00		30
7782	CLARK	MANAGER	7839	09-JUN-81	2,450.00		10
7788	SCOTT	ANALYST	7566	09-DEC-82	3,000.00		20
7839	KING	PRESIDENT		17-NOV-81	5,000.00		10
784	TURMER	SALESMAN	7698	O8-SEP-81	1,500.00		30
7876	ADAMS	CLERK	7788	12-JUN-83	1.100.00		20
7900	JAMES	CLERK	7698	03-DEC-81	950.00		30
7902	FORD	ANALYST	7566	03-DEC-81	3,000.00		20
7934	MILLER	CLERK	7782	23-JAN-82	1,300.00		10

1.11.2. A számok formátuma

Formátum	**Érték**	**Megjelenítés**	**Magyarázat**
999.99	56.478	56.48	2 tizedesre kerekítés
999V99	56.478	5648	Az érték szorzása 100-zal
9,999	8410	8,410	3 jegyenként elválasztójel
9,999	639	639	1000 alatt nincs elválasztás
99999	607	607	Normál formátum
09999	607	0607	Vezető 0 érték kiírása
9999	-5609	-5609	Mínusz jel elől
9999MI	-5609	5609-	Mínusz jel hátul
9999PR	-5609	<5609>	Negatív szám zárójelben
B999	0		Zérus érték elnyomása
B999	564	564	Normál kiírás
%99.99	124.98	##.##	Túl nagy érték jelzése
$99.99	4523	$4523	Dollár jel kiírása
$99.99PR	-45.23	<$45.23>	Kombinált formátum
9.99EEEE	12000	1.20E + 04	Exponenciális kijelzés

Megjegyzés: COLUMN egy SQL*Plus parancs, nem SQL parancs. Az oszlop formáutm érvényes a törlésig, új formátum beállításáig vagy az SQL*Plus-ból való kilépésig.

1.11.3. Dátum formátumok

Elemen	Meaning
YYYY vagy SYYYY	Év
'S'	előtag, dátum előjellel(-)
YYY, YY vagy Y	Utolso 3, 2, vagy 1 jegye az évnek
SYEAR vagy YEAR	Év, betűvel kiírva
'S'	előtag, dátum előjellel(-)
Q	Negyedév
MM	Honap
MONTH vagy MON	A hónap neve, vagy 3-betűs rövidítése
DDD DD vagy D	Hányadik nap az évben, a hónapban, vagy a héten
DAY vagy DY	A nap neve, vagy 3-betűs rövidítése
AM vagy PM	Délelőtt, délután jelölése (Meridian indicator)
A.M. vagy P.M.	Meridian indicator
HH vagy HH12	12 órás megjelenítás (1-12)
HH24	24 órás megjelenítás (0-23)
MI	Perc (Minute)
SS	Másodperc (Second)