Értékünk az ember

Humánerőforrás-fejlesztési Operatív Program

Kiss Jenő

ADATBÁZIS-KEZELÉS

SZÉCHENYI ISTVÁN EGYETEM
GYŐR

Magyarország célba ér

NEMZETI FEJLESZTÉSI TERV

Készült a HEFOP 3.3.1-P.-2004-09-0102/1.0 pályázat támogatásával.

Szerző: dr. Kiss Jenő
egyetemi adjunktus

Lektor: dr. Raffai Mária
főiskolai tanár

A dokumentum használata

Mozgás a dokumentumban

A dokumentumban való mozgáshoz a Windows és az Adobe Reader megszokott elemeit és módszereit használhatjuk.

Minden lap tetején és alján egy navigációs sor található, itt a megfelelő hivatkozásra kattintva ugorhatunk a használati útmutatóra, a tartalomjegyzékre, valamint a tárgymutatóra. A ◀ és a ▶ nyilakkal az előző és a következő oldalra léphetünk át, míg a Vissza mező az utoljára megnézett oldalra visz vissza bennünket.

Pozícionálás a könyvjelzőablak segítségével

A bal oldali könyvjelző ablakban tartalomjegyzékfa található, amelynek bejegyzéseire kattintva az adott fejezet/alfejezet első oldalára jutunk. Az aktuális pozíciónkat a tartalomjegyzékfában kiemelt bejegyzés mutatja.

A tartalomjegyzék és a tárgymutató használata

Ugrás megadott helyre a tartalomjegyzék segítségével

Kattintsunk a tartalomjegyzék megfelelő pontjára, ezzel az adott fejezet első oldalára jutunk.

A tárgymutató használata, keresés a szövegben

Keressük meg a tárgyszavak között a bejegyzést, majd kattintsunk a hozzá tartozó oldalszámok közül a megfelelőre. A további előfordulások megtekintéséhez használjuk a Vissza mezőt.

A dokumentumban való kereséshez használjuk megszokott módon a Szerkesztés menü Keresés parancsát. Az Adobe Reader az adott pozíciótól kezdve keres a szövegben.

Tartalomjegyzék

1. Alapok

1.1. Történeti áttekintés

A 60-as évek elejétől, a harmadik generációs számítógépek (1964–1975) megjelenésével, a számítógépeket már egyre nagyobb mértékben adatfeldolgozásra használták. A gazdaságban egyre több adat keletkezett, amit a korábbi módszerekkel már nem lehetett feldolgozni. Ezek a gépek a megfelelő **hardver** (integrált áramkörök, nagykapacitású perifériák) és **szoftver** (operációs rendszer, több felhasználós alkalmazások) erőforrásaikkal már nagymennyiségű adat tárolására, feldolgozására voltak képesek.

Az egyedi, adott feladat megoldására készített, alkalmazások helyett megfogalmazódott az igény *nagy osztott adatbázis-kezelő rendszerek* létrehozására, amelyek „általánosan" kezelik az adatokat. A különböző adathalmazok egységes kezelésére **adatmodellek** születtek, majd az adatmodellekre épülő adatbázisok kezelésére adatbázis-kezelő programrendszerek.

1.2. Alapfogalmak

Az alábbiakban definiáljuk az adatkezeléssel kapcsolatos legalapvetőbb fogalmakat:

(Adat)egyed (entity)

> Minden olyan dolog (objektum), ami minden más dologtól (objektumtól) megkülönböztethető. Pl.: személy, autó stb.

Egyedhalmaz/osztály (entity set/class)

> „Hasonló" egyedek (objektumok) halmaza.

Tulajdonság (attribute)

> Az egyedeket tulajdonságokkal (attribútumokkal) írjuk le. Az autó egyedtípus esetén pl.: típus, rendszám, szín, alvázszám stb.

Kulcs (key)

Ha egy vagy több tulajdonság egyértelműen meghatározza, hogy az egyed melyik értékéről van szó, akkor azokat **kulcsnak** nevezzük. Például autónál az alvázszám tulajdonság kulcs.

Adattípus (data type)

A dologra (objektumra) jellemző tulajdonságtípus.

Adatérték (data value)

Az adott dolog (objektum) egy tulajdonsága. Például egy konkrét autó színe *kék*.

Kapcsolat (relationship)

Az egyedek (egyedhalmazok) közötti viszony fogalmi tükörképe. Fajtái:

Egy-az-egyhez kapcsolat (1 : 1)

Egy-a-többhöz kapcsolat (1 : N)

Több-a-többhöz kapcsolat (M : N)

Egy-az-egyhez kapcsolat

Egy-az-egyhez (**A-B**) kapcsolat esetén az **A** egyedhalmaz (szülő) minden egyes eleméhez legfeljebb egy elem tartozhat a **B** egyedhalmazban (gyerek), és a **B** egyedhalmaz minden egyes eleméhez is csak legfeljebb egy elem tartozhat az **A** egyedhalmazban. Egy-az-egyhez típusú kapcsolatot általában az alábbi esetekben használunk:

- Egy sok elemből álló egyedhalmazt több kisebb, könnyebben kezelhető egyedhalmazra kívánunk felosztani.
- Egy egyedhalmaz valamely részét adatvédelmi megfontolásból külön kívánjuk tárolni.
- Az egyik egyedhalmazban (B) olyan adatokat szeretnénk tárolni, amely a fő egyedhalmazban (A) csak bizonyos elemekre érvényes.

Egy-a-többhöz kapcsolat

Az egy-a-többhöz (**A-B**) kapcsolat a leggyakrabban használatos kapcsolati típus. Az egy-a-többhöz kapcsolatban az **A** egyedhalmaz valamely eleméhez több elem tartozhat a **B** egyedhalmazban, de a **B** egyedhalmaz valamennyi eleméhez csak egy-egy elem tartozhat az **A** egyedhalmazban.

Több-a-többhöz kapcsolat

Több-a-többhöz (**A-B**) kapcsolat esetén az **A** egyedhalmaz valamely eleméhez több elem is tartozhat a **B** egyedhalmazban, és a **B** egyedhalmaz valamely eleméhez is több elem tartozhat az **A** egyedhalmazban. A több-a-többhöz kapcsolat felbontható két egy-a-többhöz kapcsolatra.

Adatmodell

Egyedek, tulajdonságok és kapcsolatok halmaza, amely absztrakt módon tükrözi a valós objektumoknak, azok jellemzőinek (tulajdonságainak) és viszonyainak (kapcsolatainak) elvont kategóriáit.

Adatbázis

Az adatok kapcsolataikkal együtt való tárolása, kezelése adatbázis-kezelő szoftverrendszerrel.

1.3. Korai adatmodellek

1.3.1. A hierarchikus modell

A hierarchikus adatmodell szerkezetét *fa gráffal* adjuk meg. A gráfban a csomópontokat az egyedek, az éleket a kapcsolatok jelentik. Az egyedeket tulajdonságaikkal írjuk le.

Megvalósítás: IBM IMS -1969.

1.3.2. A hálós modell

A hálós adatmodell szerkezetét *gráffal* adjuk meg. A gráfban a csomópontokat az egyedek, az éleket a kapcsolatok jelentik. Az egyedeket tulajdonságaikkal írjuk le.

A modellel kapcsolatos elvárásokat a Conference on Data System Languages (CODASYL) ajánlások (1971.) tartalmazzák. Megvalósítások: ADABAS, DEC DBMS, IDMS.

1.3.3. A relációs modell

E. F. Codd (IBM) 1970-ben publikált egy híres cikket: A Relational Model for Large Shared Data Banks (Nagy, osztott adatbankok egy relációs modellje), amely egy lekérdező nyelv fő kritériumait tartalmazta. Az adatbázisok az adatokat táblázatok formájában jelenítik meg. A modell matematikai alapjait a *relációs algebra* adja.

1976-ban az IBM-nél megszületett a SEQUEL adatbázisnyelv (E. F. Codd, P. Chen, C. Date), amit később **SQL** (Structured Query Language)-re rövidítettek, amely:

- Szabványos adatszerkezet definiálási, adatkezelési és adatbiztonsági lehetőségeket biztosít.
- A fizikai műveleteket rejtse el a lekérdezést, karbantartást (felvitel, módosítás, törlés) végző utasítások mögé.

A relációs adatmodellben az egyedet egy táblázattal adjuk meg. A táblázat oszlopai az egyedre jellemző tulajdonságok, a sorai az egyed értékei (egyedtípus előfordulásai). A táblák közötti kapcsolatokat közös tulajdonságokkal, indexeken keresztül, valósítjuk meg.

A alábbi felsorolás néhány fontos fejlesztő céget és részben azonos nevű SQL terméküket tartalmazza:

- IBM-1983: DB2
- Oracle-1983: Oracle
- Relational Co. - 1984: INFORMIX
- Sybase
- Ingres
- Microsoft: SQL-Server

Időközben már néhány cég megszűnt, terméküket mások felvásárolták.

SQL szabványosítás

Miután egyre több cég jelent meg saját fejlesztésű relációs adatbázis-kezelő rendszerrel, lassan elindult egy szabványosítási folyamat. Az első **ANSI (1986),** majd az **ISO (1987)** szabvány csak a nyelv alaputasításait tárgyalta.

Az 1989-ben megjelent **ISO: 9075:1989** szabvány már foglalkozik az előfordítókkal és az ún. *dinamikus* SQL utasításokkal is.

A jelenlegi szoftverek, az 1992-ben bejelentett **ISO: 9075:1992** szabvány elvárásainak megfelelően, bővített adattípusokkal, értékszabályokkal, kulcsdefiniálási lehetőségekkel, stb. dolgoznak. A szabványt az irodalom SQL2 néven említi.

Az SQL3 szabvány 1998/99-re készült el. Kezeli az összetett, rekurzív adatszerkezeteket, megjelenik benne az objektumorientált adatbázis-kezelés.

Az ODMG (Object Database Management Group) szabványok (1993–2003) tartalmazzák az objektumorientált adatbázisokra vonatkozó elvárásokat, követelményeket.

1.4. Objektumorientált modell

A modellek rendelkeznek az objektumorientált programozási nyelvek összes tulajdonságával.

1.5. Objektum-relációs modell

A modell objektumközpontú elemekkel, tulajdonságokkal (osztályok, egységbezárás, öröklés stb.) bővíti ki a relációs modellt.

Az osztály a relációnak felel meg. Az osztályok példányai az objektumok. Az objektumok közötti kapcsolatokat gyakran *asszociációknak* nevezzük. Lehetővé teszi összetett adattípusok használatát.

1.6. Adatbázis-kezelő rendszerek

1.6.1. Elvárások

Adatbázis-kezelő (DBMS – Database Management System) rendszerrel szembeni elvárásaink:

1. Lehessen új adatbázisokat létrehozni és azok sémáját (az adatok logikai struktúráját) egy nyelv utasításai segítségével megadni. Ezek a nyelv **adatdefiníciós** utasításai.
2. Az adatokat nyelvi utasítások segítségével lekérdezhessük és módosíthassuk. Ezek a nyelv **adatmanipulációs** utasításai.

3. Biztosítsa az adatok hosszú időn keresztül való tárolását. Védje az **adatokat** a meghibásodásokkal és az illetéktelen felhasználókkal szemben.
4. **Felügyelje** több-felhasználós környezetben az egyidejű adathozzáféréseket. A felhasználók műveletei ne legyenek hatással más felhasználók műveleteire. Az egyidejű adathozzáférések ne okozzák az adatok hibássá vagy következetlenné válását (feleljenek meg bizonyos szabályoknak, megszorításoknak).

1.6.2. Adatbázis-kezelő rendszerek főbb részei

Az adatbázis-kezelő rendszerek főbb részei:

- Lekérdezés feldolgozó
- Tárkezelő
- Tranzakció-kezelő
- Adatok, metaadatok

Az adatbázis az adatokon kívül tárolja a katalógus adatait (metaadatok) is. Az adatbázis-kezelő rendszerek inputjai:

Lekérdezések. A lekérdezés történhet:

- Egy általános lekérdező interfészen keresztül.
- Egy alkalmazói program interfészén keresztül.

Módosítások. Az adatbázisban levő adatok karbantartását (felvitel, módosítás, törlés) végző utasítások. Az utasítások kiadhatók egy általános interfészen, vagy egy alkalmazás interfészén keresztül.

Séma (struktúra) módosítások. Megfelelő jogosultsággal rendelkező felhasználók (pl. adatbázis-adminisztrátor) megváltoztathatják az adatbázis objektumok szerkezetét (definícióit).

A tárkezelő

Az adatbázis-kezelők általában közvetlenül felügyelik az adatok lemezen való tárolását. A tárkezelő két részből áll, a **pufferkezelőből** és a **fájlkezelőből.**

1. A fájlkezelő *nyilvántartja* a fájlok lemezen való elhelyezkedését és *beolvassa* egy állomány blokkjait a pufferkezelő kérésére. Az állományok lemezblokkokból (blocks, pages) épülnek fel, méretük általában 4K, 8K vagy 16K bájt.

2. A pufferkezelő a memóriát kezeli. A lemezről beolvasott blokkokat egy memóriaterületen (puffer) tárolja. A különböző műveletek innen kezelik az adatokat. Ha betelt a pufferterület, akkor a „nem használt" blokkokat visszaírja lemezre, az így felszabaduló memóriaterületre új blokkokat olvas. „Kérésre" a megadott blokkokat szintén lemezre írja.

A lekérdezés feldolgozó

A lekérdezés feldolgozó a magas szintű programnyelven megfogalmazott (pl. SQL nyelv) lekérdezéseket, adatbázis-műveleteket egyszerű utasítások sorozatává alakítja.

A lekérdezések a táblák és/vagy indexek soraira vonatkozó kérések. A lekérdezés feldolgozó legfőbb feladata a lekérdezések optimalizálása, egy „jó" *végrehajtási terv* kiválasztása. A végrehajtási terv olyan lépések sorozata, amely megadja a lekérdezés eredményét.

A tranzakció-kezelő

Az adatbázis-kezelők biztosítják, hogy egy vagy több lekérdezést, módosítást egy egységbe, tranzakcióba csoportosítsunk.

A tranzakció olyan műveletek csoportja, amelyeket egymás után egy egységként kell végrehajtani. Vagy az összes műveletet végrehajtjuk, vagy egyet sem.

A tranzakciók helyesen lefutásának biztosítása a **tranzakció-kezelő** feladata. Az alábbiakban ismertetjük az alapvető elvárásokat egy tranzakcióval szemben.

- **Atomosság.** A tranzakció vagy teljes egészében hajtódjon végre, vagy egyáltalán ne. Például az automatából történő pénzfelvétel és a hozzá kapcsolódó megterhelés az ügyfél számláján egyetlen atomi tranzakció.

- **Következetesség.** *Következetes állapotok*: az adatok megfelelnek bizonyos elvárásoknak. Például, egy repülőgép-helyfoglalási adatbázisban feltétel, hogy egyetlen ülőhelyet se rendeljünk hozzá két különböző utashoz. A feltételt megsérthetjük egy rövid időre egy tranzakció alatt (pl.: amíg az utasokat áthelyezzük az ülőhelyek között), a tranzakció-kezelő biztosítja, hogy a tranzakciók befejeződése után az adatbázis ismét következetes állapotba kerüljön.

- **Elkülönítés.** Az egyidejűleg futó tranzakciók egymásra hatását el kell különíteni egymástól, mintha egymás után futnának. Ha két ügyintéző ugyanarra a járatra ad el jegyet és a járaton már csak egy hely van, akkor az egyik kérést teljesíteni kell, a másikat el kell utasítani.

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

A tranzakció-kezelő zárolhatja a tranzakció által elérni kívánt adattételt. Amíg egy tranzakció zárolva tart egy tételt, addig a többi tranzakció (felhasználó) nem érheti el azt. Így a két tranzakció nem hat egymásra.

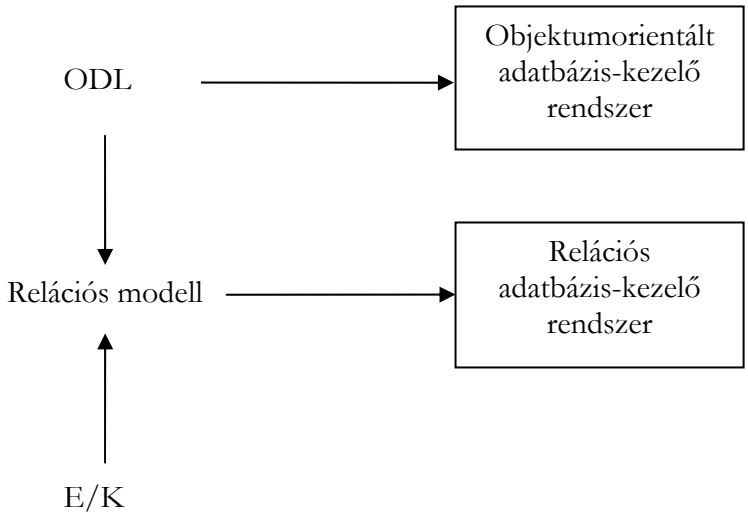
A tranzakció-kezelő a „lényeges” lépéseket (pl.: a tranzakció kezdete, az adatbázisban végzett módosítások, a tranzakció vége) feljegyzi egy naplófájlba. A naplófájl elkülönül az adatfájloktól, a naplóbejegyzéseket gyors írási algoritmussal írja lemezre a rendszer.

1.7. Ellenőrző kérdések

1. Definiálja a következő fogalmakat: egyed, tulajdonság, kapcsolat!
2. Adjon egy-egy példát a különböző kapcsolattípusokra!
3. Ismertesse a hierarchikus adatmodell lényeges jellemzőit!
4. Ismertesse a hálós adatmodell lényeges jellemzőit!
5. Ismertesse a relációs adatmodell lényeges jellemzőit!
6. Ismertesse az objektum-relációs adatmodell lényeges jellemzőit!
7. Soroljon fel néhány SQL terméket, fejlesztő céget!
8. Ismertesse röviden az SQL szabványosítási folyamatát!
9. Mik az elvárások egy adatbázis-kezelő rendszerrel szemben?
10. Sorolja fel az adatbázis-kezelő rendszerek főbb részeit!
11. Ismertesse a tárkezelő szerepét, működését!
12. Ismertesse a lekérdezés feldolgozó szerepét, működését!
13. Ismertesse a tranzakció-kezelő szerepét, működését!
14. Miért van szükség, zárolásra, naplózásra?

2. Adatmodellezés

Az adatbázis tervezés a rendelkezésre álló információk elemzésével, az információk komponensei közötti kapcsolatok meghatározásával kezdődik. Az adatbázis struktúráját *adatbázis sémának* nevezzük. A struktúra (séma) megadására különböző eszközök (nyelvek, modellek, jelölésrendszerek) állnak rendelkezésre. Tanulmányaink során a hagyományos **egyed-kapcsolat** (E/K) modellt, illetve az objektumorientált: **ODL** (**O**bject **D**efinition **L**anguage) tervezési sémát használjuk. A tervezés után következhet az adatbázis fizikai megvalósítása valamely *relációs* vagy *objektumorientált* adatbázis-kezelővel. A **2.1.** ábra ezt a folyamatot mutatja:



2.1. ábra. Az adatbázis-tervezés, megvalósítás folyamata

Relációs adatbázis-kezelő használata esetén a megtervezett modellt (ODL vagy E/K) még transzformálni kell relációs modellé.

2.1. Az objektumorientált modellezés

Az ODL segítségével az adatbázissémát az objektumorientált nyelveknek megfelelően (pl.: C++, Smalltalk) adhatjuk meg. Az objektumorientált

programozási (OOP) alapokat ismertnek feltételezzük, ezért csak vázlatosan ismertetjük a főbb fogalmakat.

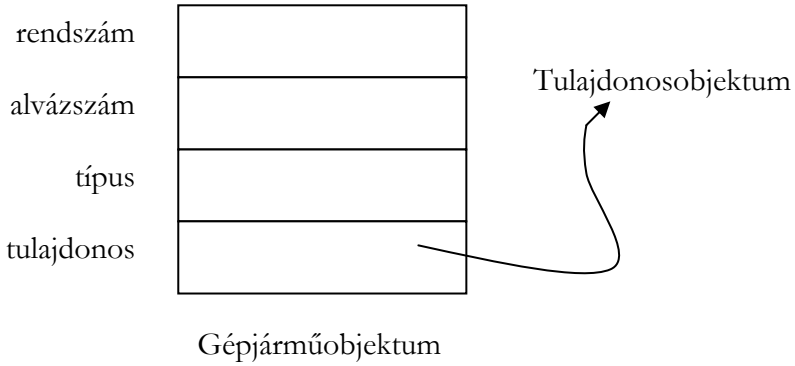
Objektum

A valós világ észlelhető egyedei, pl.: emberek, gépjárművek, filmek stb. Az objektumok rendelkeznek egyedi azonosítóval (OID), amely megkülönbözteti őket más objektumoktól.

Osztály

A hasonló tulajdonságú objektumok csoportja.

Tegyük fel, hogy a *gépjárműobjektumot* az alábbi néhány tulajdonsággal jellemzünk (**2.2.** ábra):



2.2. ábra. Egy gépjárművet reprezentáló objektum

A fenti tulajdonságok között nem soroltuk fel pl. a tulajdonos lakcímét. A lakcím már a *tulajdonosobjektumra* jellemző. A gépjárműobjektumok összessége adja a **gépjárműosztályt.** A két osztály (gépjármű és tulajdonos) között valós logikai kapcsolat áll fenn.

Egy osztály definiálásakor az alábbi jellemzőket adjuk meg:

Attribútumok

Az osztályra jellemző tulajdonságok és azok típusa (pl.: egész, szöveg stb.).

Kapcsolatok

Olyan osztályjellemzők, amelyek típusa egy másik osztály objektumára való hivatkozás, vagy hivatkozások halmaza.

Metódusok

Az osztály objektumaira alkalmazható függvények. A későbbiek során a metódusokkal nem foglalkozunk.

Alosztály

Osztályból származtathatunk alosztályokat. Az alosztály örökli az *ősosztály* (szuperosztály) jellemzőit (attribútumait, kapcsolatait), metódusait. Az alosztályhoz hozzáadhatunk új jellemzőket, metódusokat.

2.1.1. Típusok az ODL-ben

Az ODL típusrendszere alap adattípusokból és azokból származtatott összetett típusokból áll.

Alaptípusok

Atomi típusok

- Integer (egész szám)
- Float (lebegőpontos valós szám)
- Character (egy jel, karakter)
- String (jelsorozat)
- Boolean (logikai típus True, False értékkel)
- Enum (felsorolás: konkrét értékek, konstansok megadása)

Interfész típusok

Struktúrát reprezentáló típusok pl.: Gépjárműosztály. A struktúra komponensei az interfész attribútumai, illetve kapcsolatai. Az interfész típusok nevei *összetett típust* reprezentálnak.

Összetett típusok

Az alaptípusokból *típuskonstruáló* segítségével származtathatók:

- *Halmaz*
  A T típusú elemek véges halmaza: **Set**<T>.
- *Multihalmaz*
  A T típusú elemek véges multihalmaza (ugyanaz az elem többször is előfordulhat): **Bag**<T>.
- *Lista*
  A T típusú elemek véges listája (a sorrend lényeges): **List**<T>.
- *Tömb*
  A T típusú, i elemű tömb: **Array**<T, i>.
- *Struktúra*
  Legyenek $T_1, T_2, \ldots, T_n$ típusok és $F_1, F_2, \ldots, F_n$ mezőnevek! Az **n** mezőből álló, N nevű struktúrát az alábbi módon adjuk meg:

  Struct N $\{T_1\ F_1, T_2\ F_2, \ldots, T_n\ F_n\}$

Az első négy típust (halmaz, multihalmaz, lista, tömb) **kollekciótípusoknak** nevezzük.

2.1.2. Osztályok megadása az ODL-ben

Egy osztálydeklaráció megadásához az alábbi szintaktikai elemeket használjuk:

interface <név>
(*key* (kulcsattribútum1 [, kulcsattribútum2, …) |
(*keys* kulcsattribútum1 [, kulcsattribútum2, …])
{

<jellemzők listája>
};
<jellemzők listája>:= <attribútumok listája> és <kapcsolatok listája>
<attribútum>:= *attribute* <típus> <attribútum név>;
<kapcsolat>:=
relationship [Set<]<kapcsolódó osztály/interfész név>[>] <kapcsolatnév>
inverse <kapcsolódó osztály/interfész név>::<inverz kapcsolatnév>;

Szintaktika magyarázat

- Összetett kulcs esetén a kulcsot definiáló, attribútumokat zárójelbe tesszük.
- Több kulcs esetén használhatjuk a **keys** kulcsszót.
- Ha a kapcsolat típusú attribútum több értéket vehet fel, akkor a **Set** (halmaz típus) kulcsszóval jelöljük ezt.
- Egy attribútum típusa lehet atomi típus, vagy olyan struktúra, amely mezői atomi típusúak, illetve ezekre valamelyik kollekciótípust vagy struktúraképzést alkalmazva adódó összetett típus.
- Egy kapcsolat típusa vagy egy interfész típus, vagy egy kollekciótípus alkalmazva az interfész típusra.

Példa: Adjuk meg a **2.2. ábra Gépjármű** és **Tulajdonos** osztályok ODL deklarációját (egy egyszerűsített gépjármű nyilvántartási adatbázis (GépjárműNyilv) osztályai)!

Megoldás:

interface Gépjármű (keys *rendszám*, *alvázszám*) {
attribute string **rendszám**;
attribute string **alvázszám**;
attribute string **motorszám**;
attribute enum Fajta
{személygépkocsi, haszonjármű, busz, motorkerékpár, egyéb} **gjFajta**;

attribute string **gyártmány**;
attribute string **típus**;
attribute enum Szín {fehér, fekete, piros, …} **gjSzín**;
attribute integer **sajátTömeg**;
attribute integer **gyártásiÉv**;
attribute integer **hengerűrtartalom**;
relationship Set<Tulajdonos> **gépjárműTulajdonosai**
inverse Tulajdonos::**tulajdonosGépjárművei**; };
interface Tulajdonos (key *személyiIgSzám*) {
attribute string **személyiIgSzám**;
attribute string **név**;
attribute string **anyjaNeve**;
attribute string **szülHely**;
attribute struct Dátum
{integer év, integer hónap, integer nap} **szülDátum**;
attribute struct Cím
{string irányítószám, string település, string utca} **lakcím**;
relationship Set<Gépjármű> **tulajdonosGépjárművei**
inverse Gépjármű::**gépjárműTulajdonosai**;
};

Megjegyzések:

- A fenti attribútumokat megtalálhatjuk egy gépjármű forgalmi engedélyében.
- A *Szín* felsorolásban csak néhány színt adtunk meg, természetesen az összes előforduló színt fel kellene sorolni.
- Nem foglalkozunk az „üzembentartó” forgalmi engedély bejegyzéssel, feltételezzük, hogy a tulajdonos az üzembentartó is.

- Feltételezzük, hogy a tulajdonosok magánszemélyek, tehát értelmes pl. az „anyjaNeve” attribútum.
- Feltételezzük, hogy egy tulajdonosnak több gépjárműve lehet, illetve, hogy egy gépjármű több tulajdonoshoz tartozhat.
- A Gépjármű osztály kulcsa a *rendszám* és az *alvázszám* is. A továbbiakban elsődleges kulcsként a rendszám attribútumot használjuk.
- Az ODL-ben minden kapcsolatnál kötelező az inverz kapcsolat megadása is.
- A Dátum struktúra helyett használhatjuk a **Date** típust is.

2.1.3. Alosztályok az ODL-ben

Előfordulhat, hogy egy osztályban vannak olyan speciális tulajdonságok, amelyekkel nem rendelkezik az osztály minden objektuma. Ilyenkor az osztályt érdemes lehet alosztályokra bontani.

A példánkban szereplő *Gépjármű* osztály pl. a *gjFajta* tulajdonság alapján *Személyautó, Haszonjármű, Busz, Motorkerékpár, Egyéb* alosztályokra bontható. Az alosztályokhoz a fajtára jellemző attribútumokat adhatunk, míg a „közös” attribútumokat, kapcsolatokat a *Gépjármű* osztály tartalmazza.

A deklarációban az alosztály neve után kettősponttal elválasztva adjuk meg a szuperosztály nevét pl.:

interface Személyautó:Gépjármű {…};

Egy osztálynak több alosztálya is lehet, illetve egy alosztályhoz több szuperosztály is tartozhat. A származtatási láncolatot *osztályhierarchiának* nevezzük. Az osztály deklarációban a szuperosztályokat (az adott osztály őseit) vesszővel választjuk el egymástól. Az osztály az összes ősének attribútumait, kapcsolatait örökli.

2.1.4. Feladatok

1. Példa: A **2.3. ábra** egy számlatömb számláját mutatja. Készítsük el egy számlákat nyilvántartó (bejövő/kimenő számlák), **Számlák** nevű, adatbázis ODL modelljét!

Megoldás: A számlán levő információk tárolására tervezett egyszerűsített számítógépes nyilvántartás elsősorban az ÁFA-bevallással kapcsolatos elvárásoknak szeretne megfelelni. Tartalmazza a bejövő (szállító), és kimenő (vevő) számlák információit is. Adott időszakban (hónap, negyedév, év) befizetendő ÁFA: vevői számlák ÁFÁ-ja – szállítói számlák ÁFÁ-ja, ha ez pozitív érték. Ha negatív, akkor beszámítható a következő időszak elszámolásában, vagy speciális esetben visszaigényelhető.

A **2.1. táblázat** összefoglalja a számlával kapcsolatos adatokat, információkat, röviden utal az egyes adatelemek (tulajdonságtípusok) jellemzőire.

Megjegyzések:

Szállító – Vevő osztályok helyett **Partner** osztályt terveztünk, mert a számlán mindig csak egy partner jelenik meg, a másik a nyilvántartást végző cég. Egyedi azonosító: Partnerkód (PKod).

A *Számla* osztály helyett (redundancia elkerülése) **SzlaFej** (egyedi azonosító: Számlaazonosító (SZAzon)) és **SzlaTétel** (nincs egyedi azonosító) osztályokat terveztünk.

A bejövő/kimenő számlák megkülönböztetésére bevezettük az SZTipus attribútumot.

Az ÁFA-százalék numerikus értékei: 20, 15, 5, 0 (Mentes).

SZÁMLA Sorszám: **K** 0193516

A számlakibocsátó neve, címe, aláírása: Adóazonosító szám:		A vevő neve, címe:	
A fizetés módja:	A teljesítés időpontja:	A számla kelte:	A fizetés határideje:

A termék vagy szolgáltatás

a termék (szolgáltatás), megnevezése, besorolási száma, egyéb jellemzői	menny. egys.	mennyisége	egységára (ÁFA nélk.)	értéke (ÁFA nélkül)	ÁFA kulcsa	áthárított ÁFA összege	értéke (ÁFA-val együtt)

A számlában szereplő ÁFA részletezése	ÁFA-kulcsok %	%	%	AM	Összesen
Számlaérték ÁFA nélkül					
EREDETI 1 PÉLDÁNY — Az ÁFA összege					
EREDETI 1 PÉLDÁNY — A számla végösszege					

aláírás P. H.

PM által ajánlott

B. 13–374/V/új r. sz. - (3 példányos számla) - 8655 (Dfsz.: 5427)
Gyártja és forgalmazza: Pátria Nyomda Rt. 1117 Bp., Hunyadi János út 7. Tel.: 463-0440, Fax: 463-0450

2.3. ábra. Számlaformátum

Adatok, információk	**Leírás**	**Azonosító**	**Adattípus**
Számla sorszám		SZSzam	szöveg
A számlakibocsátó: neve, címe, adóazonosító száma	Cím: irányítószám, település, utca, házszám.		
A vevő: neve, címe		PNev PIrsz PTelep PUtca	szöveg szöveg szöveg szöveg
A fizetés módja	készpénz, átutalás, csekk, inkasszó.	SZFizMod	felsorolás
A teljesítés időpontja	ÁFA bevalláshoz ez kell!	SZTeljDat	dátum
A számla kelte	Nem tároljuk.		
A fizetés határideje	Nem tároljuk.		
A termék (szolgáltatás), megnevezése, egyéb jellemzői	Termék: Vámtarifa, szolgáltatás: SZJ.	TMegn TVamt_SZJ	szöveg szöveg
Mennyiségi egység	db, kg, m, l, cm, doboz, raklap.	TMe	felsorolás
Mennyiség	2 tizedes.	TMenny	valós
Egységára (ÁFA nélkül)	2 tizedes.	TEar	valós
Értéke (ÁFA nélkül)	Számítható!		
ÁFA kulcsa	20%, 15%, 5%, Mentes	TSzazal	egész
Áthárított ÁFA összege	Számítható, egészre kerekített!		
Értéke (ÁFÁ-val együtt)	Számítható!		

2.1. táblázat.

A Számlák adatbázis ODL modellje:

```
interface Partner (key PKod)  {
attribute string PKod;
attribute string PNev;
attribute struct Cím {string PIrsz, string PTelep, string PUtca} PCim;
Relationship Set <SzlaFej> partnerSzámlái
                    Inverse SzlaFej::számlaPartner;
};

interface SzlaFej (key SZAzon)  {
attribute integer SZAzon;
attribute string SZSzam;
attribute struct Dátum
          {integer SZEv, integer SZHo, integer SZNap} SZTeljDat;
attribute enum FizMod {átutalás, készpénz, csekk, inkasszó} SZFizMod;
attribute enum Típus {szállító, vevő} SZTipus;
Relationship Partner számlaPartner
                    Inverse Partner::partnerSzámlái;
Relationship Set <SzlaTétel> számlaTételei
                    Inverse SzlaTétel::számlaFej;
};

interface SzlaTétel  {
attribute string TMegn;
attribute string TVamt_SZJ;
attribute enum Me {db, m, cm, kg, doboz, l, raklap} TMe;
attribute float TMenny;
```

attribute float **TEar**;

attribute integer **TSzazal**;

Relationship SzlaFej **számlaFej**

inverse Szlafej::**számlaTételei**;

};

Az ODL-séma felírását segítheti a kapcsolatok táblázatos felírása (**2.2. táblázat**).

Osztály	**Kapcsolat**	**Inverz kapcsolat**	**Kapcsolódó osztály**
Partner	partnerSzámlái	számlaPartner	SzlaFej
SzlaFej	számlaTételei	számlaFej	SzlaTétel

2.2. táblázat.

2. Példa: Egy Folyóirat-előfizetési nyilvántartási számítógépes rendszerben a **2.3. táblázat** adatait kezeljük. Készítsük el a **Folyóiratok** adatbázis ODL sémáját (kulcsokat is)!

Adatok, információk	**Leírás**
A folyóirat típusa	napi, családi (képes), tudományos, egyéb.
A folyóirat címe	
A folyóirat kiadó megnevezése	
A folyóirat kiadó címe	Irányítószám, település, utca, házszám.
Havi előfizetési díj Ft-ban	Legalább 100Ft, 100000Ft-nál kisebb.
Előfizető név, megnevezés	Kötelező.
Előfizető címe	Irányítószám, település, utca, házszám.
Előfizetés kezdő dátuma	Tetszőleges hónap első napja.
Előfizetés hónapok száma	1/4 évre, 1/2 évre vagy 1 évre lehet előfizetni.
Előfizetett mennyiség	Nagyobb nullánál, 1000-nél kisebb.

2.3. táblázat.

Szabályok:

- Több előfizetése is lehet egy előfizetőnek.
- Egy folyóiratból több példány fizethető elő.
- Egy kiadó több folyóiratot ad ki.

Megoldás: A **2.4. táblázat** összefoglalja a folyóirat előfizetéssel kapcsolatos adatokat, információkat, röviden utal az egyes adatelemek (tulajdonságtípusok) jellemzőire. A „használható" osztály-kulcsok miatt 3 azonosító attribútumot (fazonosító, kazonosító, eazonosító) vezettünk be.

Adatok, információk	**Leírás**	**Azonosító**	**Adattípus**
Folyóirat azonosító kód	Egyedi sorszám	fazonosító	szöveg
A folyóirat típusa	napi, családi (képes), tudományos, egyéb.	ftípus	felsorolás
A folyóirat címe	Kötelező.	fcím	szöveg
A folyóirat kiadó azonosító	Sorszám, egyedi azonosító.	kazonosító	szöveg
A folyóirat kiadó megnevezése	Kötelező.	knév	szöveg
A folyóirat kiadó címe	Irányítószám, település, utca, házszám.	kcím	struktúra
Havi előfizetési díj Ft-ban	Legalább 100Ft, 100000Ft-nál kisebb.	fhavidíj	egész
Az előfizető azonosító kódja	Egyedi sorszám, kisebb 100000-nél.	eazonosító	szöveg
Előfizető név, megnevezés	Kötelező.	enév	szöveg
Előfizető címe	Irányítószám, település, utca, házszám.	ecím	struktúra
Előfizetés kezdő dátuma	Tetszőleges hónap első napja.	efkezdet	dátum
Előfizetés hónapok száma	3, 6, 9 hónap	efhónap	felsorolás vagy egész
Előfizetett mennyiség	Nagyobb nullánál, 1000-nél kisebb.	efmenny	egész

2.4. táblázat.

A Folyóiratok adatbázis ODL modellje:

interface Kiadó (key *kazonosító*) {

attribute string **kazonosító**;

attribute string **knév**;

attribute struct cím {string **irányítószám**, string **település**, string **utca**} **kcím**;

Relationship Set <Folyóirat> **kiadóFolyóiratai**

Inverse Folyóirat::**folyóiratKiadója**; };

interface Folyóirat (key *fazonosító*) {

attribute string **fazonosító**;

attribute enum típus {napi, családi, tudományos, egyéb} **ftípus**;

attribute string **fcím**;

attribute integer **fhavidíj**;

relationship Kiadó **folyóiratKiadója**

inverse Kiadó::**kiadóFolyóiratai**;

rlationship Set <Előfizetés> **folyóiratElőfizetetései**

iverse Előfizetés::**előfizetésFolyóirata**;};

interface Előfizető (key *eazonosító*) {

attribute string **eazonosító**;

attribute string **enév**;

attribute struct cím {string **irányítószám**, string **település**, string **utca**} **ecím**;

relationship Set <Előfizetés> **előfizetőElőfizetései**

inverse Előfizetés::**előfizetésElőfizetője**; };

interface Előfizetés {
attribute date **efkezdet**;
attribute integer **efhónap**;
attribute integer **efmenny**;
relationship Előfizető **előfizetésElőfizetője**
inverse Előfizető::**előfizetőElőfizetései**;
relationship Folyóirat **előfizetésFolyóirata**
inverse Folyóirat::**folyóiratElőfizetései**; };

Az ODL-séma felírását segítheti a kapcsolatok táblázatos felírása (**2.5. táblázat**).

Osztály	**Kapcsolat**	**Inverz kapcsolat**	**Kapcsolódó osztály**
Kiadó	kiadóFolyóiratai	folyóiratKiadója	Folyóirat
Előfizető	előfizetőElőfizetései	előfizetésElőfizetője	Előfizetés
Folyóirat	folyóiratElőfizetései	előfizetésFolyóirata	Előfizetés

2.5. táblázat.

Példa: Egy Ügynöki-biztosítások nyilvántartási számítógépes rendszerben a **2.6. táblázat** adatait kezeljük. Készítsük el a **Biztosítások** adatbázis ODL sémáját (kulcsokat is)!

Adatok, információk	**Leírás**
Ügynöknév	Biztosítási termékeket jutalékos rendszerben értékesítő személy.
Ügynök születési dátuma	
Ügynök lakcím	Irányítószám, település, utca, házszám.
Ügynök jutalék (ezrelék)	A biztosítási összeg adatott ezreléke az ügynök jutaléka.
Biztosított személy (ügyfél) neve	

2.6. táblázat.

Táblázat folytatás

Adatok, információk	Leírás
Ügyfél születési dátuma	
Ügyfél lakcím	Irányítószám, település, utca, házszám.
Ügyfél telefon	Csak egy telefonszámot tartunk nyilván.
Biztosítás kötvényszám	Egyedi azonosító.
Biztosítás módozat	A biztosítást végző cégek termékeinek neve. Pl.: a Signal Biztosító RT. esetén: ”S0200”, ”S0205”, ”S0501” stb.
Biztosítás kötési dátuma	
Biztosítási összeg	100.000-nél nagyobb, 10.000.000-nál kisebb.

Szabályok:

- Egy ügynök több biztosítást köthet.
- A kötött biztosítás csak egy ügynökhöz, egy személyhez tartozhat.
- Egy személynek több biztosítása is lehet.

Megoldás: A **2.7. táblázat** összefoglalja a biztosításkötésekkel kapcsolatos adatokat, információkat, röviden utal az egyes adatelemek (tulajdonságtípusok) jellemzőire. A „használható” osztály-kulcsok miatt 2 azonosító attribútumot (ügynökAzon, ügyfélAzon) vezettünk be.

Adatok, információk	Leírás	Azonosító	Adattípus
Ügynökazonosító		ügynökAzon	szöveg
Ügynöknév		ügynökNév	szöveg
Ügynök születési dátuma		ügynökSzülDátum	dátum
Ügynök lakcím	Irányítószám, település, utca, házszám.	ügynökLakcím	szöveg
Ügynök telefonszám		ügynökTelefon	szöveg

2.7. táblázat.

Táblázat folytatás

Adatok, információk	**Leírás**	**Azonosító**	**Adattípus**
Ügynök jutalék (ezrelék)		ügynökJutalék	egész
Biztosított személy (ügyfél) azonosító		ügyfélAzon	szöveg
Ügyfél neve		ügyfélNév	szöveg
Ügyfél születési dátuma		ügyfélSzülDátum	dátum
Ügyfél lakcím	Irányítószám, település, utca, házszám.	ügyfélLakcím	szöveg
Ügyfél telefon		ügyfélTelefon	szöveg
Biztosítás kötvényszám		kötvényszám	szöveg
Biztosítás módozat		módozat	felsorolás
Biztosítás kötési dátuma		kötésiDátum	dátum
Biztosítási összeg (Ft)		összeg	egész

A Biztosítások adatbázis ODL modellje:

interface Ügyfél (key *ügyfélAzon*) {

attribute string **ügyfélAzon**

attribute string **ügyfélNév**;

attribute struct cím {string **irányítószám**, string **település**, string **utca**} **ügyfélLakcím**;

attribute string **ügyfélTelefon**;

attribute date **ügyfélSzülDátum**;

relationship Set <Biztosítás> **ügyfélBiztosítások**

inverse Biztosítás::**biztosításÜgyfele**;

relationship Ügynök **ügyfélÜgynöke**
inverse Ügynök::**ügynökÜgyfelei**;
};
interface Ügynök (key *ügynökAzon*) {
attribute string **ügynökAzon**
attribute string **ügynökNév**;
attribute struct cím {string **irányítószám**, string **település**, string **utca**} **ügynökLakcím**;
attribute string **ügynökTelefon**;
attribute date **ügynökSzülDátum**;
attribute integer **ügynökJutalék**;
relationship Set <Biztosítás> **ügynökBiztosítások**
inverse Biztosítás::**biztosításÜgynöke**;
relationship Set<Ügyfél> **ügynökÜgyfelei**
inverse Ügyfél::**ügyfélÜgynöke**;
};
interface Biztosítás (key *kötvényszám*) {
attribute string **kötvényszám**;
attribute string **módozat**;
attribute date **kötésiDátum**;
attribute integer **összeg**;
relationship Ügynök **biztosításÜgynöke**
inverse Ügynök::**ügynökBiztosítások**;
relationship Ügyfél **biztosításÜgyfele**
inverse Ügyfél::**ügyfélBiztosítások**;
};

Az ODL-séma felírását segítheti a kapcsolatok táblázatos felírása (**2.8. táblázat**).

Osztály	Kapcsolat	Inverz kapcsolat	Kapcsolódó osztály
Ügyfél	ügyfélBiztosítások	biztosításÜgyfele	Biztosítás
Ügynök	ügynökBiztosítások	biztosításÜgynöke	Biztosítás
Ügyfél	ügyfélÜgynöke	ügynökÜgyfelei	Ügynök

2.8. táblázat.

2.2. Egyed-kapcsolat (E/K) diagramok

2.2.1. Egyed-kapcsolat alapfogalmak

Az egyed-kapcsolat modell grafikus formában mutatja az adatbázis szerkezetét. A három alapelem:

- **Egyedhalmazok**

Az egyedhalmazok elemei az egyedek.

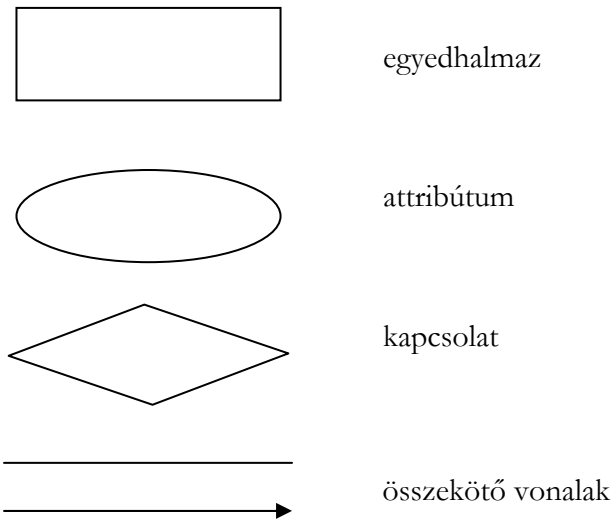
- **Attribútumok**

Az attribútumok értékei az egyed tulajdonságait írják le.

- **Kapcsolatok**

Két vagy több egyedhalmazt kapcsolnak össze. A kapcsolatok kétirányúak.

A **2.4.** ábra az E/K modellben használható alap grafikus elemeket mutatja:



2.4. ábra. Az E/K modell grafikus elemei

Példa: A **Rendelések** nyilvántartásban a **2.9. táblázat**ban összefoglalt információkat kezeljük. Készítsük el a Rendelések E/K modelljét!

Adatok, információk	**Leírás**
Termékazonosító kód	Termékazonosító kód.
Termék megnevezés	Termék megnevezés.
Termék egységár (Ft)	Aktuális egységár (Ft).
A vevő azonosító kódja	Egyedi azonosító.
A vevő neve, megnevezése	Kötelező.
Kapcsolattartó a vevő részéről	A kapcsolattartó személy neve.
Vevő: Utca, házszám	Nem kötelező.
Vevő: Település	Kötelező.
Vevő: Irányítószám	Nem kötelező.
Vevő: Telefonszám	Nem kötelező.
A rendelés azonosító kódja	Egyedi azonosító.
A megrendelés dátuma	Alapértelmezett érték: rendszerdátum.

Táblázat folytatás

Adatok, információk	Leírás
Szállítás dátuma	Ha megtörtént a szállítás, akkor: Szállítás dátuma >= Megrendelés dátuma.
Fuvarozási díj	Nagyobb nullánál.
Rendelt mennyiség	Nagyobb nullánál.
Rendeléskor a termék egységára (Ft)	Nagyobb nullánál.

2.9. táblázat.

Rendelési szabályok:

- A vevő egy megrendelésen több terméket is rendelhet.
- Egy rendelés csak egy vevőt érinthet.
- A megrendelt termékeket együtt kiszállítja ki a cég a vevőhöz.

Megoldás: A diagram elkészítése előtt táblázatot készítünk, amely a fenti információkat (attribútumokat) már a „megfelelő” egyedhalmazokhoz köti:

Egyedhalmaz	Attribútum		Tulajdonság típusa
	név	**szöveges értelmezés**	
Termékek	tkod	azonosító kód	azonosító
	tnev	megnevezés	leíró
	tear	egységár (Ft)	leíró
Vevők	vkod	azonosító kód	azonosító
	vnev	név, megnevezés	leíró
	vkapcs	kapcsolattartó	leíró
	vutca	utca, házszám	leíró
	vtelep	település	leíró
	virsz	irányítószám	leíró
	vtel	telefonszám	leíró

Táblázat folytatás

Egyedhalmaz	Attribútum		Tulajdonság típusa
	név	**szöveges értelmezés**	
RendelésFejek	rkod	azonosító kód	azonosító
	rdat	megrendelés dátuma	leíró
	rszdat	szállítás dátuma	leíró
	rfdij	fuvarozási díj	leíró
RendelésTételek	tazon	azonosító kód	azonosító
	rmenny	rendelt mennyiség	leíró
	rear	rendelési egységár	leíró

2.10. táblázat.

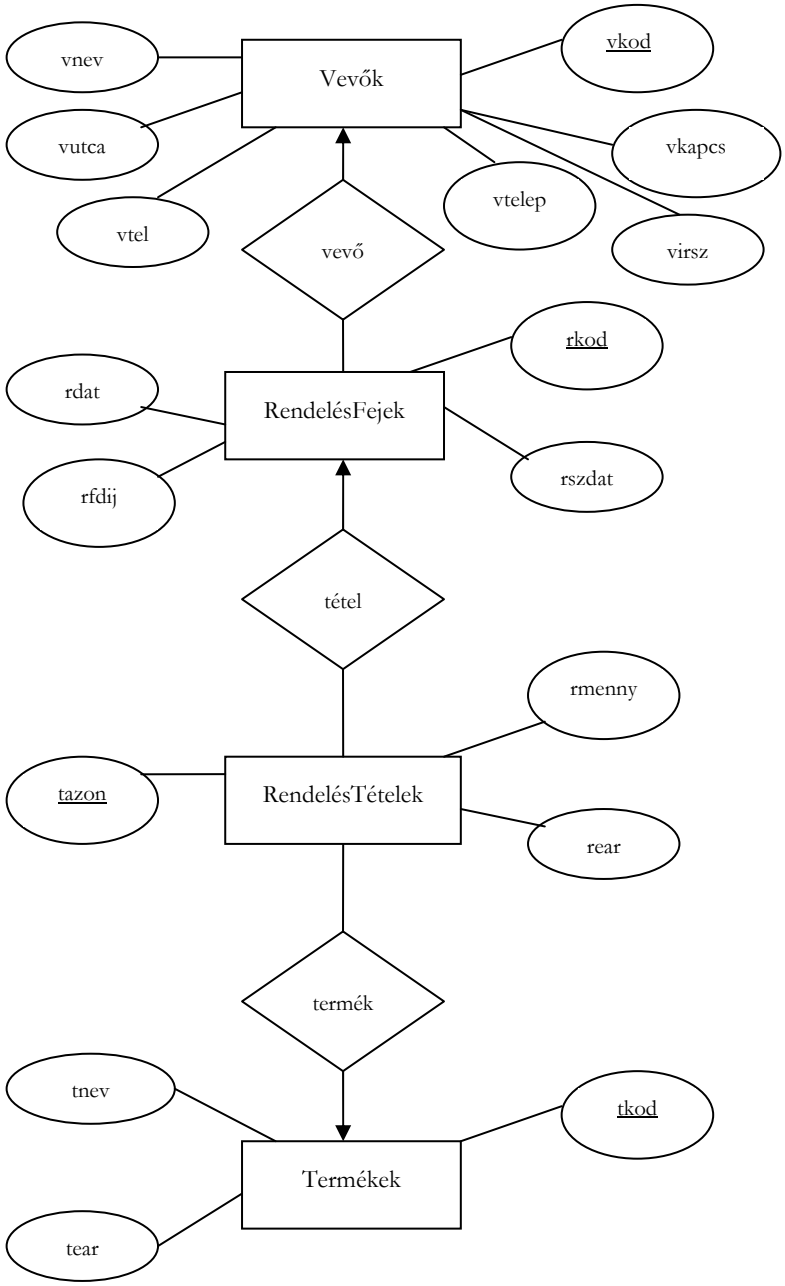
A 2.2. táblázat a Rendelések egyedhalmaz helyett RendelésFejek és RendelésTételek egyedhalmazt használ. Így „egyszerűbb” lesz a transzformálás relációs modellbe (nem lesz redundancia: felesleges adattárolás). Minden egyedhalmazban megadtunk azonosító (kulcs) attribútumot. Ezek lehetnek egyszerű sorszámok, amit majd az adatbázis-kezelő rendszer generál. Az E/K diagramban a kulcs mezőt aláhúzással jelöljük.

A rendszerben az alábbi egy-több típusú kapcsolatok jelennek meg:

Egyedhalmazok	Kapcsolatnév
Vevők-RendelésFejek	vevő
RendelésFejek-RendelésTételek	tétel
Termékek-RendelésTételek	termék

2.11. táblázat.

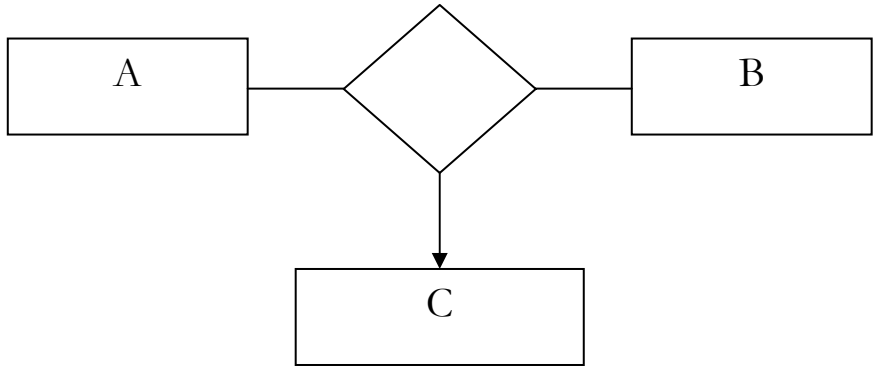
A kapcsolat „egy” oldalát jelzi a nyíl. Pl.: egy vevőhöz több rendelés tartozhat. Az E/K diagram a **2.5. ábra** látható:



2.5. ábra. Rendelések E/K diagram

2.2.2. Sokágú kapcsolatok

Előfordulhatnak olyan kapcsolatok, amikor a kapcsolatban kettőnél több egyedhalmaz vesz részt. Ilyenkor a rombuszból annyi vonal indul ki, amennyi egyedhalmaz részt vesz a kapcsolatban. A **2.6** ábra 3-ágú kapcsolatot mutat az A, B és C egyedhalmazok között:



2.6. ábra. Többágú E/K kapcsolat

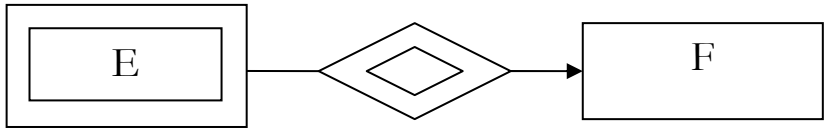
Ha tehetjük, akkor kerüljük a többágú kapcsolatokat, mert relációs modellbe való transzformáció előtt általában az ilyen kapcsolatokat először binárissá (2-ágúvá) kell átalakítani.

A többágú kapcsolatokban a *nyíl* szerepe megváltozik. A fenti ábra úgy értelmezhető, hogy egy konkrét A-beli és egy konkrét B-beli egyed előforduláshoz csak egy C-beli egyed előfordulás tartozik.

2.2.3. Gyenge egyedhalmazok

Ha egy egyedhalmaz kulcsában levő attribútumok között van más egyedhalmazbeli attribútum (akár az összes), akkor az ilyen egyedhalmazt **gyenge egyedhalmaznak** nevezzük. Erre azért van szükség, mert a saját attribútumok nem adnak kulcsot.

A gyenge egyedhalmazt dupla kerettel jelöljük. A gyenge egyedhalmaz több-egy kapcsolatát is dupla kerettel jelöljük. A gyenge egyedhalmaz ezen a kapcsolaton keresztül egészíti ki a kulcsát a kapcsolódó egyedhalmaz kulcs attribútumaiból.



2.7. ábra. Gyenge egyedhalmaz

Az E gyenge egyedhalmaz kulcs attribútumai között van F-beli kulcs attribútum is.

2.2.4. Diagram specialitások

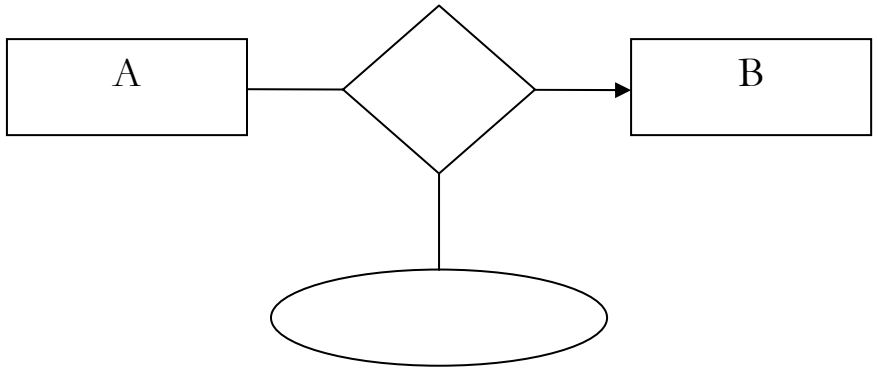
A továbbiakban röviden megadjuk az E/K diagramban használható egyéb tervezési elemeket:

Szerepek

Ha egy kapcsolatban ugyanaz az egyedhalmaz 2-szer, vagy többször is szerepel, akkor egy-egy vonal a kapcsolat adott szerepe.

Kapcsolatok attribútumai

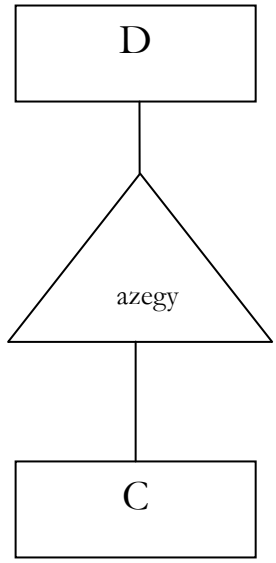
Kényelmi szempontok miatt előfordul, hogy egy kapcsolathoz rendelünk tulajdonságot, vagy tulajdonságokat (**2.8.** ábra).



2.8. ábra. Kapcsolat attribútummal

2.2.5. Alosztályok, öröklődési kapcsolat

Az ODL-nek megfelelően a szuperosztály (D) és az alosztály (C) is egy-egy egyedhalmaz. A két téglalapot háromszöggel kapcsoljuk össze, amelybe az „azegy” (angol megfelelő: **is a**) szócskát írjuk. A háromszög csúcsa mutat a szuperosztályra:



2.9. ábra. Alosztály kapcsolat

A közös attribútumokat a D egyedhalmazhoz, az alosztály eltérő attribútumait a C egyedhalmazhoz rajzoljuk.

2.2.6. Feladatok

1. Példa: Készítsük el a 2.1.4 fejezet 1. Példa adatbázis-tervéhez tartozó E/K diagramot!

Megoldás: A diagram elkészítése előtt táblázatot készítünk, amely a megadott információkat (attribútumokat) már a „megfelelő” egyedhalmazokhoz köti:

Egyedhalmaz	Attribútum		Tulajdonság típusa
	név	**szöveges értelmezés**	
Partnerek	**PKod**	Partnerkód	azonosító
	PNev	Partner név/megnevezés	leíró
	PIrsz	Cím - irányítószám	leíró
	PTelep	Cím - település	leíró
	PUtca	Cím – Utca, házszám	leíró
SzlaFejek	**SZAzon**	Számlaazonosító	azonosító
	SZSzam	Számla sorszám	leíró
	SZFizMod	Fizetési mód	leíró
	SZTeljDat	Teljesítés dátuma	leíró
	SZTipus	Számla típusa (szállítói, vevői)	leíró
SzlaTételek	**TSorsz**	Tétel sorszám	leíró
	TMegn	Megnevezés	leíró
	TVamt_SZJ	Termék-, szolgáltatás kód	leíró
	TMe	Mértékegység	leíró
	TMenny	Mennyiség	leíró
	TEar	Nettó egységár (Ft)	leíró
	TSzazal	ÁFA-százalék	leíró

2.12. táblázat.

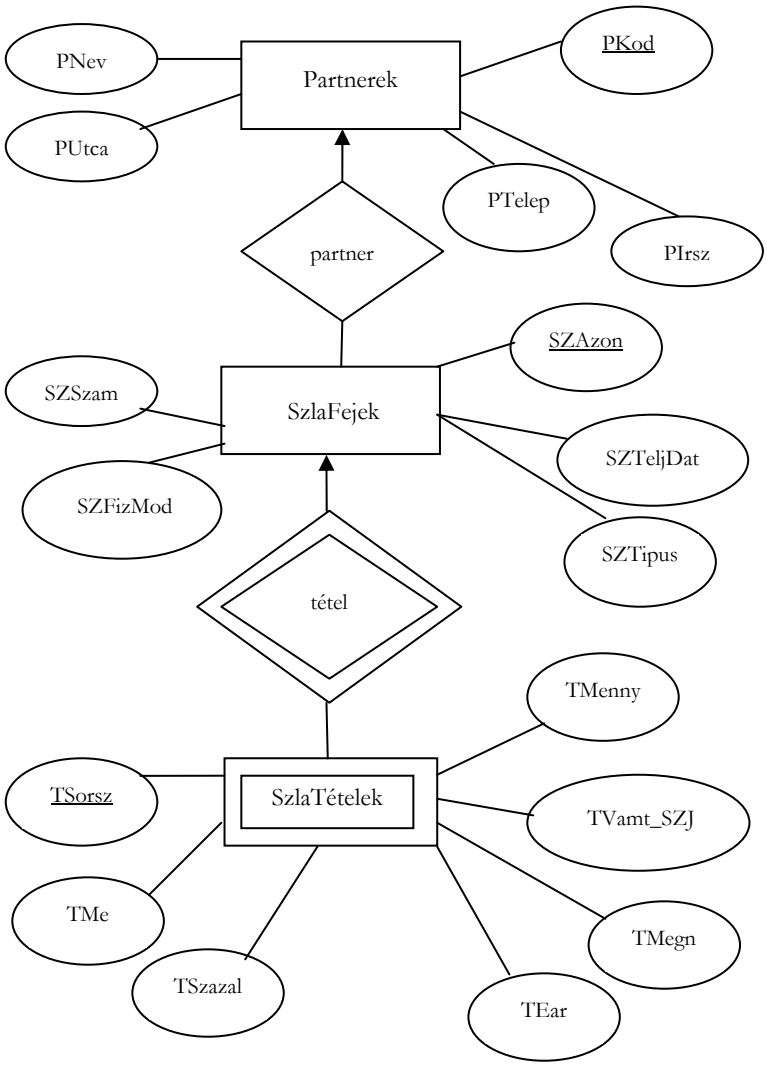
Az SzlaTételek egyedhalmazban bevezettük a TSorsz attribútumot, amellyel összetett kulcsot (SZAzon, TSorsz) definiálunk a gyenge egyedhalmazban.

A rendszerben az alábbi **egy-több típusú** kapcsolatok jelennek meg:

Egyedhalmazok	Kapcsolatnév
Partnerek - SzlaFejek	partner
SzlaFejek - SzlaTételek	tétel

2.13. táblázat.

Az adatbázis-terv E/K diagramja:



2.10. ábra. A Számlák adatbázis E/K diagramja

2. Példa: Készítsük el az alábbiakban részletezett Bolt adatbázis E/K diagramját!

Egy **Bolt** (butik) az alábbi információkat kezeli egy számítógépes nyilvántartási rendszerben:

Adatok, információk
Áru cikkszám, megnevezés, mértékegység, eladási ár, készlet, minimális készlet, maximális készlet.
Szállítói törzsadatok: azonosító, név, cím, kapcsolattartó, telefon, megjegyzés.
Rendelés sorszám, dátum, kitől rendeltünk.
A **megrendelt áru** cikkszáma, mennyisége, egységára, megtörtént-e a szállítás.

Rendelési szabályok:

- Egy rendelés csak egy szállítót érinthet.
- A megrendelt áruk szállítása külön-külön is történhet.
- Akkor kell rendelni, ha a készlet lecsökken a minimális készletre.
- Rendeléssel a feltöltés a maximális készletig történhet.
- Rendelésenként a cikkszám egyedi.

Megoldás: A diagram elkészítése előtt táblázatot készítünk, amely a megadott információkat (attribútumokat) már a „megfelelő” egyedhalmazokhoz köti:

Egyedhalmaz	Attribútum		Tulajdonság típusa
	név	**szöveges értelmezés**	
Áruk	CIKKSZAM	Cikkszám	**azonosító**
	CIKKNEV	Cikk megnevezés	leíró
	KESZLET	Aktuális készlet	leíró
	MEGYS	Mértékegység	leíró
	ELADAR	Eladási egységár	leíró
	MINKESZ	Minimális készlet	leíró
	MAXKESZ	Maximális készlet	leíró
Szállítók	SZAZON	Szállítóazonosító	**azonosító**
	SZNEV	Szállító név/megnevezés	leíró
	SZCIM	Cím: irányítószám, település, utca, házszám	leíró
	SZKAPCS	Kapcsolattartó személy	leíró
	SZKTEL	Telefon	leíró
	SZMEGJ	Megjegyzés a szállítóról	leíró
RendelésFejek	RENDAZON	Rendelésazonosító	**azonosító**
	RENDDAT	Rendelés dátuma	leíró
RendelésTételek	RMENNY	Megrendelt mennyiség	leíró
	REAR	Rendelési egységár	leíró
	RSZALLJEL	Szállításjelző	leíró

2.14. táblázat.

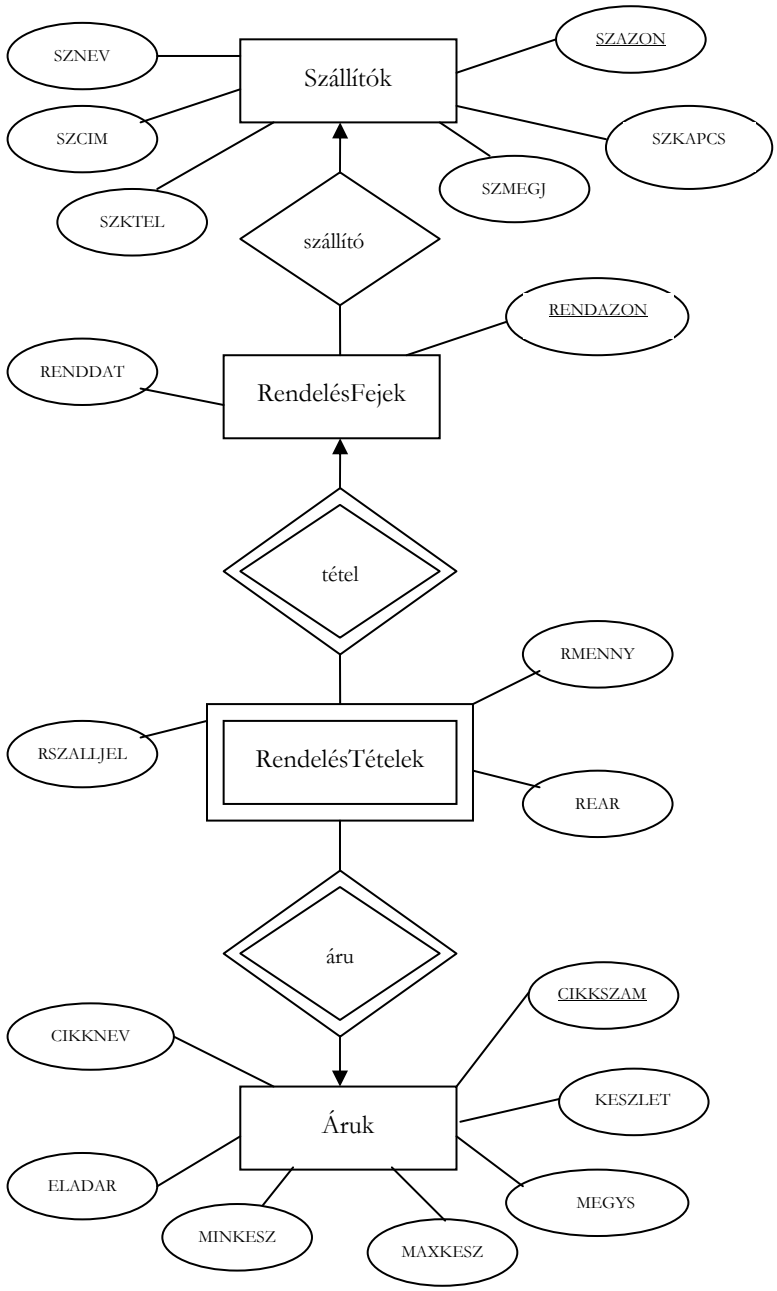
A RendelésTételek egyedhalmazban nincs kulcs attribútum, ezért gyenge egyedhalmaz. Összetett kulcsát (RENDAZON, CIKKSZAM) más egyedhalmazoktól kapja a kapcsolatokon keresztül.

A rendszerben az alábbi **egy-több típusú** kapcsolatok jelennek meg:

Egyedhalmazok	Kapcsolatnév
Szállítók - RendelésFejek	szállító
RendelésFejek - RendelésTételek	tétel
Áruk - RendelésTételek	áru

2.15. táblázat.

Az adatbázisterv E/K diagramja:



2.11. ábra. A Bolt adatbázis E/K diagramja

3. Példa: Készítsük el az alábbiakban részletezett Panzió adatbázis E/K diagramját!

Egy **Panzió-üzemeltető** az alábbi információkat kezeli egy számítógépes nyilvántartási rendszerben:

Adatok, információk
A panzió **szobáinak** törzsadatai: szobaszám, ágyak száma, szobaár.
A vendégek **nyilvántartandó** adatai: név, személyi igazolvány szám vagy útlevélszám, lakcím, telefon.
A panzióvendégek érkezésének dátuma, az eltöltött éjszakák száma.

Üzemeltetési szabályok:

- Minden vendég nyilvántartási adatait felvisszük a rendszerbe.
- Egy vendég többször is megszállhat a panzióban.
- A számlát személyenként állítjuk ki.
- A rendszer meg tudja adni a szabad szobákat.

Megoldás: A diagram elkészítése előtt táblázatot készítünk, amely a megadott információkat (attribútumokat) már a „megfelelő” egyedhalmazokhoz köti.

Egyedhalmaz	Attribútum		Tulajdonság típusa
	név	**szöveges értelmezés**	
Vendégek	vazon	Vendégazonosító (sorszám)	**azonosító**
	vnev	Név	leíró
	vszigsz	Személyi igazolvány száma vagy útlevél száma	leíró
	virsz	Lakcím-irányítószám	leíró
	vtelep	Lakcím-település	leíró
	vutca	Lakcím-utca, házszám	leíró
	vtel	Telefon	leíró
Szobák	szazon	Szobaazonosító (szobaszám)	**azonosító**
	szagy	Ágyak száma	leíró
	sznapidij	Szobaár (Ft) egy éjszakára egy főnek	leíró
	szfoglj	Foglaltság jelző	leíró
Vendégforgalom	vfazon	Forgalomazonosító (sorszám)	**azonosító**
	erkdat	Érkezés dátuma	leíró
	vejszaka	Vendégéjszakák száma	leíró

2.16. táblázat.

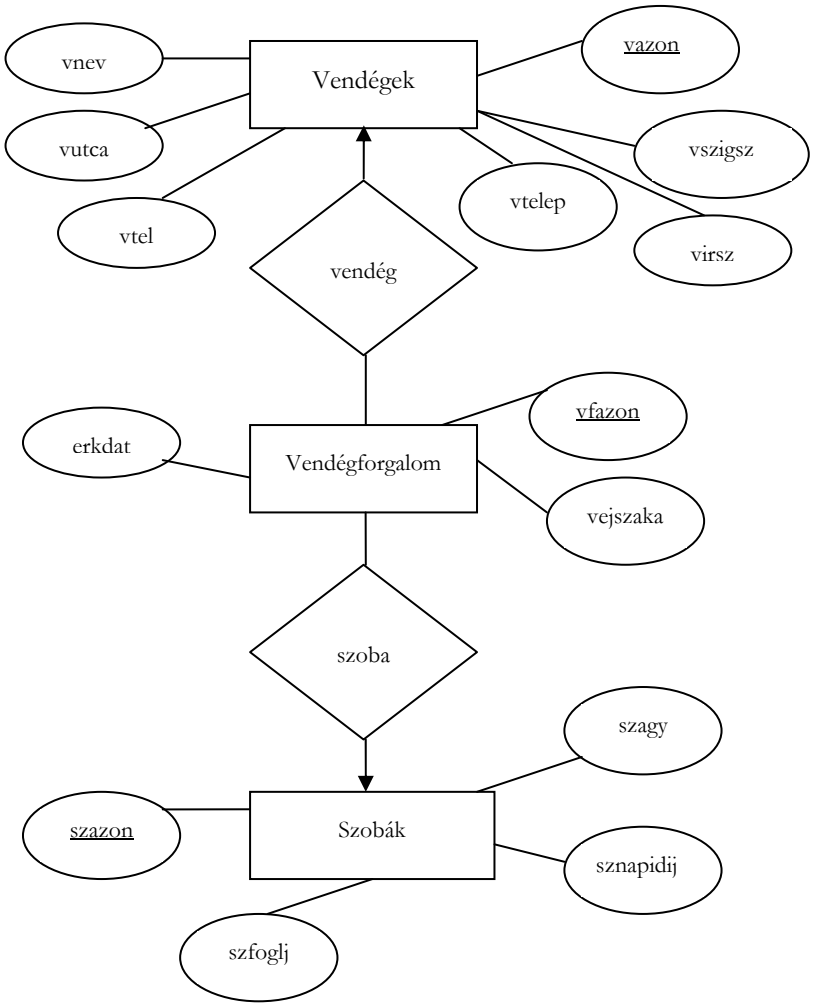
A **Vendégek** egyedhalmazban a *vszigsz* attribútum nem biztos, hogy mindig egyedi lesz, ezért bevezettük a *vazon* kulcs-attribútumot. A **Vendégforgalom** egyedhalmazban összetett kulcsot lehetne definiálni (vazon, szazon, erkdat) gyenge egyedhalmazzal dolgozva, de célszerűbb bevezetni a *vfazon* kulcs-attribútumot.

A rendszerben az alábbi **egy-több típusú** kapcsolatok jelennek meg:

Egyedhalmazok	Kapcsolatnév
Vendégek - Vendégforgalom	vendég
Szobák - Vendégforgalom	szoba

2.17. táblázat.

Az adatbázis E/K diagramját a **2.12. ábra** mutatja.



2.12. ábra. A Panzió adatbázis E/K diagramja

2.3. Ellenőrző kérdések

1. Ismertesse röviden az adatbázis-tervezés, megvalósítás folyamatát!
2. Ismertesse röviden a következő OOP fogalmakat: objektum, osztály, attribútum, metódus, kapcsolat, alosztály!
3. Ismertesse az ODL alaptípusokat!
4. Ismertesse az ODL összetett típusait!
5. Ismertesse az ODL osztály (interface) deklaráció szintaktikai elemeit!
6. Ismertesse a következő E/K alapfogalmakat: egyedhalmaz, attribútum, kapcsolat!
7. Adja meg az E/K-modell grafikus alapelemeit!
8. Ismertesse a következő E/K fogalmakat: sokágú kapcsolat, gyenge egyedhalmaz, szerepek, kapcsolat attribútum, alosztály-öröklődés!

3. A relációs adatmodell

A relációs adatbázis adatmodelljét elkészíthetjük a relációk elméletére épülő „normalizálási” megfontolások segítségével, illetve a korábban tárgyalt modellek „átírásával”.

3.1. Alapfogalmak

A relációs modellben az adatok relációkban (táblázatokban) jelennek meg. A táblázat *oszlopai* az egyed attribútumai, tulajdonságai. A táblázat *soraiban* egy konkrét egyed-előfordulás attribútum értékei jelennek meg.

A **3.1. táblázat**ban a *Gépjármű* reláció néhány attribútumát, sorát látjuk:

rendszám	alvázszám	gjFajta	gyártmány	gjSzín
ABC123	WVWZZZ3GH3S456987	személygépkocsi	Volkswagen	fehér
XYZ987	SZKAFF2JG5L123456	személygépkocsi	Suzuki	piros
ACY456	RENKKZ4GH6Q321456	haszonjármű	Renault	fekete

3.1. táblázat.

3.1.1. Sémák

A **relációséma** a reláció nevének és attribútum halmazának megadását jelenti.

Az előbbi táblázatnak megfelelő „hiányos” relációséma:

Gépjármű (rendszám, alvázszám, gjFajta, gyártmány, gjSzín)

Megállapodás alapján a sémabeli attribútumok **sorrendjét** használjuk a reláció (táblázat), vagy egy sor megadásakor is.

Az attribútumok száma a **reláció foka**.

Az adatbázis modellben általában több relációsémát kell megadnunk.

A relációsémákból álló halmazt **adatbázissémának** nevezzük.

3.1.2. Sorok

A reláció attribútumokat tartalmazó sorától különböző sorait soroknak (tuple) nevezzük.

Például a 3.1. táblázatbeli reláció egy sora:

(XYZ987, SZKAFF2JG5L123456, személygépkocsi, Suzuki, piros)

A sorban levő attribútum érték a **komponens**.

A reláció sorok halmaza, ezért *nem fordulhat elő két teljesen azonos sor*. Ezt a feltételt több esetben úgy tudjuk megvalósítani, hogy olyan „mesterséges" attribútumot is megadunk, aminek csak az egyediség biztosítása a szerepe.

3.1.3. Értéktartományok

Az attribútumok **típusa** csak atomi típus lehet. Nem megengedett a struktúra, halmaz stb. Minden attribútumhoz tartozik egy **értéktartomány**, ami elemi típus. *A komponensek csak az attribútumnak megfelelő értéktartományból vehetnek fel értékeket.*

3.1.4. Relációk előfordulásai

Időben a relációk változnak. Új sorok kerülnek a relációba, más sorokat törlünk, meglévő sorok komponenseit módosítjuk. Természetesen a relációséma is változhat, de ez az előbbiekhez képest ritkábban fordul elő, inkább csak a tervezési, tesztelési folyamat során. Az adott reláció sorainak halmazát **reláció előfordulásnak** nevezzük. Az éppen létező állapotot **aktuális előfordulásnak** nevezzük.

3.2. Relációs algebra

A relációs algebrai kifejezések alapjait a relációk képezik, mint operandusok. Egy reláció megadható a nevével (pl.: R vagy S) vagy közvetlenül, sorainak egy listájával.

3.2.1. Halmazműveletek

Unió

Az **R** és **S** relációk uniója (egyesítése) eredményül olyan relációt ad, amely tartalmazza az **R** és **S** sorait. Az ismétlődő sorokat elhagyjuk. *Jelölés*: ∪.

A művelet elvégzéséhez a két relációnak *azonos fokúnak* kell lenni. Legyen az **R** és **S** relációk sémája: (A, B, C)! Eltérő attribútum nevek esetén alkalmazhatjuk az átnevezés műveletét.

Példa: Legyen az R és S reláció egy-egy előfordulása az alábbi, végezzük el az unió műveletet:

R	A	B	C
	2	-2	3
	4	1	4
	1	2	4

és

S	A	B	C
	1	2	4
	2	-2	1
	4	1	4
	4	1	2

Megoldás: Először az egyesített reláció részeként vesszük az **R** reláció sorait, majd az **S** relációból hozzáírjuk azokat a sorokat, amelyek nem fordultak elő az **R**-ben.

Eredményül az alábbi reláció adódik:

R∪S	A	B	C
	2	-2	3
	4	1	4
	1	2	4
	2	-2	1
	4	1	2

Metszet

Az **R** és **S** relációk metszete eredményül olyan relációt ad, amely azokat a sorokat tartalmazza, amelyek az **R** relációban és az **S** relációban is előfordulnak. *Jelölés*: ∩.

Példa: Végezzük el az előző relációkon a metszet műveletet!

Megoldás: a relációk metszete:

R∩S	A	B	C
	4	1	4
	1	2	4

Különbség

Az **R** és **S** relációk különbsége eredményül olyan relációt ad, amely tartalmazza az **R** azon sorait, amelyek az **S** relációban nem fordulnak elő.

Jelölés: \.

Példa: Végezzük el az előző relációkon a különbség műveletet!

Megoldás: a relációk különbsége:

R \ S	A	B	C
	2	-2	3

3.2.2. Relációs műveletek

Vetítés (projekció)

Az **R** reláció vetítésével olyan relációt hozunk létre, amely az **R** bizonyos oszlopait tartalmazza valamilyen sorrendben. *Jelölés*: $\pi_L(R)$, ahol **L** attribútum listát jelent.

Példa: Legyen az R egy előfordulása az alábbi, végezzük el a $\pi_{B, C}(R)$ vetítés műveletet:

R	A	B	C
	-2	-2	3
	4	6	4
	1	2	4
	3	1	5
	-1	7	2

Megoldás: a vetítés eredménye:

$\pi_{B, C}(R)$	B	C
	-2	3
	6	4
	2	4
	1	5
	7	2

Kiválasztás (szelekció)

Az **R** relációra alkalmazott kiválasztás művelet olyan relációt eredményez, amely az **R** sorainak részhalmazát adja. A sorok kiválasztásához feltételt (logikai kifejezést) kell megfogalmaznunk. *Jelölés*: $\sigma_F(R)$, ahol **F** jelenti a feltételt.

A feltételben konstansok, attribútum nevek, operátorok (pl.: +, -, AND), théta-relációk (<, >, =, ≤, ≥, ≠) fordulhatnak elő. Például: F lehet A > 1 AND B < 4, ahol A és B attribútumokat jelent. A σ művelet az **R** reláció minden sorára megvizsgálja, hogy az F feltétel teljesül-e (behelyettesíti a kifejezésbe az attribútumok soron levő értékeit). Ha az adott soron teljesül a feltétel, akkor az bekerül az eredmény relációba, egyébként nem.

Példa: Legyen **R** a fenti reláció, **F** = A > 1 AND B < 4! Végezzük el a $\sigma_F(R)$ műveletet!

Megoldás: a szelekció eredménye:

$\sigma_{A>1 \text{ AND } B<4}(R)$	A	B	C
	3	1	5
	3	3	3

Csak a fenti két sor értékeivel teljesül a megadott feltétel.

Descartes-szorzat

Az **R** és **S** reláció *Descartes-szorzata* (egyszerűen szorzata) eredményül olyan relációt ad, amely az összes lehetséges módon párosított **R**-beli és **S**-beli sorokat tartalmazza. *Jelölés*: **R × S**.

Az eredmény reláció sémája az **R** és **S** sémájának egyesítése. Az **R** attribútumai megelőzik az **S** attribútumait. Azonos attribútum nevek esetén vagy átnevezünk, vagy ún. minősített nevet használunk. A minősítést a reláció nevével végezzük, pl. R.A jelöli az **R**-beli A attribútumot, S.A az **S**-belit.

Az eredmény reláció sorainak száma: **n · m**, ha *n* az **R** reláció, *m* az **S** reláció számossága. Nagy *n* és *m* esetén a sorok száma igen nagy lehet.

Példa: Adottak az alábbi R és S relációk előfordulásai. Határozzuk meg az R és S relációk **R × S** szorzatát!

R	A	B	C
	2	a	3
	4	b	4
	1	c	4

és

S	A	D	E
	1	2	e
	2	-2	f
	4	1	g

Megoldás: Az A attribútum előfordul mindegyik relációban, ezért minősítve az eredmény relációban az R.A és S.A neveket használjuk:

R × S	R.A	B	C	S.A	D	E
	2	a	3	1	2	e
	2	a	3	2	-2	f
	2	a	3	4	1	g
	4	b	4	1	2	e
	4	b	4	2	-2	f
	4	b	4	4	1	g
	1	c	4	1	2	e
	1	c	4	2	-2	f
	1	c	4	4	1	g

3.2.3. Származtatott műveletek

Természetes összekapcsolás

A *Descartes-szorzat* egy sorában megjelenő **R**-beli és **S**-beli sorok általában logikailag függetlenek. A gyakorlatban inkább a logikailag összetartozó sorok kezelése a fontosabb.

A *természetes összekapcsolás* műveletében az **R** és **S** relációk azon sorait illesszük egy sorba, amelyek **megegyeznek** az **R** és **S** sémájának összes közös attribútumán. *Jelölés*: ⋈.

Legyenek **R** és **S** közös attribútumai: A és B. A természetes összekapcsolás művelet elvégzéséhez, az alábbi lépéseket kell végrehajtani:

- Elkészítjük **R** és **S** *Descartes-szorzatát*.
- Végrehajtjuk a $\sigma_{R.A=S.A \text{ AND } R.B=S.B}(R \times S)$ műveletet (kiválasztás adott feltétellel: csak a logikailag összetartozó **R**-beli és **S**-beli sorok maradnak).
- Végrehajtjuk a $\pi_{A, B, .., H, K, ..}(\sigma_{R.A=S.A \text{ AND } R.B=S.B}(R \times S))$ vetítés műveletet (az **R** sémája (A, B, C, …), az **S** sémája (A, B, H, K, …)). Az azonos attribútumú oszlopok közül elhagyjuk az **S**-belieket.

Más megközelítése a művelet elvégzésének: Végigmegyünk az **R** összes során és megvizsgáljuk, hogy illeszthető-e sor hozzá az **S**-ből (akkor illeszthető, ha azonos értékek szerepelnek a közös attribútumokban). Ha

igen, akkor elvégezzük az összes illesztést. A természetes összekapcsolás sémájában a közös attribútumok csak egyszer jelennek meg.

Példa: Adottak az alábbi R és S relációk előfordulásai. Határozzuk meg az R és S relációk **R** ⋈ **S** természetes összekapcsolását!

R	A	B	C
	2	a	3
	4	b	4
	1	c	4

és

S	A	D	E
	1	2	e
	2	-2	f
	4	1	g
	4	1	d

Megoldás: Először kiszámítjuk a *Descartes-szorzatot*. Az A attribútum előfordul mindegyik relációban, ezért minősítve a szorzat relációban az R.A és S.A neveket használjuk:

R × S	R.A	B	C	S.A	D	E
	2	a	3	1	2	e
	2	a	3	2	-2	f
	2	a	3	4	1	g
	2	a	3	4	1	d
	4	b	4	1	2	e
	4	b	4	2	-2	f
	4	b	4	4	1	g
	4	b	4	4	1	d
	1	c	4	1	2	e
	1	c	4	2	-2	f
	1	c	4	4	1	g
	1	c	4	4	1	d

Ezután kiválasztjuk az R.A = S.A feltételnek megfelelő sorokat:

σ(R × S)	R.A	B	C	S.A	D	E
	2	a	3	2	-2	f
	4	b	4	4	1	g
	4	b	4	4	1	d
	1	c	4	1	2	e

Végül elhagyjuk az S.A oszlopot (vetítünk):

R ⋈ S	A	B	C	D	E
	2	a	3	-2	f
	4	b	4	1	g
	4	b	4	1	d
	1	c	4	2	e

Théta-összekapcsolás

Néha a *természetes összekapcsolásnál* „általánosabb” illesztésre is szükségünk van. Azaz nemcsak az „=” relációt engedjük meg, hanem bármely Théta-operátort (<, >, =, ≤, ≥, ≠) az **R**-beli és **S**-beli attribútumok között.

Az **R** és **S** relációk *théta-összekapcsolása* az R × S relációból kiválasztja azokat a sorokat, amelyek megfelelnek egy **Θ** (théta) feltételnek. A **Θ** helyett a könnyebb írásmód miatt **F**-et használunk. *Jelölés*: $\mathbf{R} \bowtie_{\mathbf{F}} \mathbf{S}$, ahol az **F** feltétel konstansokat, attribútum neveket, théta-operátorokat, logikai műveleteket tartalmazhat.

Példa: Adottak az R és S relációk alábbi előfordulásai. Határozzuk meg az R és S relációk **R** $\bowtie_F$ **S** Théta-összekapcsolását, ahol F = R.A < S.A OR B = E!

R	A	B	C
	2	a	3
	4	b	4
	1	c	4

és

S	A	D	E
	1	2	e
	2	-2	b
	4	1	g
	4	1	c

Megoldás: Először kiszámítjuk a *Descartes-szorzatot*. Az A attribútum előfordul mindegyik relációban, ezért minősítve a szorzat relációban az R.A és S.A neveket használjuk:

R × S	R.A	B	C	S.A	D	E
	2	a	3	1	2	e
	2	a	3	2	-2	b
	2	a	3	4	1	g
	2	a	3	4	1	c
	4	b	4	1	2	e
	4	b	4	2	-2	b
	4	b	4	4	1	g
	4	b	4	4	1	c
	1	c	4	1	2	e
	1	c	4	2	-2	b
	1	c	4	4	1	g
	1	c	4	4	1	c

Ezután kiválasztjuk az **F** feltételnek megfelelő sorokat:

$R \bowtie_{R.A < S.A \text{ OR } B = E} S$	R.A	B	C	S.A	D	E
	2	a	3	4	1	g
	2	a	3	4	1	c
	4	b	4	2	-2	b
	1	c	4	2	-2	b
	1	c	4	4	1	g
	1	c	4	4	1	c

3.3. A relációs algebra kiterjesztése

Az SQL-ben a relációk *multihalmazok*. Ugyanaz a sor többször is megjelenhet egy SQL-relációban (lekérdezés eredménytáblában). Ezért kibővítjük a relációs algebrát, egyrészt multihalmazokra is értelmezzük az eddigi műveleteket, másrészt bevezetünk új műveleteket az SQL-nek megfelelően.

3.3.1. Halmazműveletek

A relációkban ismétlődő sorokat is megengedünk. Ezután a művelet alsó indexében **H** jelöli a halmaz, **M** jelöli a multihalmaz műveletet. Ha nincs alsó index, akkor alapértelmezésben a multihalmazos változatot használjuk.

Unió

Az $\mathbf{R} \cup_M \mathbf{S}$ művelet esetén egy sor annyiszor fordul elő az eredményben, amennyi az **R**-ben és az **S**-ben levő *előfordulások összege*.

Metszet

Az $\mathbf{R} \cap_M \mathbf{S}$ művelet esetén egy sor annyiszor fordul elő az eredményben, amennyi az **R**-ben és az **S**-ben levő *előfordulások minimuma*.

Különbség

Az **R** $\setminus_M$ **S** művelet esetén egy sor annyiszor fordul elő az eredményben, amennyi az **R**-ben és az **S**-ben levő *előfordulások különbsége. Ha ez negatív, akkor egyszer sem.*

Példa: A megadott R és S reláció előfordulásokban megengedünk ismétlődő sorokat. Végezzük el az unió, metszet, különbség műveleteket!

R	A	B	C
	2	-2	3
	4	1	4
	1	2	4
	1	2	4

és

S	A	B	C
	1	2	4
	2	-2	1
	4	1	4
	4	1	2
	1	2	4

Megoldás:

A relációk egyesítése:

R $\cup_M$ S	A	B	C
	2	-2	3
	4	1	4
	1	2	4
	1	2	4
	1	2	4
	2	-2	1
	4	1	4
	4	1	2
	1	2	4

A relációk metszete:

R $\cap_M$ S	A	B	C
	4	1	4
	1	2	4
	1	2	4

A relációk különbsége:

$R \setminus_M S$	A	B	C
	2	-2	3

3.3.2. Kiválasztás

A $\sigma_F(R)$ kiválasztás tartalmaz egy **R** relációt (lehetnek ismétlődő sorok) és egy **F** feltételt. A művelet végrehajtása hasonlóan történik, mint ismétlődő sorok nélkül. Ha egy sor megfelel a feltételnek, akkor bekerül az eredményrelációba.

3.3.3. Vetítés

Ha **R** egy reláció, akkor a $\pi_L(R)$ az R reláció **L** listára történő vetítése. A vetítési lista a következő típusú elemeket tartalmazhatja:

1. Az R egy attribútumát.
2. **Attribútum átnevezést**: az R reláció X attribútumát és átnevezzük Y-ra. *Jelölés*: $X \rightarrow Y$.
3. **Kifejezés átnevezést**: az E kifejezést átnevezzük Y-ra. *Jelölés*: $E \rightarrow Y$.

Az eredmény egy olyan reláció, amelynek sémája megegyezik az L listában felsorolt attribútumokkal, az átnevezéseket figyelembe véve.

3.3.4. Relációk Descartes-szorzata

Az $\mathbf{R \times S}$ művelet végrehajtása hasonlóan történik, mint ismétlődő sorok nélkül. Ha R és S két reláció, akkor az szorzat egy olyan reláció, amelynek sémája az R és az S attribútumaiból áll.

Ha egy **r** sor **n**-szer jelenik meg az R-ben, az **s** pedig **m**-szer az S-ben, akkor a szorzatban az **rs** sor **n·m**-szer jelenik meg.

3.3.5. Összekapcsolások

Az $\mathbf{R \bowtie S}$ és $\mathbf{R \bowtie_F S}$ műveletek végrehajtása hasonlóan történik, mint ismétlődő sorok nélkül.

3.3.6. Ismétlődések kiküszöbölése

A $\delta(R)$ művelet az R relációból kiszűri az ismétlődő sorokat.

3.3.7. Csoportosítás és összesítés

Az SQL csoportosításokat (GROUP BY) és az összesítéseket (SUM, AVG, MIN, MAX, COUNT) általában együtt használjuk. A $\gamma_L(R)$ művelet, csoportosítást és összesítést végez az R reláción, az L attribútum lista alapján:

1. Az **R** sorait csoportokba osztjuk. Valamennyi csoport azokból a sorokból épül fel, amelyek az L lista csoportosított argumentumaira egy bizonyos értékkel rendelkeznek. Ha nincsenek csoportosított attribútumok, akkor a teljes R reláció lesz egy csoport.
2. Minden egyes csoportra képezünk egy olyan sort, amely a következőket tartalmazza:
   - a csoportosított attribútumok értékeit az adott csoportra és
   - a csoport összes sorára vonatkozó összesítéseket, amelyeket az L lista összesített attribútumai határoznak meg.

Példa: A megadott R reláción végezzük el $\gamma_{A,\ SUM(B*C)->X}(R)$ műveletet!

R	A	B	C	D
	2	3	2	e
	2	3	4	f
	2	3	1	g
	2	3	1	d
	4	4	2	e
	4	4	4	f
	4	4	1	g
	4	4	1	d
	1	4	2	e
	1	4	4	f
	1	4	1	g
	1	4	1	d

Megoldás:

$\gamma_{A, SUM(B*C)->X}(R)$	A	X
	1	32
	2	24
	4	32

3.3.8. Rendezés

A $\tau_L(R)$ művelet rendezi az **R** reláció sorait az **L** attribútum listának megfelelően. Ha L az A_1, A_2 …, A_n attribútumok listája, akkor az R sorai először az A_1 attribútum értékei szerint vannak rendezve. Egyenlő A_1 értékek esetén az A_2 értékei számítanak. Azok a sorok, amelyek megegyeznek az A_1 és A_2 értékeiken, az A_3 értékek szerint kerülnek rendezésre és így tovább. Azok a sorok, amelyek még az A_n attribútumon is megegyeznek, tetszőleges sorrendbe helyezhetők. Az alapértelmezett rendezési sorrend, növekvő, de csökkenőre változtatható az attribútum utáni DESC kulcsszóval.

3.4. Relációs sémák tervezése

Ha túl sok információt (attribútumot) helyezünk el egy relációban, akkor problémák, anomáliák fordulhatnak elő. Alapvető anomáliák:

- Redundancia

Az információk feleslegeses ismétlődnek több sorban.

- Beszúrási problémák

Nem tudunk egy adatot (sort) bevinni a relációba, ha pl. egy kötelezően megadandó másik adatot (vele kapcsolatban levőt) nem ismerünk.

- Módosítási problémák

Módosítunk egy adatot az egyik sorban, és ugyanazt az adatot egy másik sorban nem módosítjuk.

- Törlési problémák

Törlések után egy értékhalmaz üressé válhat (csak egész sorokat törölhetünk), vele együtt egyéb információ is eltűnhet a relációból.

Az előző problémák megelőzéséhez, megszüntetéséhez megvizsgáljuk a rendelkezésünkre álló technikákat (függőségek vizsgálata, relációk felbontása, normálformák).

3.5. Funkcionális függőségek

3.5.1. Alapok

Legyenek az **R** reláció attribútumai: A_1, A_2, …, A_n. Akkor az **R** reláció sémája: $R(A_1, \dots, A_n)$.

Az **X** *attribútum halmaz*, ha $X \subseteq R$ (az X attribútumai R-ből valók), jelölés: $X = (A_{i1}, A_{i2}, \dots, A_{ik})$ vagy $X = A_{i1}A_{i2} \dots A_{ik}$.

Az Y attribútum halmaz *funkcionálisan függ* X-től, ha **R** bármely két sorára igaz, hogy ha megegyeznek X-en, akkor Y-on is megegyeznek ($X, Y \subseteq R$). *Jelölés*: $X \rightarrow Y$).

Példa: Legyen az **R** reláció: (Név, Irányítószám, Település, Utca, Telefon)! Adjunk meg funkcionális függéseket!

Megoldás:
Település, Utca → Irányítószám
Irányítószám → Település
Név, Település, Utca → Telefon

Ezután a relációs sémához hozzátartozik a függőségek halmaza **F** is: (R, F). Megadjuk a séma attribútumait és az *érdemi függéseket* (az adatbázis bármely változása esetén fennállnak). $X \rightarrow Y$ teljesülhet úgy is, hogy az adott relációban nincs is két olyan sor, amik X-en megegyeznek. X-nek és Y-nak nem kell diszjunktnak lenniük (lehetnek közös attribútumaik).

Az **r** reláció akkor *illeszkedik* az (R, F) sémára, ha az attribútumai az **R**-ben adottak és teljesülnek benne az **F** függései.

Armstrong-axiómák

Az axiómák megadják, hogy néhány funkcionális függőségből milyen továbbiak következnek:

(1) *Reflexivitás*: Ha $X, Y \subseteq R$ és $Y \subseteq X$, akkor $X \rightarrow Y$.

(2) *Bővítés*: Ha $X, Y, Z \subseteq R$ és $X \rightarrow Y$, akkor $XZ \rightarrow YZ$.

(3) *Tranzitivitás*: Ha $X, Y, Z \subseteq R$, $X \rightarrow Y$ és $Y \rightarrow Z$, akkor $X \rightarrow Z$.

Levezethetőség

Egy $X \rightarrow Y$ függőség *levezethető* egy adott **F** függőséghalmazból, ha az axiómák ismételt alkalmazásával **F**-ből megkapjuk $X \rightarrow Y$-t.

Jelölés: $F \vdash X \rightarrow Y$.

Néhány további szabály, ami levezethető az **Armstrong** axiómákból:

Egyesítési szabály

Ha Y, Z funkcionálisan függnek X-től, akkor YZ is funkcionálisan függ X-től:

$$\{X \rightarrow Y, X \rightarrow Z\} \vdash X \rightarrow YZ.$$

Bizonyítás:

1) $X \rightarrow Y$: ez F-beli

2) $XZ \rightarrow YZ$: bővítve Z-vel

3) $X \rightarrow Z$: ez F-beli

4) $X \rightarrow XZ$: bővítve X-el

5) $X \rightarrow YZ$: 4) és 2) + tranzitivitás

Áltranzitív szabály

Ha Y funkcionálisan függ X-től és Z funkcionálisan függ YW-től, akkor Z funkcionálisan függ XW-től is:

$$\{X \rightarrow Y, YW \rightarrow Z\} \vdash XW \rightarrow Z.$$

Bizonyítás:

1) $X \rightarrow Y$: ez F-beli

2) $XW \rightarrow YW$: bővítve W-vel

3) $YW \rightarrow Z$: ez F-beli

4) $XW \rightarrow Z$: 2) és 3) + tranzitivitás

Felbontási szabály

Ha Y funkcionálisan függ X-től és $Z \subseteq Y$, akkor Z is funkcionálisan függ X-től:

$\{X \rightarrow Y\} \vdash X \rightarrow Z$

Bizonyítás:

1) $X \rightarrow Y$: ez F-beli

2) $Y \rightarrow Z$: reflexivitás

3) $X \rightarrow Z$: 1) és 2) + tranzitivitás

Lezárás

Ha **F** egy függéshalmaz, akkor a lezártja $\mathbf{F}^+$ az **F**-ből levezethető összes függés: $F^+ = \{X \rightarrow Y \mid F \vdash X \rightarrow Y\}$.

Az $\mathbf{F}^+$ nagy lehet: pl. **R** $(A, B_1, \ldots, B_n)$ és $\mathbf{F} = \{A \rightarrow B_i \mid 1 <= i <= n\}$, akkor ez *n* db függés. $\mathbf{F}^+$-ban benne van minden $A \rightarrow B_{i1} \ldots B_{il}$ függés, azaz 2^n eleme van.

Ha X egy attribútum halmaz (R, F)-ben, akkor *lezártja* $X^+(F) = \{A \in R \mid F \vdash X \rightarrow A\}$, azaz azon attribútumok, amik függnek X-től.

Belátható, hogy $X \subseteq X^+(F) \subseteq R$.

$F \vdash X \rightarrow Y \Leftrightarrow Y \subseteq X^+(F)$ (Az $X \rightarrow Y$ függés akkor, és csak akkor vezethető le F-ből, ha Y benne van $X^+(F)$-ben.)

Következmény: Ha minden X-re ismerjük, vagy ki tudjuk számítani $X^+(F)$-et, akkor tetszőleges $X \rightarrow Y$ függésről eldönthető, hogy F^+-beli-e vagy sem.

3.5.2. A reláció kulcsa

$X \subseteq R$ *szuperkulcsa* az (R, F) sémának, ha $F \vdash X \rightarrow R$. Azaz, X szuperkulcs, ha $R = X^+(F)$.

$X \subseteq R$ *kulcsa* az (R, F) sémának, ha szuperkulcs és nincs olyan valódi részhalmaza, ami szuperkulcs.

3.5.3. Attribútum halmaz lezártjának kiszámítása

Az X attribútum halmaz lezárásának kiszámítására az alábbi algoritmussal történhet:

$X_0 = X$,

.

$X_i = \ldots$,

$X_{i+1} = X_i \cup \{A \in R \mid \text{van olyan } U \rightarrow V \in F, \text{ hogy } U \subseteq X_i \text{ és } A \in V\}$,

.

$X^+(F) = X_{\text{utolsó}}$ (amikor már nem bővíthető)

Kezdetben legyen X lezártja maga az X! Ezután az X attribútumait felhasználva keresünk olyan A attribútumot, amelyet az F függései meghatároznak. Ha van ilyen, akkor azt hozzáveszszük X-hez, és folytatjuk az előző keresést. Ha már nincs ilyen A attribútum, vagy az X tartalmazza az R összes attribútumát, akkor meghatároztuk az X lezártját.

Példa: Legyen: **R** (A, B, C, D), **F** = {A → B, B → C, BC → D}! Határozzuk meg az A attribútum $A^+(F)$ lezártját!

Megoldás:

$X_0 = \{A\}$;

$X_1 = \{A, B\}$, mert teljesül az A → B függés F-ben;

$X_2 = \{A, B, C\}$, mert teljesül a $B \rightarrow C$ függés F-ben;

$X_3 = \{A, B, C, D\} = X_{utolsó}$, mert teljesül az $BC \rightarrow D$ függés R-ben.

Az algoritmust használva adott X-ről el lehet dönteni, hogy (szuper)kulcs-e! Ha $X^+(F) = R$ igaz, akkor szuperkulcs. Ha X-ből nem hagyható el attribútum, akkor X kulcs. Látható, hogy az előző példában $A^+(R)=R$, tehát az A attribútum kulcsa az **R** relációnak.

3.5.4. Felbontások

Adott (R, F) sémából anomáliát nem tartalmazó olyan felbontást akarunk előállítása, amiből ugyanaz az információ nyerhető, mint az eredetiből.

$\varrho = (R_1, \ldots, R_k)$ az (R, F) séma felbontása, ha $R_i \subseteq R$ és $\cup_{i=1,k} R_i = R$.

Ha **r** egy (R, F) sémára illeszkedő reláció, akkor legyen:

$r_i = \pi_{Ri}(r)$ és $m_\varrho(r) := r_1 \bowtie r_2 \bowtie \ldots \bowtie r_k$

Tételek:

(1) $r \subseteq m_\varrho(r)$
(2) $r_i = \pi_{Ri}(m_\varrho(r))$
(3) $m_\varrho(m_\varrho(r)) = m_\varrho(r)$

Hűséges felbontás

Adott (R, F). A séma **ϱ** felbontása *hűséges* (veszteségmentes, lossless), ha minden (R, F)-re illeszkedő **r** relációra $r = m_\varrho(r)$.

Példa: Legyen (R, F) a következő: R(A, B, C), $F = \{C \rightarrow A\}$. Legyen **r** az alábbi reláció:

R	A	B	C
	a	c	e
	a	d	f
	b	c	g
	b	d	h

Hűséges-e az s(A, B) és t(B, C) felbontás?

A számítás mutatja, hogy r(A, B, C) ≠ s(A, B) ⋈ t(B, C), azaz ez a felbontás nem hűséges.

Az (R, F) séma $\rho = (R_1, R_2)$ akkor és csak akkor hűséges, ha $R_1 \setminus R_2$, vagy $R_2 \setminus R_1$ levezethető $R_1 \cap R_2$ -ből.

3.5.5. Normálformák

A relációs sémák normál formára hozása (normalizálás) biztosítja a különböző anomáliák megszüntetését. A bevezetett normálformák közül csak a fontosabbakkal foglalkozunk.

Boyce-Codd normálforma (BCNF)

Az (R, F) séma Boyce-Codd normálformában van, ha tetszőleges, *nem triviális* ($A \not\subset X$) $X \rightarrow A \in F^+$ és esetén X szuperkulcs.

Csak olyan függések vannak, ahol *a szuperkulcs minden attribútumot meghatároz*, nincs tranzitív függés kulcstól. A BCNF relációban **nincs redundancia**, ezért fontos szerepet játszik az adatbázis tervezésben.

Ha (R, F) nem BCNF, akkor van olyan $X \rightarrow Y \in F$, amely jobboldalának valamely A attribútumára $X \rightarrow A$ (nem triviális) és X nem szuperkulcs.

A nem BCNF vizsgálathoz csak F függéseit kell végignézni.

Tetszőleges (R, F) sémának van hűséges felbontása BCNF relációkra.

Bizonyítási elv:

- Ha az (R, F) séma BCNF, akkor rendben van.
- Ha nem, akkor két valódi részre bontjuk hűségesen: (R_1, R_2).
- A hűséges felbontásokat ismételjük R_1-re és R_2-re.
- Az eljárás véget fog érni, mert 2 attribútum esetén nem kell tovább bontani. Hűséges lesz, mert ha egy hűséges felbontás egyik részét tovább bontjuk, akkor hűséges marad.

A felbontás menete:

1. Keresünk a felbontandó sémában egy olyan $X \to A \in F^+$ függést, ami sérti a BCNF tulajdonságot, azaz A és X része a sémának, $A \notin X$ és X nem szuperkulcs:

   $R_1 := XA$; $R_2 := R \setminus \{A\}$.
   Ezek kisebbek **R**-nél: R_2 a különbség művelet miatt, R_1 pedig azért, mert ha $R_1 = R$ volna, akkor $X \to XA = R$ miatt X szuperkulcs lett volna. Hűséges a felbontás, mert a kétrészes teszttel $R_1 \cap R_2 = X \to A = R_1 \setminus R_2$. A felbontás után az $X \to A$ függéssel nem lesz több probléma:

   - R_2-ben nincs A,
   - R_1-ben X szuperkulcs lesz.

2. Minden felbontás után meg kell nézni, hogy a kapott relációk BCNF-ben vannak-e. Ehhez meg kell konstruálni F^+_S-et, ha S a vizsgált reláció: ez az F^+_R azon függéseiből áll, amiknek mindkét oldala F-ben van. Ezeket a függéseket úgy kapjuk, hogy minden $X \subseteq S$ részhalmazra kiszámoljuk $X^+(F)$-et és $X \to Y$ pontosan akkor lesz benne F^+_S-ben, ha
   $Y \subseteq X^+(F) \cap S$.

3 attribútum esetén a BCNF tulajdonság csak úgy sérülhet, ha $X \to Y$, ahol X, Y egy-egy attribútum és X nem kulcs.

Mindig ellenőrizni kell, hogy a kapott relációkban mik a (szuper)kulcsok, hogy egy függésről el tudjuk dönteni, hogy sérti-e a BCNF-et vagy nem.

Függőség megőrzése

Adott (R, F) séma és ennek egy $\varrho = (R_1, \ldots, R_k)$ felbontása. $\pi_\varrho(F) := \{X \to Y \in F^+ \mid \exists i\ (1 \le i \le k)\ X, Y \subseteq R_i\}^+$ az F függéseinek vetítése a ϱ felbontásra. A ϱ felbontás *függőségőrző*, ha $\pi_\varrho(F) = F^+$.

Van olyan reláció, amit nem lehet függőségőrző módon BCNF-ekre bontani, azaz *a felbontás BCNF-re nem feltétlenül függőségőrző*. Bevezetünk egy új normálformát, ami biztosítja függőségek megőrzését.

3NF

Az (R, F) séma A attribútuma *prím* (elsődleges), ha szerepel valamelyik kulcsban.

Az (R, F) séma *3NF* (harmadik normálformájú), ha tetszőleges *nem triviális* $X \to A \in F^+$ függés esetén vagy X *szuperkulcs* vagy A *prím* attribútum.

Minden BCNF séma egyben 3NF is.

Ha (R, F) egy 3NF séma, akkor minden nem prím A attribútumra és

$X \subseteq R$ kulcsra igaz, hogy nincs olyan Y, hogy $X \to Y$, $Y \not\to X$, $Y \to A$ és $A \notin Y$. (Nem-elsődleges attribútum nem függ tranzitívan kulcstól.)

3NF tulajdonság ellenőrzése

Háromnál kevesebb attribútumos reláció mindig 3NF.

Ha (R, F) nem 3NF, akkor van olyan $X \to Y \in F$, amely jobboldalának valamely A attribútumára $X \to A$ nem triviális, X nem szuperkulcs és A nem prím. (Az ilyen $X \to A \in F^+$.)

Az $X^+(F)$ kiszámításával tudjuk ellenőrizni egy adott $X \to A$ függésre, hogy X szuperkulcs-e ($X^+(F)=R$).

Algoritmus 3NF vizsgálathoz:

- Meghatározzuk az összes kulcsot.
- Meghatározzuk az összes prím attribútumot.
- Minden **F**-beli $X \to Y$ függésre nézzük meg:
  - Igaz-e, hogy $Y \subseteq X$? Ha igen, akkor a függés triviális.
  - Igaz-e, hogy X kulcs? Ha igen, akkor nem sérti a feltételt.
  - Igaz-e, hogy Y-ban csak prím attribútumok vannak? Ha igen, akkor nem sérti a feltételt.
  - Ha egyik sem, akkor van olyan függés, ami sérti a feltételt, tehát *nem 3NF*.

3NF felbontás

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Példa: Bontsuk 3NF-re az R = (A, B, C, D, E), F = {AE → BC, AC → D, CD → BE, D → E} sémát!

Megoldás: Ez nem 3NF, mert a kulcsok: semelyik egyelemű halmaz nem kulcs (csak D lehetne, de az ő lezártja csak DE), viszont a kételeműek közül szuperkulcs lesz AC, AD, AE (A-nak benne kell lennie minden kulcsban, mert A nincs jobboldalon), AB viszont nem szuperkulcs. Ezek kulcsok is lesznek, mert egyik egyelemű sem volt kulcs. A *prím attribútumok*: A, C, D, E, vagyis B nem az.

A CD → B függés rossz a 3NF szempontjából, mert CD nem szuperkulcs és B nem prím. Ezután R_1={C, D, B}, R_2={A, C, D, E}. A megoldást az R_2 további vizsgálatával, felbontásával kapjuk.

3.5.6. Többértékű függőségek

Adott az R (Tanár, Tantárgy, Osztály) relációséma. A reláció többértékű függőségei (egyik attribútumhoz egy másik attribútum több értéke tartozhat, jelölés: —»):

Tanár —» Tantárgy (egy tanár több tantárgyat taníthat)

Tanár —» Osztály (egy tanár több osztályban taníthat)

A **3.2. táblázat** a fenti reláció egy lehetséges előfordulását mutatja:

Tanár	Tantárgy	Osztály
Tóth Péter	matematika	1. A
Tóth Péter	matematika	2. B
Tóth Péter	fizika	2. B
Tóth Péter	fizika	3. B
Simon Éva	kémia	2. A
Simon Éva	kémia	2. B
Simon Éva	kémia	3. B

3.2. táblázat.

Egy tanárhoz a tárgy és osztály attribútum értékek egy halmaza tartozik.

Azt mondjuk, hogy az X attribútum halmaztól többértékűen függ az Y attribútum halmaz, jelölésben X —» Y, ha minden X-en felvett értékhez létezik az Y attribútum halmazon felvett értékek olyan halmaza, amelyik semmilyen függési kapcsolatban sincsenek az R \ X \ Y attribútum halmazon felvett értékekkel.

A pontos definíció a következő:

Az R relációs sémában teljesül az X —» Y többértékű függőség, ha minden az R sémához tartozó **r** reláció előfordulásra igaz, hogy tetszőleges t_1, t_2 sorokra, melyekre $t_1[X] = t_2[X]$, léteznek t_3, $t_4 \in r$ melyekre:

$$t_3[XY] = t_1[XY]$$

$$t_3\{R \setminus XY] = t_2[R \setminus XY]$$

$$t_4[XY] = t_2[XY]$$

$$t_4[R \setminus XY] = t_1[R \setminus XY]$$

teljesül.

A funkcionális függőség *egyenlőséggeneráló* függőség, azaz két dolog egyenlőségéből másik két dolog egyenlőségére következtet. A többértékű függőség *sorgeneráló* függőség, azaz két sor létezése, melyek valahol egyenlők, maga után vonja más sorok létezését.

A többértékű függőségekhez is létezik egy teljes és ellentmondásmentes axióma rendszer, hasonlóan a funkcionális függőségek Armstrong-axiómáihoz.

Negyedik normálforma (4NF)

A Boyce–Codd normálforma általánosítható arra az esetre, ha funkcionális függőségek mellett a többértékű függőségeket is figyelembe vesszük.

Legyen R relációs séma, F többértékű és funkcionális függőségek halmaza R-en. R *negyedik normálformában* (4NF) van, ha tetszőleges X —» Y $\in F^+$ többértékű függőségre, melyre $Y \not\subset X$ és $R \neq XY$, teljesül, hagy X szuperkulcs R-ben.

Ha egy séma 4NF, akkor BCNF is.

Az R séma F *többértékű* és *funkcionális* függőségek halmazával felbontható $\rho = (R_1, R_2, \ldots, R_k)$ alakba, ahol minden egyes R_i már 4NF-ben. A szétvágás veszteségmentes.

A módszer ugyanazt az elvet követi, mint a BCNF felbontás, azaz ha az R séma nincs 4NF-ben, akkor van egy X —» Y függőség, ami megsérti a 4NF feltételt. Ekkor R felbontható $R_1 = XY$ és $R_2 = R \setminus Y$ sémákra, mindkettőnek kevesebb attribútuma van, mint R-nek, azaz az eljárás véges időn belül véget ér.

Példa: Bontsuk fel az R (Tanár, Tantárgy, Osztály) többértékű függéseket tartalmazó relációt 4NF-re!

Megoldás:

R_1 = (Tanár, Tantárgy) és R_2 = (Tanár, Osztály)

Az R relációséma $\rho = (R_1, R_2)$ szétvágása akkor és csak akkor veszteségmentes összekapcsolású az F többértékű és funkcionális függőségek halmazára vonatkozólag, ha

$$(R_1 \cap R_2) \text{ —» } (R_1 \setminus R_2)$$

3.6. ODL-sémák átírása relációsra

Első megközelítésben az osztályokból relációk, a tulajdonságokból attribútumok lesznek. Problémát okozhatnak a nem atomi típusú tulajdonságok, illetve a kapcsolatok.

3.6.1. Attribútumok átírása

A problémát az összetett típusok: struktúra, halmaz, multihalmaz, lista, tömb átírása okozza. Általában az összetett típushoz új relációt kell definiálni, majd gondoskodni kell a megfelelő kapcsolat biztosításáról (közös attribútum).

Példa: Írjuk át a Gépjármű osztály ODL-sémáját relációssá!

Emlékeztetőül az ODL-séma:

```
interface  Gépjármű
        (keys rendszám, alvázszám)
{
        attribute string rendszám;
        attribute string alvázszám;
        attribute string motorszám;

        attribute enum Fajta
{személygépkocsi, haszonjármű, busz, motorkerékpár, egyéb} gjFajta;
        attribute string gyártmány;
        attribute string típus;
        attribute enum Szín {fehér, fekete, piros, …} gjSzín;
        attribute integer sajátTömeg;
        attribute integer gyártásiÉv;
        attribute integer hengerűrtartalom;
        relationship Set<Tulajdonos> gépjárműTulajdonosai
                inverse Tulajdonos::tulajdonosGépjárművei;
};
```

Megoldás: A problémát a 2 felsorolás okozza. A felsorolásokra bevezetünk egy-egy relációt:

Fajta (fajtaKód, fajtaNév)

Szín (színKód, színMegnevezés)

A *fajtaKód* és *színKód* „mesterséges" attribútumok az új relációk kulcsai lesznek, másrészt a *Gépjármű* relációhoz való kapcsolatot biztosítják.

A **Fajta** reláció előfordulása lehet:

fajtaKód	fajtaNév
1	személygépkocsi
2	haszonjármű
3	motorkerékpár
4	busz
5	egyéb

A **Szín** reláció előfordulása lehet:

színKód	színMegnevezés
1	fehér
2	piros
3	fekete
4	sárga
5	szürke

Ezek felhasználásával a *Gépjármű* reláció sémája a *Tulajdonos* kapcsolat nélkül:

Gépjármű (rendszám, alvázszám, motorszám, *fajtaKód*, gyártmány, típus, *színKód*, sajátTömeg, gyártásiÉv, hengerűrtartalom)

A két kulcs attribútum közül a rendszámot használjuk elsődleges kulcsnak, ezt jelöli az **aláhúzás,** a dőlt betűtípus az idegen kulcsot (FOREIGN KEY) jelöli, amivel kapcsolatot építünk a megfelelő relációval.

3.6.2. Kapcsolatok átírása

Az ODL-ben a kapcsolat hivatkozástípusú, egy vagy több értékkel. A relációs modellben csak atomi típusok lehetnek, ezért a kapcsolathoz a *kapcsolódó reláció kulcs attribútumait* használjuk.

Példa: Írjuk át az előző feladatban szereplő kapcsolatot is!

Megoldás: A *Tulajdonos* reláció kulcsa a *személyiIgSzám* attribútum, ami egyedileg azonosítja a gépjármű tulajdonosokat. Ezt kell hozzávenni a *Gépjármű* relációséma eddigi attribútumaihoz:

Gépjármű (rendszám, alvázszám, motorszám, *fajtaKód*, gyártmány, típus, *színKód*, sajátTömeg, gyártásiÉv, hengerűrtartalom, *személyiIgSzám*)

Mivel a kapcsolat többértékű (egy gépjárműnek több tulajdonosa is lehet), a fenti séma redundanciát is tartalmaz.

Példa: Adjuk meg a *Tulajdonos* ODL osztály relációsémáját!

Megoldás: Az ODL-séma:

```
interface  Tulajdonos
        (key személyiIgSzám)     {
        attribute string személyiIgSzám;
        attribute string név;
        attribute string anyjaNeve;
        attribute string szülHely;
        attribute struct Dátum
                {integer év, integer hónap, integer nap} szülDátum;
        attribute struct Cím
                {string irányítószám, string település, string utca} lakcím;
        relationship Set<Gépjármű> tulajdonosGépjárművei
              inverse Gépjármű::gépjárműTulajdonosai; };
```

A problémát a két struktúra típusú tulajdonság, illetve a kapcsolat okozhatja. A születési dátumot gyorsan elintézhetjük, mert a relációs adatbázis-kezelők használják a dátum adattípust. A struktúrában szereplő mezőket különálló attribútumként kezeljük, így azok atomi típusúak lesznek. Struktúra halmaz esetén (egy tulajdonosnak több címe is lehet) *Cím* relációt kellene létrehozni egy mesterséges **egyedi azonosító** attribútummal (pl.: sorszám). Ezt a kulcsot helyeznénk el a *Tulajdonos* relációban (redundancia).

A megoldás:

Tulajdonos (személyiIgSzám, név, anyjaNeve, szülHely, szülDátum, irányítószám, település, utca, *rendszám*)

Mivel egy tulajdonosnak több gépjárműve lehet, a relációséma redundanciát tartalmaz. A mostani és a korábban említett redundancia megszüntetéséhez bevezetünk egy új relációt:

GépjárművekTulajdonosok (*rendszám*, *személyiIgSzám*)

sémával. Az „összetartozó" rendszámokat, személyi igazolvány számokat helyezzük egy sorba. Ebben a relációban a két attribútum együttesen adja a kulcsot. Ezután a *Gépjármű* sémából kivesszük a *személyiIgSzám*, a *Tulajdonos* sémából a *rendszám* attribútumot.

Végezetül a **GépjNyilv** adatbázis redundancia nélküli sémája az alábbi:

Gépjármű (rendszám, alvázszám, motorszám, *fajtaKód*, gyártmány, típus, *színKód*, sajátTömeg, gyártásiÉv, hengerűrtartalom)

Fajta (fajtaKód, fajtaNév)

Szín (színKód, színMegnevezés)

Tulajdonos (személyiIgSzám, név, anyjaNeve, szülHely, szülDátum, irányítószám, település, utca)

GépjárművekTulajdonosok (*rendszám*, *személyiIgSzám*)

3.6.3. Feladatok

1. Példa: A Számlák adatbázis ODL sémája (2.1.4 fejezet 1. Példa) alapján adjuk meg az adatbázis relációs sémáját!

Megoldás: Először átírjuk az ODL osztályokat relációkká:

Partnerek (PKod, PNev, PIrsz, PTelep, PUtca)
SzlaFejek (SZAzon, SZSzam, SZFizMod, SZTeljDat, SZTipus, SZMegj)
SzlaTételek (TMegn, TVamt_SZJ, TMe, TMenny, TEar, TSzazal)

A kapcsolatokat is figyelembe véve módosítani kell az SzlaFejek és az SzlaTételek sémáját. Az SzlaTételek relációban bevezetjük a TSorsz (tétel sorszám) attribútumot, hogy a SZAzon attribútummal együtt összetett elsődleges kulcsot kapjunk. Az ODL tervben 3 felsorolás típusú attribútum (SZFizMod, SZTipus, TMe) szerepelt. A mértékegységek kezelésére bevezetjük a Megys relációt. A fizetési mód és a számla típusa attribútumok kezelését az adatbázis-kezelő segítségével oldjuk meg (pl.: CHECK CONSTRAINT, számított mező használata). Az adatbázis relációs sémája:

Partnerek (PKod, PNev, PIrsz, PTelep, PUtca)

SzlaFejek (SZAzon, SZSzam, SZFizMod, SZTeljDat, SZTipus, SZMegj, *PKod*)

SzlaTételek (*SZAzon*, TSorsz, TMegn, TVamt_SZJ, *MeKod*, TMenny, TEar, TSzazal)

Megys (MeKod, MeNev)

3.7. Az E/K diagramok átírása relációs adatbázissémára

Az E/K modellben használt **attribútumok** általában atomi típusúak, ezért az egyedhalmazokat attribútumaikkal együtt egy-egy relációba visszük. Gyenge egyedhalmazok esetén további attribútumokat (a kapcsolódó egyedhalmaz kulcs attribútumait) is beviszünk a relációba.

A **kapcsolatokat** első lépésben szintén relációkká írjuk át, ha van saját attribútuma a kapcsolatnak, akkor azt is bevisszük a relációba. Az ismétlődések miatt előfordulhat, hogy attribútumokat át kell nevezni.

Példa: Írjuk át a **2.5.** ábrán látható E/K diagramot relációs adatbázissémává!

Megoldás:

1. lépés: Egyedhalmazokból, kapcsolatokból relációkat készítünk.

Termékek (tkod, tnev, tear)

Vevők (vkod, vnev, vkapcs, vutca, vtelep, virsz, vtel)

RendelésFejek (rkod, rdat, rszdat, rfdij)

RendelésTételek (tazon, rmenny, rear)

Vevők_RendelésFejek (vkod, rkod)

RendelésFejek_RendelésTételek (rkod, tazon)
RendelésTételek_Termékek (tazon, tkod)

2. lépés: Ha lehet „egyszerűsítjük" (elhagyunk relációkat) a sémát. Például, ha a *RendelésFejek* relációba bevisszük a **vkod** attribútumot, akkor elhagyható a *Vevők_RendelésFejek* reláció. Ha a *RendelésTételek* relációba bevisszük az **rkod** attribútumot, akkor elhagyható a *RendelésFejek_RendelésTételek* reláció. Ha a *RendelésTételek* relációba bevisszük a **tkod** attribútumot, akkor elhagyhatjuk a *RendelésTételek_Termékek* relációt.

A megoldás:

Termékek (tkod, tnev, tear)

Vevők (vkod, vnev, vkapcs, vutca, vtelep, virsz, vtel)

RendelésFejek (rkod, rdat, rszdat, rfdij, *vkod*)

RendelésTételek (tazon, rmenny, rear, *rkod*, *tkod*)

3.8. Ellenőrző kérdések

1. Ismertesse röviden a következő relációs modell alapfogalmakat: sémák, sorok, értéktartományok!
2. Sorolja fel a relációs algebra műveleteit!
3. Ismertesse a relációs algebra halmazműveleteit!
4. Ismertesse a relációs algebra következő műveleteit: vetítés, kiválasztás!
5. Ismertesse a relációs algebra Descartes-szorzat műveletét!
6. Ismertesse a relációs algebra összekapcsolási műveleteit!
7. Mit jelent a relációs algebra kiterjesztése?
8. Ismertesse a kiterjesztett relációs algebra következő műveleteit: ismétlődések kiküszöbölése, csoportosítás és összesítő műveletek, rendezés!
9. Ismertesse a következő fogalmakat: funkcionális függés, logikai következmény!
10. Adja meg az Armstrong-axiómákat!
11. Bizonyítsa az egyesítési szabályt!
12. Bizonyítsa az áltranzitív szabályt!

13. Bizonyítsa az felbontási szabályt!
14. Ismertesse a következő fogalmakat: levezethetőség, lezárás (függés-halmaz, attribútum halmaz)!
15. Ismertesse a következő fogalmakat: szuperkulcs, kulcs!
16. Ismertesse a következő fogalmakat: reláció felbontása, hűséges felbontás!
17. Ismertesse a Boyce-Codd normálformát!
18. Ismertesse a 3. normálformát!
19. Ismertesse a minimális fedés fogalmát!
20. Ismertesse a többértékű függőség fogalmát!
21. Ismertesse a 4. normálformát!
22. Hogyan történik az ODL-sémák átírása relációsra?
23. Hogyan történik az E/K diagramok átírása relációs sémára?

4. Adattárolás

4.1. Adatelemek és mezők

4.1.1. Adatelemek ábrázolása

Számok

Minden adatot, a számokat is, bájtok sorozatával ábrázolunk. Egy INTEGER típusú attribútumot általában kettő vagy négy bájttal ábrázolunk, FLOAT típusú mezőt rendszerint négy vagy nyolc bájttal.

Rögzített hosszú karaktersorozatok

Az SQL CHAR(n) típusa rögzített, *n* hosszú karakterláncot reprezentál. A megfelelő mező *n bájtból álló tömb*. Ha a mező értéke *n*-nél rövidebb, akkor a tömböt helykitöltő karakterekkel (pad character) töltjük fel.

Ha egy mezőt CHAR(6) típusúnak deklaráltunk, és egy sorban 'kapu' a mező értéke, akkor a tömb a 'kapu¬¬' karakterekből áll. A ¬ karakter a helykitöltést jelzi.

Változó hosszú karaktersorozatok

A relációkban olyan oszlopok is előfordulnak, amikben a karakterláncok hossza változik. Ez az SQL VARCHAR(n) adattípusa. Ilyenkor n + 1 bájtot rendelünk a karakterlánc tárolásához a tényleges hossztól függetlenül. A VARCHAR karakterlánc reprezentációjának két szokásos módja a következő:

1. **Hossz plusz tartalom**: Az első bájt(ok) értéke a karaktersorozat bájtjainak számával egyezik meg.
2. **Záró karakterrel végződő lánc**: A karaktersorozat végét egy speciális karakter jelzi, mely nem megengedett a karakterláncokban.

Dátumok és időpontok

Az SQL2-szabvány a dátumokat (DATE típus) 10 hosszú karakterlánccal ábrázolja EEEE-HH-NN (év-hó-nap) formátumban. Például a '2005-05-01' a 2005. május 1-jét ábrázolja.

Az időpontot (egész számú másodperceket) egy OO:PP:MM (óra:perc:másodperc) formátumú 8 karakteres láncként ábrázolhatjuk. A TIME típus a másodperc törtrészét is tartalmazza. A 8 karakter után pont következik, majd annyi számjegy, amennyi a másodperc törtrészének leírásához szükséges.

A DATETIME típus együtt tárolja a dátumot és az időpontot.

Bitek

Egy bitsorozat esetén (az SQL2-ben BIT(n) típusa) minden 8 bitet egy bájtba foglalhatunk. Ha az n nem osztható 8-cal, akkor az utolsó bájt fel nem használt bitjeitől eltekinthetünk. A 010111110001 bitsorozat ábrázolható úgy, hogy 01011111 az első bájt, és 00010000 a második. A logikai érték (**Bool-érték)** ábrázolásához elegendő egy bit, 10000000 jelenti az igaz, 00000000 a hamis értéket. Könnyebb ellenőrés miatt 11111111-et használhatjuk az igaz, 00000000-t a hamis ábrázolására.

Felsorolási típusok

A felsorolási típus értékeit egész számokkal ábrázolhatjuk. Csak annyi bájtot használunk, amennyi minimálisan szükséges. Például a {PIROS, FEHÉR, ZÖLD} felsorolásban a PIROS-at ábrázolhatjuk a 0-val, a FEHÉR-et 1-gyel, a ZÖLD-et 2-vel. Az összes ábrázolható a 00, 01, 10 bit párokkal. Egész számként ábrázolva, például a ZÖLD 1 bájton 00000010.

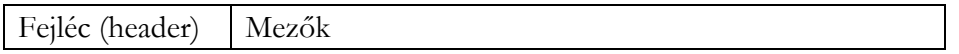
Bináris, nagy objektumok

A nagy méretek miatt ezek az adatok több blokkon férnek csak el, pl.: a különböző formátumú képfájlok (például GIF vagy JPEG), filmek MPEG vagy hasonló formátumban, szövegek (Word dokumentumok, Excel táblázatok). Az ilyen értékeket bináris, nagy objektumoknak (binary, large object — **BLOB**) nevezik. Legegyszerűbb esetben a blokkok láncolt listája tárolja az adatot. Gyorsabb elérést biztosít, ha az összetartozó blokkokat azonos cilinderen helyezzük el. Több lemez használatával csíkokra szedhetjük a BLOB-ot, ami párhuzamos olvasást biztosít.

4.1.2. Rekordok és mezők

A relációk sémáit az adatbázis tárolja. A séma tartalmazza a rekord mezőinek neveit, adattípusait és a mezők kezdetét a rekordon belül.

Az alábbi ábra egy rekord felépítését mutatja:

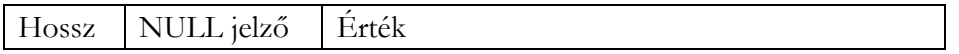


4.1. ábra. Egy rekord felépítése

A fejléc tartalma lehet:

- A rekord kezelésével kapcsolatos információk (pl.: törölt-e, melyik relációhoz tartozik stb.).
- A mezők típusa.
- Időbélyeg (mikor módosult utoljára).

Az alábbi ábra egy mező „általános" felépítését mutatja:



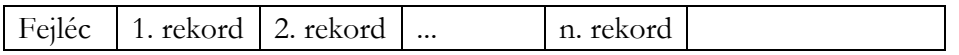
4.2. ábra. Egy mező felépítése

A rekordok *típusa* lehet:

- **Kötött formátumú:** a mezők száma, mérete, típusa és sorrendje rögzített.
- **Változó formátumú:**
  - Mezők hossza nem fix (szöveget tartalmazó adattáblák).
  - Ismétlődő mezők lehetnek, és az ismétlések száma nem fix.
  - Többértékű mezők.
  - Nagyméretű mezők (képek, videók stb.).

Rögzített hosszú rekordok blokkjai

A rekordokat a lemez blokkjaiban (page) tároljuk. A blokkokat (lapokat) általában az adatbázis-kezelő mozgatja. A blokkméret az adott operációs rendszerre jellemző blokkméret n - szerese, ahol n = 1, 2, 4, 8 stb. A blokkméret gyakran 8 Kbájt, azaz 8192 bájt. A **4.3** ábrán egy rekordokat tartalmazó blokk felosztása látható:



4.3. ábra. Rekordokat tartalmazó tipikus blokk

A *blokkfejléc* (block header) többek között az alábbi információkat tartalmazhatja:

1. Hivatkozás egy vagy több olyan blokkra, melyek egy blokkláncolat részét képezik.
2. Információ a blokk láncban betöltött szerepéről.
3. Információ arról, hogy a blokk rekordjai melyik relációhoz tartoznak.
4. Egy olyan táblázat, amely a blokk összes rekordjának az *eltolási értékét* (blokkon belüli kezdőcím) tartalmazza.
5. Egy blokkazonosító.
6. Időbélyegzés(ek) jelölik a blokk utolsó módosítási és/vagy elérési idejét.

A rekord *eltolási érték táblázat* a fejlécen kívül is lehet (pl. MS SQL Server 2005 esetén a blokk végén található), mérete változik a blokkon levő rekordok számának megfelelően.

Általában a blokkot nem töltik ki teljesen a rekordok, hogy az esetleges módosításokhoz legyen hely a blokkon belül. Bizonyos adatbázis-kezelők tiltják, mások megengedik, hogy egy rekord *blokkhatáron átnyúljon*. Átnyúló rekord esetén blokk-mutató jelzi, hogy melyik blokkban folytatódik a rekord.

Példa: Tegyük fel, hogy a rekordjaink 400 bájt hosszúak, 8K-s (8192 bájt) blokkokat használunk. Ebből 96 bájtot fogunk blokkfejlécre felhasználni. Rekordonként 16 bájtos bejegyzés található az eltolási érték táblázatban. Hány rekordot helyezhetünk el a blokkban?

Megoldás: Egy rekord az eltolási érték táblázat bejegyzéssel együtt 416 bájtot foglal el. A fejrész helyfoglalása után 8096 bájtot használhatunk az adatok tárolására. Az osztás (8096/416) elvégzése után adódik, hogy maximum 19 rekordot helyezhetünk el egy blokkban és marad szabadon 192 bájt.

Változó hosszú rekordok blokkjai

Egy lehetséges megoldás (a 4.2 ábra egy másik megoldást sugall) a következő: tegyük az összes rögzített hosszú rekordot a változó hosszú mezők elé. Ezután a rekord fejlécébe helyezzük el az alábbi információkat:

1. A rekord hossza.
2. Mutatók, vagyis eltolási értékek az összes változó hosszú mező elejére.

Ismétlődő mezőket tartalmazó rekordok

Az F mezőből változó számú előfordulást tartalmaz egy rekord, az F mező rögzített hosszú. Az F mező összes előfordulásából csoportot képezünk, és a rekord fejlécébe elhelyezünk egy mutatót, amely az első előfordulásra mutat. Legyen az F mező egy előfordulásának a hossza L bájt. Ekkor a csoportbeli adatok eléréséhez az F eltolási értékéhez hozzáadjuk az L minden egész számú többszörösét (0, L, 2L, 3L stb.) mindaddig, amíg el nem érjük az F-et követő mező eltolási értékét.

Változó formátumú rekordok

A változó formátumú rekordok ábrázolásának legegyszerűbb módja, mikor címkézett mezők sorozatát adjuk meg. Minden címkézett mező a következőkből áll:

1. Információ a mezőről:
   a) mezőnév,
   b) a mező típusa, ha ez nem nyilvánvaló a mezőnévből vagy a séma-információból,
   c) a mező hossza, ha ez nem következik a típusból.
2. A mező értéke.

Átnyúló rekordok

Az adatbázis-kezelő rendszereknek olyan adatokat is kell kezelniük, amelyekhez nagy értékek tartoznak, melyek nem férnek el egy blokkban.

Ilyenkor a rekordokat két vagy több blokkba eső részekre vágjuk szét. Egy rekordnak azt a részét, amely egy blokkba kerül, rekordtöredéknek hívjuk. Olyan rekordot, amelynek két vagy több töredéke van, átnyúló rekordnak nevezünk.

Ha a rekordok átnyúlók is lehetnek, akkor több információt kell tárolnunk minden rekord és rekordtöredék fejlécében:

1. Minden rekordnak vagy töredéknek a fejlécében kell lennie egy olyan bitnek, amely megmondja, hogy ez egy töredék vagy nem.
2. Ha ez egy töredék, akkor ahhoz is kellenek bitek, hogy megmondják, hogy ez a rekord első vagy utolsó töredéke.

3. Ha ugyanannak a rekordnak következő és/vagy előző töredéke is létezik, akkor a töredékben szükséges mutatókat elhelyezni ezekre a további töredékekre is.

BLOB-ok

Egy BLOB-ot blokkok sorozatán kell tárolni. A rekordban mutató jelzi a kezdő blokkot. A gyorsabb olvasás miatt célszerű a blokkokat egy cilinderen elhelyezni, vagy több lemez használata esetén lemezcsíkokat használni. Index alkalmazásával biztosíthatjuk a részadatok (blokkok) közvetlen elérését is (pl.: adott film 25. percének megtekintése).

4.2. Rekordmódosítások

4.2.1. Beszúrás

Rendezetlen tárolás

A beszúráshoz keresünk egy olyan blokkot, amelyben van üres hely. Ha nem találunk, akkor használatba veszünk egy új blokkot, és abba tesszük a rekordot.

Rendezett tárolás

A beszúráshoz először meg kell keresnünk a rendezettségnek megfelelő blokkot. Ha van még hely a blokkban, akkor betesszük ide a rekordot. A rendezettség miatt lehet, hogy a blokkon belül a rekordokat el kell majd csúsztatni, hogy az új rekord a megfelelő helyre kerüljön. Az eltolási érték táblát módosítani kell.

Előfordulhat, hogy nincs hely a blokkban az új rekord számára. Ilyenkor az alábbi megoldásokat használhatjuk:

1. **Blokkok átszervezése.** Szabad helyet keressünk egy „közeli” blokkban, majd rekord áthelyezésekkel, a megfelelő mutatók karbantartásával szabadítunk fel helyet az új rekordnak.
2. **Túlcsordulási terület használata.** Minden blokk fejlécében helyet tartunk fenn egy olyan mutató számára, amely egy túlcsordulási blokkra mutat.

4.2.2. Törlés

A törlés történhet:

- **Blokk átszervezésével**
  A blokkok elcsúsztatásával a felszabaduló területet hozzáfűzzük a szabad területhez, ez eltolási érték táblát az új elrendezésnek megfelelően módosítjuk. Ha használtunk túlcsordulási blokkot, akkor azt is törölni kell.

- **Törlési jel használatával**
  Az eltolási érték táblában a törölt rekord bejegyzésében jelezzük a törlés tényét pl. NULL-érték bejegyzéssel, és/vagy a törölt rekord fejlécében pl. egy bittel jelezzük a törlési állapotot. Felvitelnél, módosításoknál a töröltként megjelölt területeket is felhasználhatjuk.

4.2.3. Módosítás

Ha a módosított rekordhossz nagyobb a réginél:

1. Elcsúszatjuk a rekordokat
2. Túlcsordulási blokkot használunk

Ha a módosított rekordhossz kisebb a réginél:

1. Felszabadítjuk a felesleges területet a törléshez hasonlóan
2. Szükség esetén töröljük a túlcsordulási blokkot

4.3. Indexstruktúrák

Az index segítségével egy adott kulcsú rekord megkeresése nagyon gyorsan történik (csak kevés blokkolvasásra van szükség).

4.3.1. Indexek rendezett szekvenciális fájlokon

Az adatszerkezet a következő:

- **rendezett** adatfájl
- **indexfájl** (egy kulcs-mutató párokat tartalmazó rekordok).

Az indexfájl valamennyi K keresési kulcsához tartozik egy mutató, amely az adatfájl azon rekordjára mutat, amely tartalmazza a K keresési kulcsot. Ez a szerkezet különösen akkor hasznos, amikor a keresési kulcs az adattábla elsődleges kulcsa.
Az indexek lehetnek:

- **Sűrűk**
  Az adatfájl minden rekordjához létezik egy bejegyzés az indexfájlban.

- **Ritkák**
  Az indexfájlban csak az adatfájl néhány (nem az összes) rekordjából van kulcs.

Példánkban a szekvenciális fájl a kulcs szerint rendezett. A kulcsok egész számok, 100 egymást követő többszörösei. *Feltételezzük*, hogy egy blokkban csak három rekord fér el (általában ennél jóval több rekord fér el egy 8K-s blokkban).

Sűrű indexek

Az indexfájlban minden adatrekordból van kulcs. Az indexfájl a kulcsra ugyanúgy rendezett, mint az adatfájl.

Példa: Legyen az adatfájl az alábbi (4.4. ábra). Ábrázoljuk a sűrű indexeket tartalmazó indexfájlt, ha egy blokkban 5 indexrekord fér el!

100	
200	
300	

400	
500	
600	

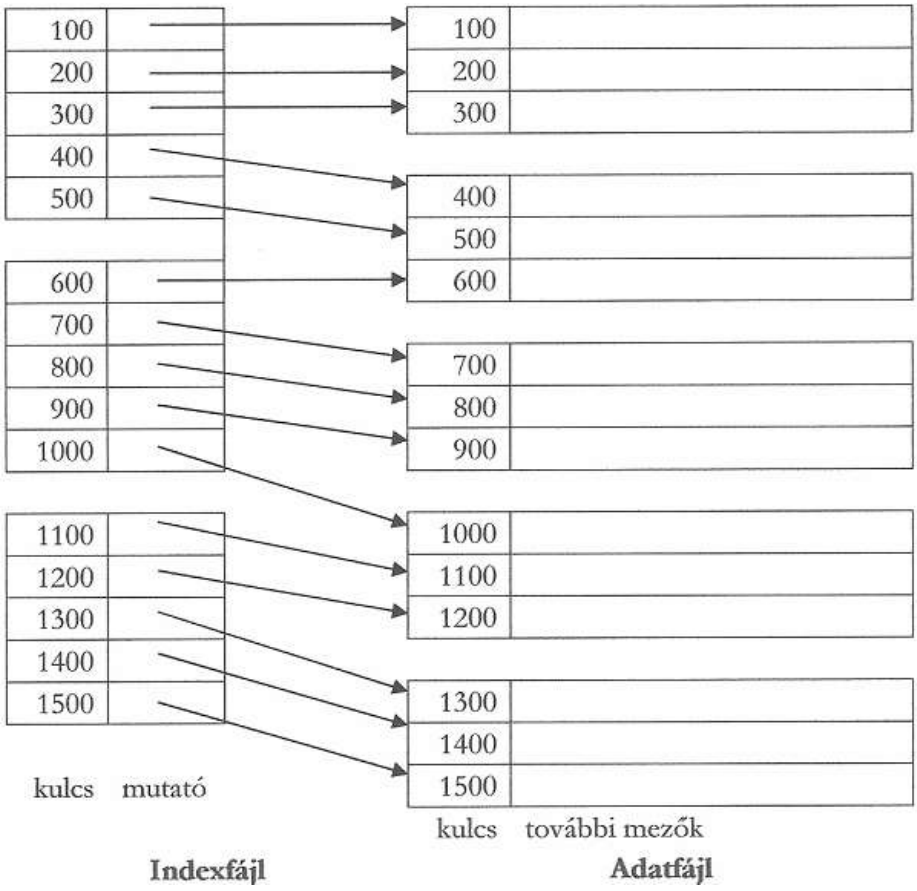
700	
800	
900	

1000	
1100	
1200	

1300	
1400	
1500	

4.4. ábra. A rendezett szekvenciális adatfájl blokkjai

Megoldás: A megoldást a 4.5. ábra mutatja.



4.5. ábra. Sűrű index szekvenciális adatfájlon

Példánkban, egy blokkban 5 indexrekord fér el, de általában ennél sokkal több. Az indexfájl sokkal kevesebb blokkot használ, mint az adatfájl.

Az index alapú keresést gyorsíthatjuk:

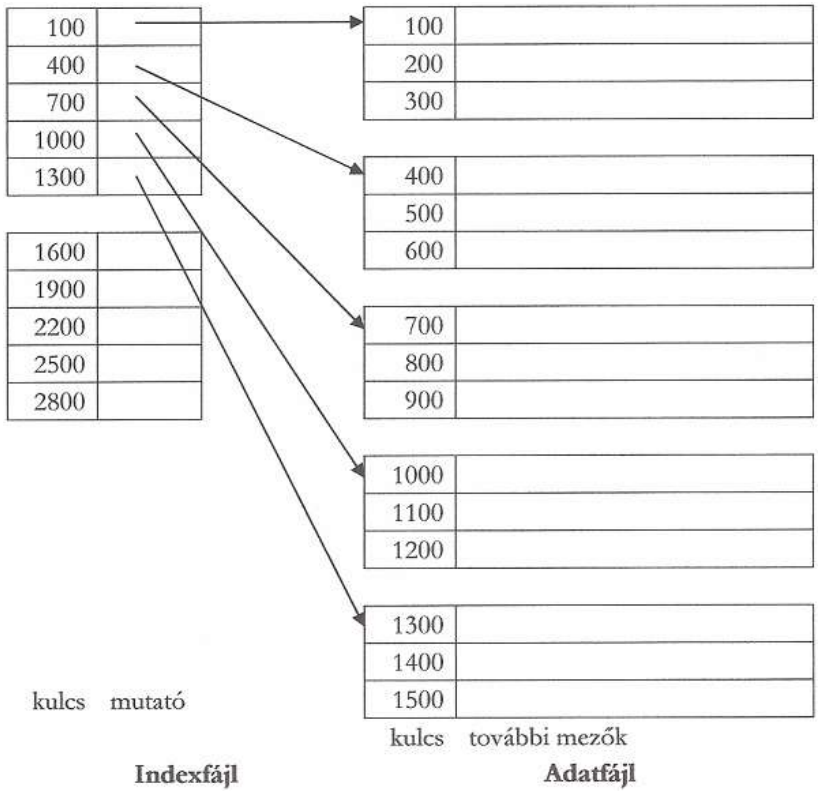
1. Ha a kulcs megtalálásához bináris keresést használunk. Ha **n** darab indexblokkunk van, akkor átlagosan $\mathbf{\log_2 n}$ blokkot kell beolvasni.
2. Ha a teljes indexfájlt memóriában tartjuk.

Ritka indexek

Ritka index esetén az adatfájl összes kulcsa nem jelenik meg az indexfájlban.

Példa: Az előzőleg megadott adatfájlhoz (4.4. ábra) készítsük el a ritka indexeket tartalmazó indexfájlt! Minden adatblokkban a legkisebb kulcsot vigyük az indexbe, az indexblokkban továbbra is 5 rekord fér el.

Megoldás: A megoldást a 4.6. ábra mutatja.



4.6. ábra. Ritka index szekvenciális fájlon

Egy rekord megtalálásához megkeressük azt a legnagyobb indexértéket, amely **kisebb vagy egyenlő, mint a keresett kulcs.** Az indexbejegyzés mutatója megadja a rekord lehetséges blokkját.

Többszintű indexelés

Meglévő indexre újabb indexet készíthetünk, így az első szintű index használatát hatékonyabbá tehetjük. A második és annál magasabb szintű indexek kötelezően *ritkák*.

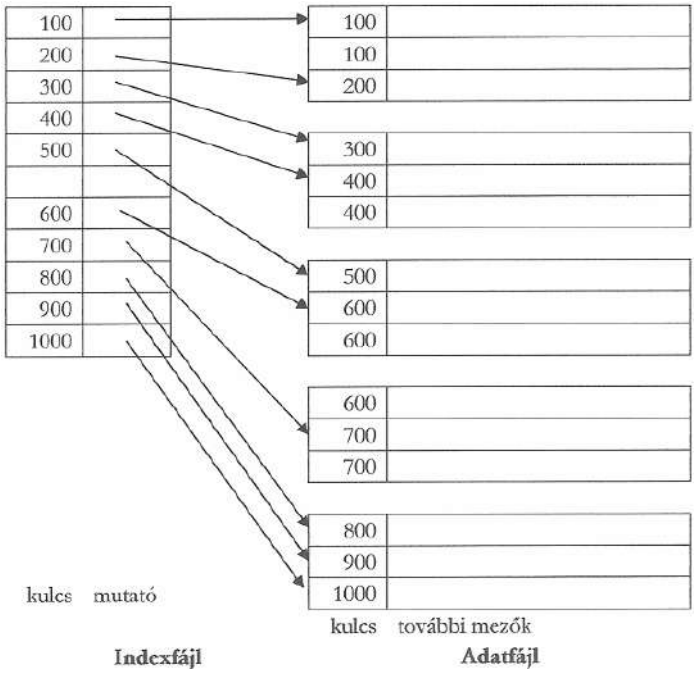
Indexelés ismétlődő keresési kulcsérték esetén

Az adatfájl rekordjai rendezettek, de az indexkulcsban előfordulhatnak azonos értékek is.

Sűrű index

Példa: Az előzőleg megadott adatfájlhoz (4.4. ábra) készítsük el a sűrű indexeket tartalmazó indexfájlt! Az adatblokkokban megengedett a kulcsérték ismétlődése. Az indexblokkban továbbra is 5 rekord fér el.

Megoldás: A megoldást a 4.7. ábra mutatja.



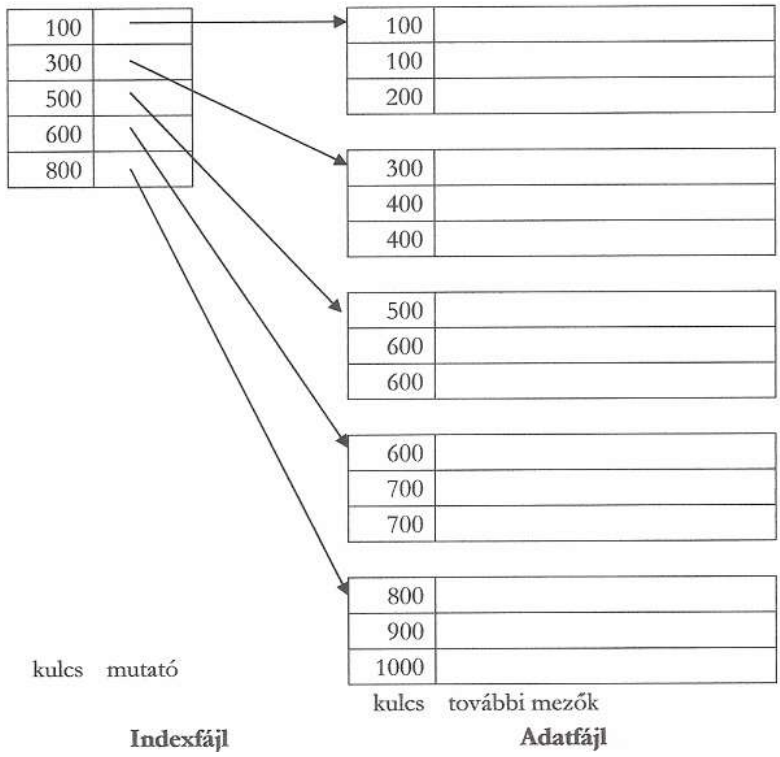
4.7. ábra. Sűrűindex ismétlődő kulcsokkal

Az indexfájlban a K indexkulcshoz olyan mutató tartozik, amely az első K értékkel rendelkező adatrekordra mutat. A többi azonos kulcsú rekord közvetlenül utána helyezkedik el. Az azonos kulcsú rekordok feldolgozásához addig olvassuk be a rekordokat, amíg a kulcs nagyobb nem lesz K-nál.

Ritka index

Példa: Az előző ábrán megadott adatfájlhoz készítsük el a ritka indexeket tartalmazó indexfájlt! Az adatblokkokban megengedett a kulcsérték ismétlődése. Az indexkulcsok az adatblokkok legkisebb keresési kulcsát tartalmazzák! Az indexblokkban továbbra is 5 rekord fér el.

Megoldás: A megoldást a 4.8**.** ábra mutatja.



4.8. ábra. Ritka index, ismétlődő kulcsokkal

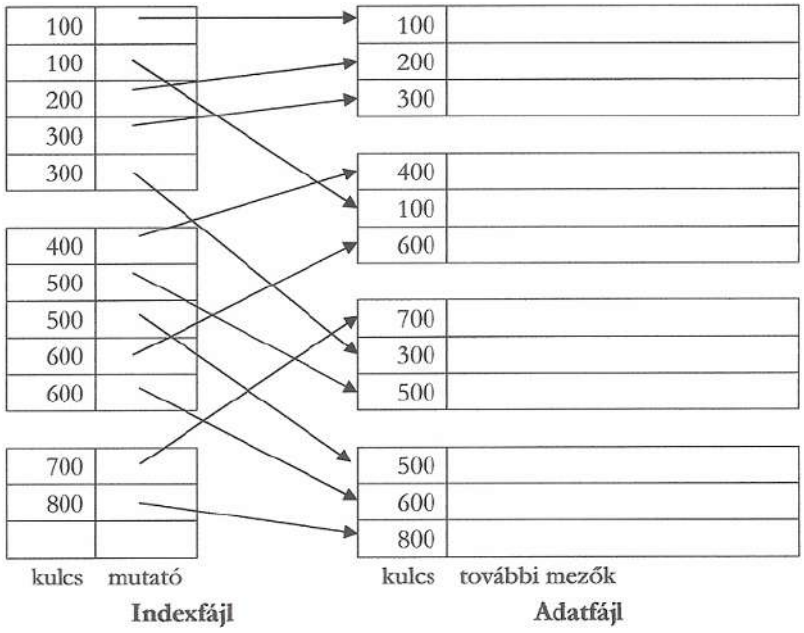
A K kulcsú adatrekordok megtalálásához meghatározzuk az (E_1, E_2) intervallumot az indexfájlban, amelyre $E_1 < K \leq E_2$. Először meghatározzuk E_2 –t, majd visszafelé haladva E_1 –t. Ha nincs olyan kulcs, amely kisebb K-nál, akkor E_1 az indexfájl első rekordja. Ezután az adatblokkok elérhetők az E_1 és E_2 közötti bejegyzések mutatói segítségével.

Másodlagos indexek

A másodlagos index sűrű index, amely általában tartalmaz ismétlődéseket. Az adatfájl nem rendezett a másodlagos index kulcsa szerint.

Példa: Az ábrán megadott adatfájlhoz készítsük el a másodlagos sűrű indexeket tartalmazó indexfájlt! Minden adatblokkban a legkisebb kulcsot vigyük az indexbe, az indexblokkban továbbra is 5 rekord fér el.

Megoldás: A megoldást a 4.9. ábra mutatja.

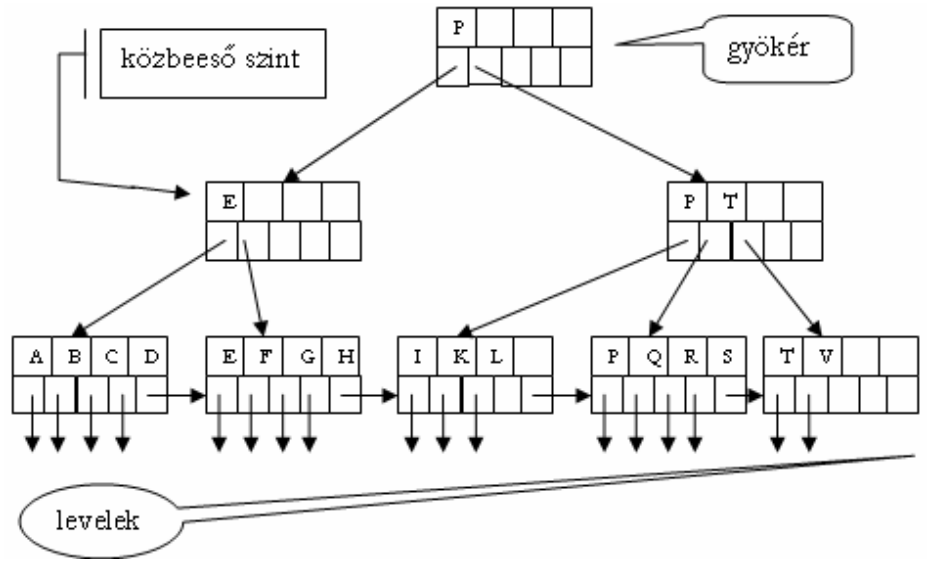


4.9. ábra. Másodlagos sűrű index

Adott kulcsú rekordok megtalálásához, az összes olyan blokkot be kell olvasni, amelyekre az indexkulcsok mutatnak.

4.3.2. B-fák

A B-fa indexblokkok faszerkezetű hierarchiát mutatnak (4.10. ábra). A *csomópontokban* indexblokkok találhatók, az élek az indexmutatók. A fa **kiegyensúlyozott**, ha valamennyi gyökértől levélig vezető út egyforma hosszú. Tipikusan három szint található egy B-fában: a *gyökér*, egy *közbeeső szint* és a *levelek* , de több szint is lehetséges.



4.10. ábra. 3-szintű kiegyensúlyozott B-fa

Legyen **n** a B-fa blokkjaiban *maximálisan elhelyezhető* indexkulcsok száma, ahol még **n + 1** mutató is található.

Példa: Tegyük fel, hogy a blokkjaink mérete 4096 bájt. A kulcsok legyenek 4 bájtos egész számok és a mutatók 8 bájtosak. Hány indexkulcsot (és mutatót) tudunk elhelyezni egy blokkban (a blokkfejrész mérete elhanyagolható)?

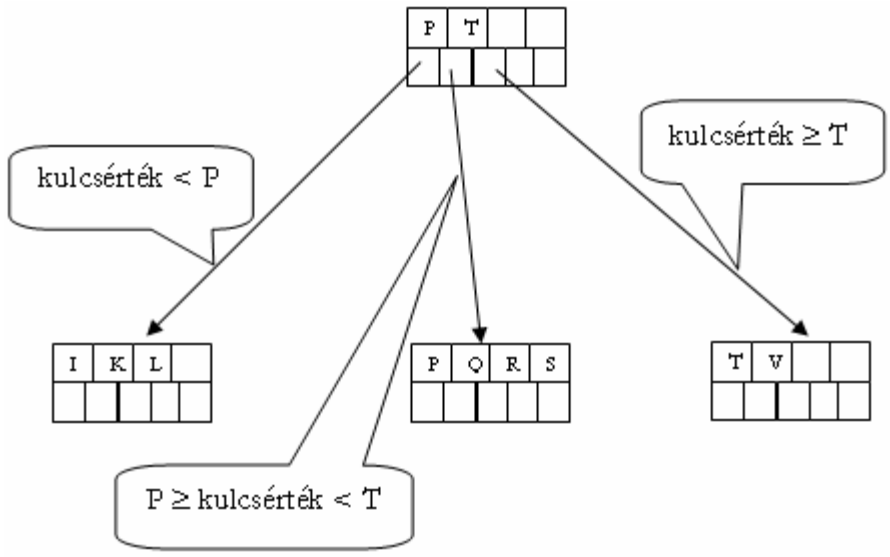
Megoldás: Ha a blokkokban nincsenek fejléc információk, akkor azt a legnagyobb egész **n** értéket keressük, amelyre $4n + 8(n + 1) \leq 4096$. Ez az érték az $n = 340$.

Az indexfájl karbantartása során az alábbi *megszorításokat* használjuk a B-fa indexblokkjaira:

- A **gyökérben** van legalább két mutató. A mutatók a B-fa következő szintjének blokkjaira mutatnak.
- A **levelekben** az utolsó mutató a *jobboldali levélblokkra* mutat. amely a soron következő nagyobb kulcsokat tartalmazza. A levélblokk többi mutatója *adatrekordra* mutat. Ezekből legalább **[(n + 1) / 2]**, azaz kb. 50%, használatban van.
- A **közbeeső szinteken** levő blokkokban a mutatók a B-fa következő szintjére mutatnak. Legalább **[(n + 1) / 2]** mutató használatban van. Legyen **i** mutató használatban, akkor **(i - 1)** kulcs van: K_1, K_2,..., K_{i-1}. Az első mutató a B-fa olyan részére mutat, ahol a K_1 kulcsnál kisebb kulcsokat tartalmazó rekordok találhatók. A második mutató a fa azon részére mutat, ahol azok a rekordok találhatók, amelyeknek a kulcsa nagyobb vagy egyenlő, mint K_1, de kisebb, mint K_2 és így tovább. Az **i**-dik mutató esetén azok a rekordok találhatók, amelyeknek kulcsa nagyobb vagy egyenlő, mint K_{i-1}.
- Minden szinten balról jobbra az indexblokkok kulcsai *nem csökkenő sorrendben* jelennek meg.

Az 4.10. ábra blokkjaiban 4 kulcsnak és 5 mutatónak van helye (n=4). A kulcsok az angol ABC nagybetűi. A megszorítások:

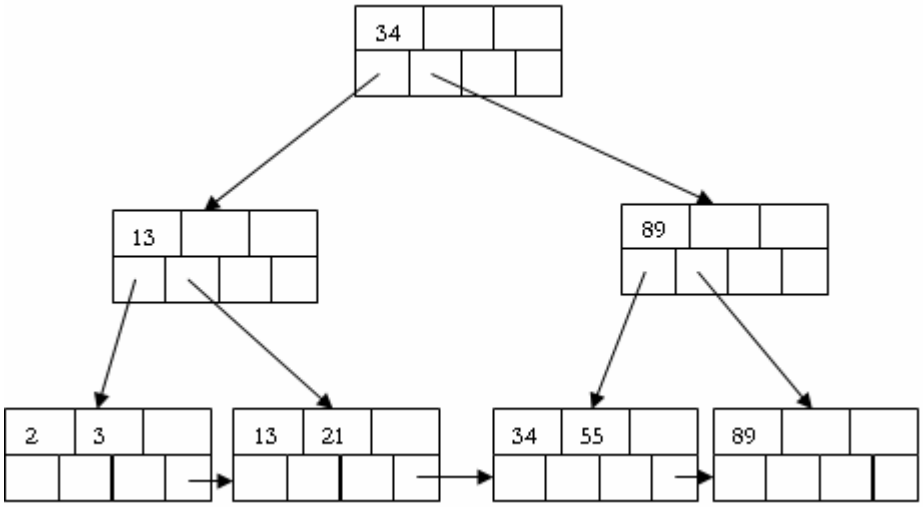
- A *levelekben* legalább 2 mutató ([(4+1)/2]=2) használatban van, amelyek adatrekordra mutatnak.
- A *közbeeső szinten* ugyancsak minimum 2 mutatónak (és 2 kulcsnak) kell lennie. A mutatók a 4.11**.** ábrának megfelelően mutatnak a levélszint blokkjaira.
- Szintenként a kulcsok ABC-re *növekvő* rendezettségben találhatók.



4.11. ábra. B-fa közbeeső szintű indexblokk értelmezése

Példa: Az adatfájl kulcsai a következő Fibonacci-számok közül kerülhetnek ki: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597. Az adatfájl jelenleg olyan rekordokból áll, melyekben a következő kulcsok vannak: 2, 3, 13, 21, 34, 55, 89. Készítsük el az aktuális adatfájlhoz tartozó 3-szintű B-fa indexet, ha egy indexblokkban maximum 3 kulcs helyezhető el. Kezdetben minden levélblokkban legyen 2 kulcs.

Megoldás: Először a levélszint blokkjait töltjük fel (4.12**.** ábra), majd a megszorításoknak megfelelően a magasabb szinteket is. A levélblokkok adatrekord mutatóit az ábrán nem tüntettük fel.



4.12. ábra. 3-szintű B-fa Fibonacci-számokkal

Keresés B-fában

Feltételezzük, hogy nincsenek ismétlődő kulcsok. Egy K kulcs keresését a gyökértől kezdjük, majd egy levélig kell lejutnunk. A keresési eljárás a következő:

- A *gyökérben* és *közbeeső szinteken* a megszorításoknak megfelelően kell vizsgálni, hogy melyik élen (mutatón) megyünk az alacsonyabb szintre.
- Ha lejutottunk egy *levélhez*, akkor megvizsgáljuk a kulcsait. Ha valamelyik kulcs megegyezik K-val, akkor a mutató alapján elérhetjük az adatrekordot. Ha nincs egyező kulcs, akkor a keresés sikertelen (nincs K-kulcsú adatrekord).

Megjegyzés: Ha egy adatfájlt a B-fa index alapján rendezetten akarunk bejárni, akkor a gyökértől elindulva a baloldali mutatókon lejutunk az első levélblokkhoz. Ezután a levélblokkokat kell bejárni balról jobbra haladva a mutatóknak megfelelően. A gyökérblokk címe a katalógusból olvasható ki.

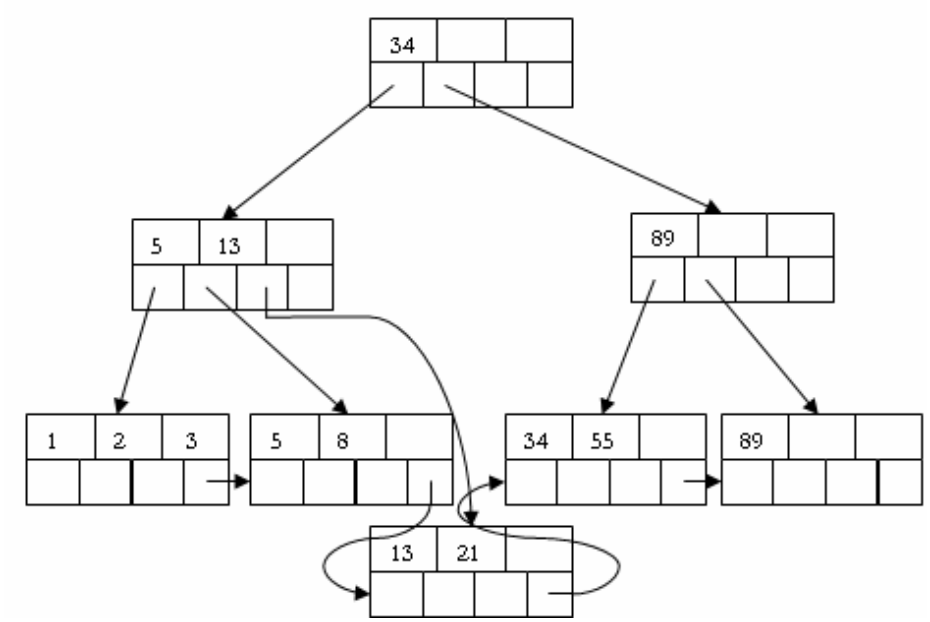
Beszúrás B-fába

A beszúrás lépései:

- A keresési algoritmus alapján helyet keresünk az új kulcs számára a rendezettségnek megfelelő levélben. Ha van szabad hely az adott blokkban, akkor elhelyezzük a kulcsot és a mutatót.
- Ha nincs hely, akkor kettévágjuk a levélblokkot, és szétosztjuk a kulcsokat a két csúcs között.
- Egy csúcs szétvágása miatt egy új kulcs-mutató párt kell beszúrni a felsőbb szinten. Ha van hely, elvégezzük a beszúrást, ha nincs, akkor szétvágjuk a szülő csúcsot és megyünk tovább fölfelé a fában.
- Ha a gyökérbe kell beszúrni és nincs hely, akkor szétvágjuk a gyökeret két csúcsra, és létrehozunk egy új gyökeret a következő szinten.

Példa: Szúrjuk be a 4.12**.** ábrán látható B-fába az 1, 5, 8 kulcsokat!

Megoldás: Az 1-es kulcs beszúrása az első levélblokkba történik, ezzel a blokk betelik. A magasabb szinteken nem kell módosítani. Az 5-ös kulcs a második levélblokk első bejegyzése lesz, amivel a második levélblokk is megtelik. A szülő blokkban a 13-mas kulcsot lecseréljük 5-re. A 8 beszúrása a második levélblokkba kerül (5 és 13 közé), de itt nincs hely, ezért „szétvágjuk" a blokkot. A 2 blokkba elosztjuk a kulcsokat, majd a szülő szinten is elvégezzük a módosítást a 4.13**.** ábrának megfelelően.



4.13. ábra. Kulcsok beszúrása

Törlés B-fából

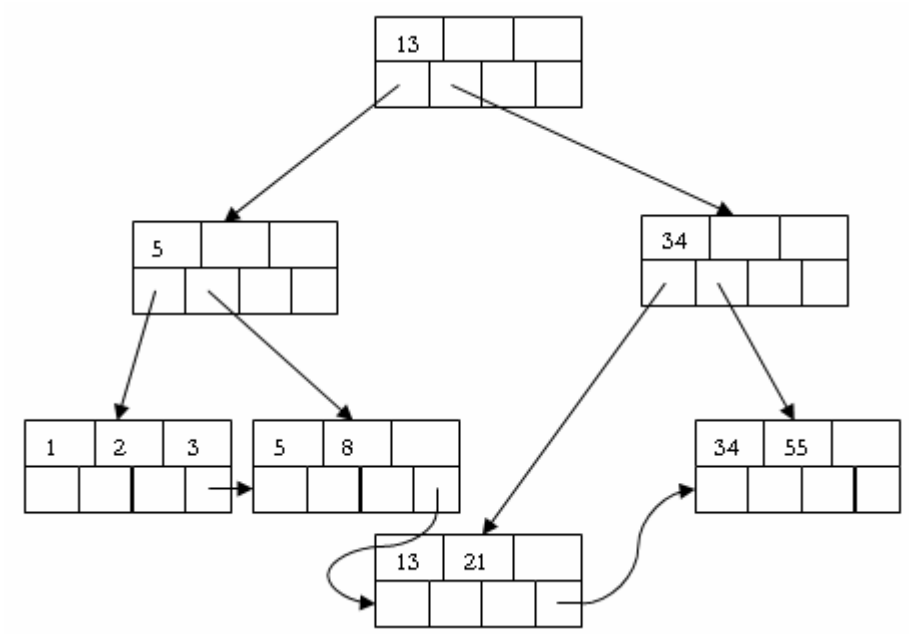
A törlés lépései:

- Töröljük az adott kulcsú rekordot az adatfájlból.
- Megkeressük a törlendő kulcsot a megfelelő levélben és töröljük a kulcs-mutató pár indexbejegyzést.
- Ha törlés után a B-fa csúcsa teljesíti a minimumfeltételt, akkor befejeztük a törlést.
- Ha nem, akkor kulcs-mutató párt kell áthelyezni a szomszédos csúcsokból. Ha nem lehet, akkor a két szomszédos csúcs összeolvasztható, az egyik indexblokk törölhető.
- Ezután a szülőben levő kulcsokat az új gyerekblokkokhoz kell igazítani és hasonlóan vizsgáljuk felfelé a szülőblokkokat.

Példa: Töröljük a 89-es kulcsot a 4.13. ábra B-fájából!

Megoldás: A kulcs az utolsó levélblokkban található. Miután kitöröljük a kulcsot, a hozzá tartozó mutatót és a megfelelő rekordot, üres lesz a

blokk. Töröljük a blokkot, majd szülő szinteken elvégezzük módosításokat. Az eredményül kapott B-fát a 4.14. ábrán láthatjuk.



4.14. ábra. Kulcs törlése B-fából

4.3.3. Tördelőtáblázatok

Egy **h**(K) *tördelőfüggvény* segítségével az argumentumként kapott **K** keresési kulcsot leképezzük a [0, B – 1] intervallumra. A *kosártömb* egy olyan tömb, amelynek indexei **0** és **B - 1** között vannak, és B számú láncolt blokk kezdő blokkjának címét tartalmazza. Azok a rekordok, amelyeket a **h** tördelőfüggvény azonos kosárba tördel, az adott kosárhoz tartozó blokkban vannak. Ha a kezdő blokk betelik, akkor egy *túlcsordulás blokkokból* álló láncot hozunk létre.

Bármely **i** ($0 \leq i \leq B-1$) kosár első blokkja megtalálható, adott **i** érték esetén:

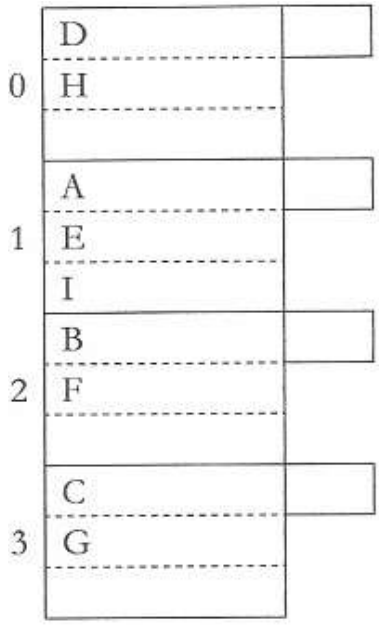
1. A memóriában lehet egy olyan tömbünk, amelyik a kosarak sorszámával indexelt és a blokkokra mutató mutatókból áll.
2. Valamennyi kosár első blokkját rögzített, egymás utáni lemezterületekre tesszük, így kiszámolható az **i** kosár első blokkja, adott **i** érték esetén.

A 4.15. ábra egy tördelőtáblázatot mutat. Feltételezzük, hogy egy blokkban három rekord fér el, a B értéke 4. Legyen a **h** tördelőfüggvény:

Asc(K) Modulo 4

A kulcsok az angol ABC nagybetűi. A tördelés eredménye:

h(D) = h(H) =0, h(A) = h(E) = h(I) =1, h(B) = h(F) =2, h(C) = h(G) =3.



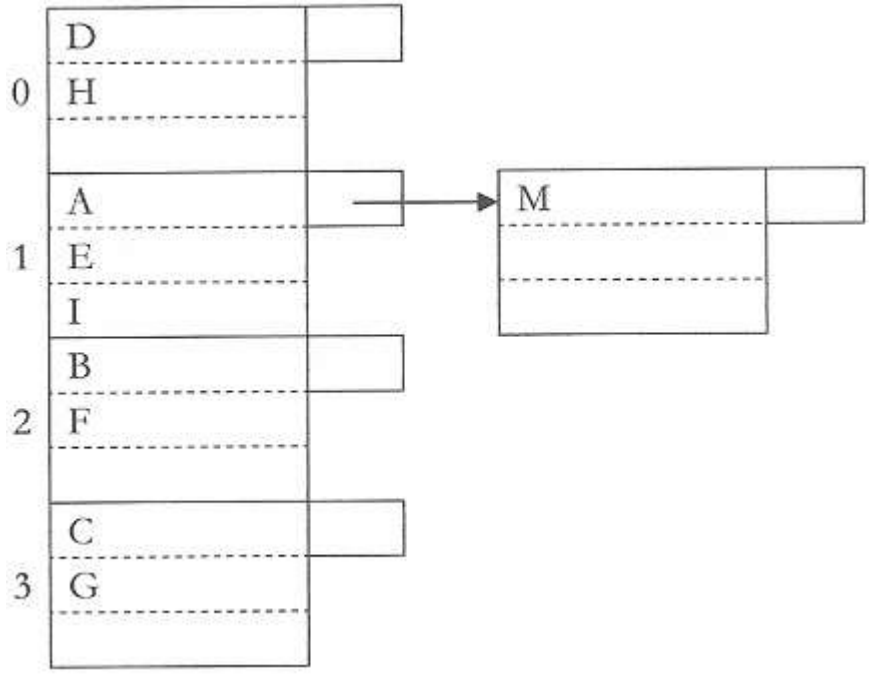
4.15. ábra. Tördelőtáblázat

Beszúrás tördelőtáblázatba

Ha egy **K** kulcsértékű új rekordot kell beszúrni, akkor kiszámoljuk a **h**(K)-t. Ha a **h**(K) jelzőszámú kosárban van szabad hely, akkor a rekordot beszúrjuk a kosárhoz tartozó blokkba, vagy ha az első blokkban nincs hely, akkor a kosárhoz tartozó lánc valamelyik túlcsordulás blokkjába. Ha a h(K) sorszámú kosárhoz tartozó lánc egyik blokkjában sincs szabad hely, akkor hozzáadunk a lánchoz egy újabb túlcsordulás blokkot, és ide szúrjuk be az új rekordot.

Példa: Adjuk hozzá az **M** kulcsértékű rekordot az előző ábrán látható tördelőtáblázathoz (**h**(M) = 1)!

Megoldás: Az új rekordot az 1-es kosárba kell helyeznünk. A kosárhoz tartozó blokk már betelt. Hozzáadunk egy új blokkot, és a kezdőblokkhoz láncoljuk. Az új rekordot az üres blokkban helyezzük el.



4.16. ábra. Beszúrás tördelőtáblázatba

Törlés tördelőtáblázatban

A **h**(K) sorszámú kosár blokkmutatóján elindulva megkeressük az összes **K** kulcsértékű rekordot és töröljük. Szükség esetén végrehajtjuk a blokkok átszervezését.

Példa: Töröljük az előző ábrán levő **E** kulcsértékű rekordot!

Megoldás: Az 1-es kosárhoz (h(E) = 1) tartozó kezdőblokkon kell elindulnunk. Töröljük az összes (jelenleg 1 db) E kulcsértékű rekordot. Átszervezhetjük a blokkláncot. A második blokk M kulcsú rekordját áttehetjük az E kulcsú rekord helyébe, így felszabadul a lánc második blokkja, amit törölhetünk.

4.4. Microsoft SQL Server 2005 fizikai adatbázis architektúra

Egy **SQL Server 2005** adatbázist kettő vagy több fizikai fájl tárol az adathordozón.

Az **SQL Server 2005** minden példánya rendelkezik öt rendszeradatbázissal (**master, resource, model, tempdb** és **msdb**), és egy vagy több felhasználói adatbázissal.

master

A *master* adatbázis tárolja az összes rendszerszintű információt. Kezeli a bejelentkezési azonosítókat, a konfigurációs beállításokat. Az adatbázis bejegyzést tartalmaz a többi adatbázisról, azok fájlhelyeiről. Tárolja az SQL Server inicializálási információit, mindig rendelkezik egy legfrissebb master másolat lehetőséggel.

resource

A *recource* adatbázis csak olvasható, tartalmazza az összes rendszer objektumot. Nem tartalmaz felhasználói adatot vagy felhasználói metaadatot.

tempdb

A *tempdb* tárolja az összes ideiglenes táblát és ideiglenes tárolt eljárást. Itt tárolódnak az egyéb ideiglenes tárolású információk is, például az SQL Server által használt munkatáblák. Az SQL Server indításával az adatbázis mindig újraépül az adatbázis egy tiszta másolatával.

model

A *model* adatbázis a rendszerben létrehozott adatbázisok sablonjaként működik. Amikor a CREATE DATABASE utasítást kiadjuk, kezdetben az adatbázis a model adatbázis tartalmának másolatából áll, ezután az új adatbázis maradék része üres lapokkal töltődik fel. Mivel a tempdb minden SQL Server indításkor újra létrejön, a model adatbázisnak mindig léteznie kell.

msdb

Az msdb adatbázist az **SQL Server Agent** használja feladatok, munkák ütemezésére, a műveletek tárolására.

Egy adatbázishoz egy időben több felhasználó is csatlakozhat. Amikor kapcsolatot létesítünk az **SQL Server 2005** egy példányával, akkor meg kell adni egy adatbázis nevét. Ezt nevezzük *aktuális* (current) adatbázisnak.

Általában a rendszeradminisztrátor által meghatározott adatbázishoz kapcsolódunk. Ez az *alapértelmezett* (default) adatbázis.

4.4.1. Lapok (Pages) és kiterjesztések (Extents)

Az SQL Server 2005 tárolási alapegysége a *lap*. Egy lap mérete 8 Kbájt, azaz 1 Mbájt 128 lapot tartalmaz. A *kiterjesztések* olyan alapegységek, amelyekben tárolási hely van adattáblák és indexek számára. *Egy kiterjesztés 8 folytatólagos lapból áll*, azaz 64 Kbájt. Ennek megfelelően az adatbázisok megabájtonként 16 kiterjesztést tartalmaznak.

Minden lap 96 bájtos *fejrésszel* kezdődik, amely rendszerinformációkat tartalmaz, úgy, mint a lap típusát, szabad hely méretét a lapon, és a lapot birtokló objektum objektumazonosítóját.

Az alábbi táblázat az adatbázis adatfájljaiban előforduló nyolcféle laptípust sorolja fel:

Laptípus	Tartalom
Adat (Data)	Adatsorok minden adattípusra, kivéve a nagy objektumok (LOB) adattípusai: **text**, **ntext**, **nvarchar(max)**, **varchar(max)**, **varbinary(max)**, **XML** (ha a **text in row** beállítás: ON) és **image**.
Index	Index bejegyzések.
Szöveg és kép (Text/Image)	A LOB adattípusok, valamint a változó hosszúságú oszlopok adatai (**varchar**, **nvarchar**, **varbinary** és **sql_variant**), ha a rekordhossz nagyobb 8K-nál.
Globális elhelyezési táblázat (Global Allocation Map, GAM), Megosztott globális elhelyezési táblázat (Shared Global Allocation Map, SGAM)	Információ az extentek kiosztásáról.

4.1. táblázat.

Táblázat folytatás

Laptípus	Tartalom
Lap szabadhely (Page Free Space, PFS)	Információ a lapokon lehetséges üres helyekről.
Index elhelyezési táblázat (Index Allocation Map)	Információ egy tábla vagy index által használt kiterjesztésekről.
Nagytömegű módosítások táblázat (Bulk Changed Map, BCM)	Információ azokról a kiterjesztésekről, amelyeket módosítottak a nagy, összetett műveletek az utolsó BACKUP LOG utasítás végrehajtása óta.
Különbözeti változások táblázat (Differential Changed Map, DCM)	Információ azokról a kiterjesztésekről, amelyek módosultak az utolsó BACKUP DATABASE utasítás végrehajtása óta.

Az adatsorok egymás után helyezkednek el a lapokon, közvetlenül a fejrész után kezdve. A *sor eltolási táblázat* a lap végén kezdődik. A sor eltolási táblázat bejegyzést tartalmaz a lapon elhelyezkedő minden sorra. A bejegyzés a sor kezdetét tartalmazza bájtban mérve a lap elejétől. A sor eltolási táblázat fordított sorrendben tartalmazza a bejegyzéseket, mint ahogyan a sorok a lapon elhelyezkednek.

A sorok nem hidalhatnak át lapokat, ezért egy sorban maximum 8060 bájtnyi adat tárolható, változó hosszúságú adatok (text, ntext, varchar(max), stb.) esetén túlcsotdulási területre kerülhetnek adatok, ilyenkor 24 bájtos mutató tárolódik a kezdő rekordban.

Kezdetben a kiterjesztés összes lapját általában nem osztja ki az **SQL Server 2005** a kisméretű adattábláknak. Kétféle kiterjesztést különböztetünk meg:

- Az **egyfajta** (uniform) kiterjesztések egy objektumhoz tartoznak, a 8 lapot csak a tulajdonos objektum használja.
- A **vegyes** (mixed) kiterjesztésen maximum 8 objektum osztozhat.

Új adattábla vagy index általában vegyes kiterjesztésen lévő lapokat kap. Amikor a tábla, vagy index mérete nő, és eléri a 8 lapot, a kiterjesztések átváltanak uniform típusúra.

4.4.2. Fizikai adatbázis fájlok

A **Microsoft SQL Server 2005** adatbázis általában operációsrendszerbeli fájlokban tárolódik. Az *adat* és *napló* információk soha nem keverednek egy fájlban, az adott fájlt csak egy adatbázis használhatja.

Az SQL Server 2005 adatbázisok háromféle fájt használhatnak:

- **Elsődleges** (Primary) adatfájlok
  Az elsődleges adatfájl az adatbázis kezdőpontja, amely azután egyéb fájlokra mutat, amelyek az adatbázishoz tartoznak. Minden adatbázishoz pontosan egy elsődleges adatfájl tartozik. Az ajánlott fájlkiterjesztés: **.mdf**.
- **Másodlagos** (Secondary) adatfájlok
  Az elsődleges adatfájlon kívüli összes adatfájlt magában foglalja. Egy adatbázishoz több másodlagos adatfájl tartozhat. Az ajánlott fájlkiterjesztés: **.ndf**.
- **Napló** (Log) fájlok
  A naplófájlok minden olyan információt tárolnak, amely szükséges az adatbázis helyreállításához. Minden adatbázishoz kötelezően tartozik legalább egy naplófájl. Ajánlott fájlkiterjesztés: **.ldf**.

Az SQL Server 2005 fájloknak kettő nevük van:

- **Logikai** fájlnév, amelyet az SQL utasításokban használunk, a fájlra történő hivatkozásokban. A logikai fájlnév meg kell, hogy feleljen SQL Server névadási szabályainak, és egyedi kell, hogy legyen az adatbázisban.
- **Fizikai** fájlnév a fájl fizikai fájlneve az operációs rendszerben.

SQL Server adat és napló fájlok elhelyezhetők FAT vagy NTFS típusú fájlrendszerben, de nem tárolhatók tömörített fájlrendszerekben.

Az adatfájl lapjait 0-tól kezdődően folytonosan sorszámozza az SQL Server. Minden fájl rendelkezik egy **fájlazonosító** számmal. Egy adatbázisban a lapot a fájlazonosító és a lapszám együtt azonosítja egyértelműen.

Minden fájlban az első lap a **fájl fejrész** (header) lap, amely a fájl attribútumairól tartalmaz információkat. A fájl elején levő néhány egyéb lap rendszerinformációkat tartalmaz (pl. elhelyezési táblázatok). Az elsődleges adatfájlban és az első napló fájlban is a tárolt rendszerlapok egyike az

adatbázis behúzó (boot) lapja. Ez a lap az adatbázis attribútumairól tartalmaz információkat.

Az SQL Server 2005 fájlok automatikusan nőhetnek az eredetileg megadott méretüktől. Létrehozáskor megadhatjuk a növekedési értéket. Amikor a fájl betelik, a rendszer automatikusan megnöveli a méretét a növekedési értékkel. Ha egy fájlcsoportban több fájl van, akkor a növelés csak az összes fájl betelte után történik meg.

Minden fájlhoz meghatározhatunk egy maximális méretet. Ha nincs megadva maximális méret, akkor a növekedés korlátja csak az adathordozó fizikai mérete. Ha nem tudjuk felügyelni, figyelni a rendelkezésre álló szabad helyet, akkor célszerű nem megadni a maximális méretet.

4.4.3. Adatbázis fájlcsoportok (Filegroups)

Az adatbázis fájlokat fájlcsoportokba szervezhetjük elhelyezési és adminisztrációs (mentés, visszaállítás) okokból. Bizonyos rendszerek képesek megnövelni teljesítményüket, ha az adatok és indexek elhelyezését felügyelhetik a meghatározott lemezes meghajtókon. A fájlcsoportok ezen folyamatokat segíthetik. A rendszer adminisztrátor fájlcsoportokat hozhat létre az egyes meghajtókon. Ezután hozzárendelhetők táblák, indexek, vagy a **LOB** adatok egy táblából a megadott fájlcsoportokhoz.

Egy fájl nem lehet tagja több fájlcsoportnak. Táblák, indexek, text, ntext, és image adatok rendelhetők hozzá egy fájlcsoporthoz. Ebben az esetben minden lapjuk az adott fájlcsoporthoz tartozik.

A naplófájlok nem lehetnek fájlcsoportok részei. A naplóhelyet mindig külön kezeli a rend-szer az adathelytől.

Kétféle fájlcsoport létezik:

- **Elsődleges** (Primary)
  Az elsődleges fájlcsoport tartalmazza az elsődleges adatfájlt és minden olyan egyéb fájlt, amelyet nem rendeltünk hozzá más fájlcsoporthoz. A rendszertáblák lapjai mindig az elsődleges fájlcsoportban tárolódnak.
- **Felhasználói** (User-defined))
  A felhasználói fájlcsoportok a FILEGROUP kulcsszóval definiálhatók a CREATE DATABASE vagy ALTER DATABASE utasításokban.

Minden adatbázisban egy fájlcsoport alapértelmezett fájlcsoportként funkcionál. Ha olyan táblához, vagy indexhez helyez el lapot az SQL Server, amelyhez létrehozáskor nem adtunk meg fájlcsoportot, akkor az alapértelmezett fájlcsoportból veszi a lapot.

Az **SQL Server 2005** adatbázisok leválaszthatók a szerverről és újra csatlakoztathatók egy másik, vagy ugyanazon szerverhez.

4.5. Ellenőrző kérdések

1. Hogyan történik a következő adatelemek tárolása: számok, szövegek, dátumok, időpontok?
2. Hogyan történik a következő adatelemek tárolása: bitsorozatok, felsorolások, BLOB-ok?
3. Hogyan történik a mezők, rekordok tárolása?
4. Ismertesse a blokkok felépítését!
5. Ismertesse a rekord beszúrás karbantartási művelet vázlatos végrehajtását!
6. Ismertesse a rekord módosítás karbantartási művelet vázlatos végrehajtását!
7. Ismertesse a rekord törlés karbantartási művelet vázlatos végrehajtását!
8. Ismertesse a következő fogalmakat: sűrű indexek, ritka indexek.
9. Vázolja sűrű index létrehozását rendezett szekvenciális adatfájlon!
10. Vázolja ritka index létrehozását rendezett szekvenciális adatfájlon!
11. Vázolja sűrű index létrehozását rendezett szekvenciális adatfájlon ismétlődő kulcsokkal!
12. Vázolja ritka index létrehozását rendezett szekvenciális adatfájlon ismétlődő kulcsokkal!
13. Vázolja egy 3-szintű kiegyensúlyozott B-fa blokkszerkezetét!
14. Adja meg a B-fa blokkjaira alkalmazott megszorításokat!
15. Hogyan hajtjuk végre a keresés műveletét B-fában!
16. Ismertesse az indexrekord beszúrás művelet vázlatos végrehajtását B-fa esetén!
17. Ismertesse az indexrekord törlés művelet vázlatos végrehajtását B-fa esetén!
18. Ismertesse a tördelőtáblázatok használatát!
19. Ismertesse az SQL Sever 2005 rendszeradatbázisait!
20. Ismertesse az SQL Sever 2005 laptípusait!
21. Ismertesse az SQL Sever 2005 fizikai fájltípusait!
22. Ismertesse az SQL Sever 2005 fájlcsoportok használatát!

5. AZ SQL2 NYELV

5.1. Áttekintés

5.1.1. Szintaktikai elemek

Az SQL nyelv alapegységeket (token) épít fel, amiket elválasztó jelek (szóköz, kocsivissza (CR), soremelés (LF), pontosvessző, stb.) határolnak. Az alapegységek tovább csoportosíthatók az alábbiak szerint:

- kulcsszók (kis-/nagybetű),
- azonosítók,
- operátorok (műveleti jelek),
- literálok (szám, dátum, karakter adattípusú értékek)
- SQL utasítást lezáró elválasztó jelek

Az SQL utasítást egy elválasztó jel zárja le. Általában az alábbi lehetőségeket használják az egyes SQL megvalósítások:

- pontosvessző,
- kocsivissza és/vagy soremelés,
- kitüntetett kulcsszó (pl.: GO).

5.1.2. Azonosítók képzési szabályai

Az azonosítók objektumok, programok, eljárások, változók, stb. szimbolikus nevei. Az SQL megköveteli, hogy betűvel kezdődjön egy azonosító, folytatásként betű, számjegy, néhány speciális karakter ($, @, _ , stb.) fordulhat elő. A legnagyobb hossz általában 30 karakter, de nagyon eltérő hosszak is előfordulhatnak az egyes SQL megvalósításokban.

5.1.3. Kifejezések és műveletek

A kifejezés tartalmazhat adatmezőt, literálokat (adattípusnak megfelelő konstansok), operátorokat, SQL kulcsszavakat, oszlopfüggvényeket, általános beépített függvényeket, valamint speciális esetben programnyelvi változókat.

Az SQL az alapvető adattípusoknak megfelelő műveleteket (operátorokat) támogatja. Tehát megkülönböztetünk:

- numerikus,
- karakteres,

- dátum-, idő jellegű,
- bináris- és logikai

műveleteket.

Az adattípustól függetlenül használhatjuk az ún. théta operátorokat: <, <=, >, >=, =, # vagy <>. Ezeket tagadhatjuk a ! jellel, vagy a NOT (logikai nem) művelettel.

A kifejezések kiértékelése a "szokásos" módon történik az alábbi sorrendben:

- zárójelek
- előjelek
- hatványozás
- szorzás, osztás, maradékképzés
- összeadás, kivonás
- numerikus, vagy arra visszavezethető relációk
- logikai NEM művelet (NOT vagy !)
- logikai ÉS művelet (AND)
- logikai VAGY művelet (OR)
- dátumokon, szöveges mezőkön végezhető mintakeresések (LIKE), átalakítások

5.1.4. NULL-értékek

Bizonyos esetekben előfordul, hogy az adattábla adott helyére (mezőjébe) nem tudunk megadni adatot. Ilyenkor az adatnélküli mező határozatlan tartalommal, NULL-értékkel rendelkezik. NULL-értéket tartalmazó kifejezések eredménye határozatlan (maybe, ?).

Logikai igazságtáblák

Az alábbi ábra a három alapvető logikai művelet ún. **igazságtáblázatát** mutatják (T-True, F-False).

AND	T	F	?
T	T	F	?
F	F	F	F
?	?	F	?

OR	T	F	?
T	T	T	T
F	T	F	?
?	T	?	?

NOT	
T	F
F	T
?	?

5.1. ábra. Logikai igazságtáblák

A táblázatok segítséget nyújthatnak összetett logikai kifejezések kiértékelésében, amennyiben NULL-érték is előfordulhat.

5.1.5. Objektumok

Röviden ismertetjük az SQL2 különböző objektumait. Az "objektum" szót, abban az értelemben használjuk, hogy szimbolikus névvel azonosíthatók, kezelhetők.

Adatséma

Az adatbázissémák olyan teljes, vagy részadatbázisok leírásai, amelyekbe tartozó objektumokat csak adott felhasználók, jelszók ismeretében, érhetnek el. Tehát az adatsémák definiálják, hogy adott felhasználók az adatbázis mely részleteit, milyen jogosultságokkal érthetik el.

Adattípus

Az SQL öt alapvető adattípust különböztet meg. Ezek a következők:

Adattípus	Leírás
számszerű	Olyan adatok tárolására szolgál, amelyekkel numerikus műveleteket végezhetünk.
szöveges	Tetszőleges szöveges információ tárolására szolgál.
dátum (idő) jellegű	Dátum és/vagy idő jellegű adatokat tároló adattípus.
bináris vagy logikai	Kétértékű adatok (True/False, Igen/Nem, stb.) tárolását végzi.
nyers, szerkezet nélküli	A rendszer számára közömbös a szerkezetük.

5.1. táblázat.

Az alábbi táblázat a fontosabb, általánosan használt, adattípusokat tartalmazza:

Adattípus	**Leírás**
BIT(n)	Rögzített hosszúságú bitsorozat.
BIT VARYING(n)	Változó hosszúságú bitsorozat.
SMALLINT INTEGER LONG	Egész számok.
DECIMAL(p,[s])	Fixpontos valós szám (p az összes jegyek, s a tizedes jegyek száma)
REAL, FLOAT DOUBLE PRECISION	Lebegőpontos szám.
DATE, TIME DATETIME TIMESTAMP INTERVAL	Dátum-, idő típusok.
CHAR(n)	Rögzített hosszúságú karaktersorozat.
CHAR VARYING(n)	Változó hosszúságú karaktersorozat.
TEXT	Hosszú szöveg.
IMAGE	Kép, videó.

5.2. táblázat.

Az egyes adattípusnak megfelelő konstansok használatát az alábbi táblázat mutatja:

Konstans adattípus	**Használat**
Számok	-4556; 231.67; 3.14E-5
Bitsorozat	B'01001'
Logikai	True; B'1'; False; B'0'
Szöveg	'szöveg'
Dátum	'2005-08-20'
Idő	'08:33:45'

5.3. táblázat.

Jelkészlet

A rendszer által kezelt jelek halmaza, operációsrendszer függő. A leggyakrabban használatos jelkészletek:

- ASCII
- EBCDIC

Jelsorrend

Adott karakterkészleten belül többféle rendezési sorrend definiálható, figyelembe véve a nemzeti karakterek rendezési szabályait is.

Jelkészlet leképezés

Tetszőleges karakterkészlet 1:1 értékű leképezése egy másikra. Lehetővé teszi régebbi adatbázisok adatainak új jelkészlettel történő kezelését.

Oszloptípus (DOMAIN)

Az alap adattípusokra építve definiálhatunk saját adattípust, melyet az adattábla mezőinek adattípus definíciójában szintén használhatunk. Ha módosítunk az oszloptípus definíción, akkor az automatikusan tükröződik (öröklődik) minden olyan tábla szerkezetében, ahol felhasználtuk mező adattípus definiálására.

Az oszloptípus definiálásakor az adattípus, méret jellemzők mellett megadhatunk alapértelmezett értéket, ellenőrzési szabályt. Az alapértelmezett érték automatikusan bekerül a tábla újonnan felvitt sorába. Az adatmódosítást vizsgálja az ellenőrzési szabály, a tiltott módosítást nem engedi bevinni a mezőbe.

Adattábla (TABLE)

Az adatbázis alapvető objektuma. Táblázatos formában tárolja az egyedtípus előfordulásokat. A táblázat oszlopai (mezői) az egyedhez definiált tulajdonságtípusok.

Oszloponként legalább meg kell adni a nevet, adattípust, méretet. Ezen kívül az SQL2 további mezőjellemzőket (indexelt, NULL-érték használat, alapértelmezett érték (DEFAULT), adatérték-szabály (CONSTRAINT), stb.) is támogat.

Nézettábla (VIEW)

Adattáblákra, korábban definiált nézettáblákra alapozva egy lekérdezést (SELECT) definiálunk. Csak a definíció kerül tárolásra. Amikor a nézettáblára hivatkozunk, akkor a rendszer a definíció alapján előállítja a lekérdezést. Ezután a lekérdezés eredménytáblájával dolgozhatunk. Lehetséges a nézettábla alapján az adattáblában történő módosítás is. Az adatséma jellemző objektumai a nézetek.

Indextábla (INDEX)

Adattáblára alapozva definiálhatunk indexeket. Az indextábla az indexkulcsokat (kimásolva az adattáblából) és a kulcshoz tartozó sor (rekord) fizikai címét tartalmazza rendezett formában, így a kulcsszerinti keresés nagyon gyors.

A rendezettség iránya lehet növekvő, csökkenő. Az indexkulcs lehet összetett, több mező együtt adja az indexkulcsot. Egyedi index esetén egy tetszőleges kulcsérték csak egyszer fordul elő. Adattáblánként egy Primary Key (elsődleges kulcs, egyedi index) definiálható, amely a táblák közötti kapcsolatok létrehozásánál játszik fontos szerepet. A primary key index definiálása az adattábla szerkezetének megadásakor, vagy módosításakor történik. A rendszer automatikusan kezeli az indextáblákat. Ha egy rendezettségre igény jelentkezik, akkor megnézi, hogy van-e olyan indexe. Ha igen, akkor használja, ha nem, akkor elvégzi a szükséges rendezést. Az adattáblával együtt az indextáblák automatikusan karbantartódnak.

Adatérték-szabály

Az oszloptípus és az adattábla-szerkezet létrehozásakor megszorításokat (CONSTRAINT), ellenőrzési szabályt adhatunk meg. Az adattábla szerkezet megadásakor lehet ún. táblaszintű ellenőrzési szabályt is megadni, ami-kor az ellenőrzés több mezőre is kiterjed. Az ellenőrzési szabály logikai típusú kifejezés megfelelő szintaktikával. Ha a kifejezés kiértékelése hamis értéket ad, akkor az adott mezőbe, vagy táblasorba nem engedi a rendszer bevinni a hibás értéket, értékeket.

Hivatkozási függőségi szabály

Logikai kapcsolatban (szülő-gyerek) álló táblákra hozhatunk létre hivatkozási függőségi szabályt (Referential Integrity). A két tábla között a kapcsolatot közös tulajdonságú mezők biztosítják, indexeken keresztül. A szülő

tábla indexe *elsődleges kulcs* (primary key), a gyerek tábla indexe az *idegen kulcs* (foreign key).

Az alap adat karbantartási funkciókat (felvitel, módosítás, törlés) korlátozhatjuk (restriction), ún. kaszkádolt végrehajtásokat állíthatunk be, illetve szabadon (függetlenül a két táblában) engedhetjük a karbantartások elvégzését.

Az alábbi táblázat összefoglalja a hivatkozási épség megőrzésének szabályait:

Karbantartás	**Szülő tábla**	**Gyerek tábla**
Felvitel	Szabadon vihetünk fel rekordot.	Csak létező szülőhöz (a kapcsolómező értéke előfordul a szülő táblában) lehet felvinni gyerek rekordot.
Módosítás	A kapcsolatot biztosító mező módosítását tiltjuk, vagy kaszkádolt módosítást hajthatunk végre, azaz a kapcsolódó gyerekrekordok kulcsmezőjét automatikusan módosítja a rendszer.	A kapcsolatot biztosító kulcsmező módosítását tilthatjuk, vagy csak létező szülőbeli kulcsértékre módosíthatunk.
Törlés	Ha van kapcsolódó gyerek rekord, akkor tiltjuk a törlést (ilyenkor először törölhetjük a kapcsolódó gyerekrekordokat), vagy kaszkádolt törlést hajtatunk végre, azaz a kapcsolódó gyerekrekordokat automatikusan töröltetjük.	A kapcsolódó gyerekrekordokat szabadon törölhetjük.

5.4. táblázat.

5.1.6. Utasítások

Az alábbi táblázat csoportokba gyűjtve felsorolja az alapvető SQL utasításokat:

Csoport	Utasítás
adatdefiníciós	CREATE, ALTER, DROP
adatkezelési	SELECT, INSERT, UPDATE, DELETE
adatbiztonsági	GRANT, REVOKE
egyéb	LOAD...

5.5. táblázat.

Az utasítások megadásakor a következő jelöléseket használjuk:

Jelölés	Leírás
[ ]	Opcionális, nem kötelezően megadandó elem vagy utasításrész.
< >	Kötelezően megadandó elem vagy utasításrész.
…	Tetszés szerint ismételhető elem vagy utasításrész.
|	A megadott jellel elválasztott elemek közül egy választható.
{ }	Az elemek közül egyet kötelező megadni.

5.6. táblázat.

5.1.7. Katalógus

A katalógus az adatbázishoz tartozik, a rendszer automatikusan kezeli. Tartalmazza az ún. metaadatokat hasonló szerkezetben, mint a többi adatot. Ezek lehetnek:

- objektumok leírása (név, jellemzők (pl. adatérték-szabály, alapértelmezett érték, stb.)),
- felhasználók azonosítói (név, jelszó),
- hozzáférési jogok,
- eljárásjellegű elemek (modulok, tárolt eljárások, triggerek).

A modul valamilyen programozási nyelven megírt, egy vagy több eljárást (programkód egységet) foglal magában. A tárolt eljárások olyan speciális eljárások, amiket az adatbázisban (a katalógusban) tárolunk. Írásukhoz általában ún. kibővített SQL utasítások állnak rendelkezésünkre, amelyekkel már vezérlési szerkezetek (elágazás, ciklus) is megvalósíthatók. Az eljá-

rásokat az alkalmazások megfelelő szintaktikával hívhatják, paramétereket adhatnak át. A futás végrehajtását, felügyeletét az adatbázist tároló számítógép végzi. A tárolt eljárás közvetlenül nem épül be az alkalmazásba.

A triggerek különleges eljárások, amelyek feltételek alapján kerülnek végrehajtásra pl. adatbevitel, módosítás, törlés előtt vagy után. Tehát ezek közvetlenül nem hívhatók, az adott karbantartási esemény aktivizálja végrehajtásukat. A hivatkozási épség megőrzését is sok esetben triggerek segítségével oldják meg.

5.2. Microsoft SQL Server 2005 logikai adatbázis architektúra

5.2.1. Principálok (Principals)

A principálok olyan egyedek, akik kérhetnek SQL Server erőforrást. Hierarchiába szervezhetők, mindegyik rendelkezik biztonsági azonosítóval (SID). Lehetnek:

- Windows-szintű principálok:
  a. Windows tartományi bejelentkezési azonosító (Domain Login)
  b. Windows helyi bejelentkezési azonosító (Local Login)
- SQL server-szintű principál:
  SQL Server bejelentkezési azonosító (Login)
- Adatbázis-szintű principálok:
  c. Adatbázis felhasználó (User)
  d. Adatbázis szerep (Database Role)
  e. Alkalmazás szerep (Application Role)

Az SQL Server **sa** bejelentkezési azonosító (rendszer adminisztrátor), szerverszintű principál, telepítéskor jön létre. Minden adatbázis felhasználó használhatja a **public** adatbázis szerepet. Minden adatbázis tartalmaz két felhasználót: INFORMATION_SCHEMA és **sys**. Ezek nem principálok, nem módosíthatók, nem törölhetők.

5.2.2. Felhasználók (Users)

A *felhasználói azonosító* (ID) az adatbázisban azonosít egy felhasználót. Minden engedély és objektumtulajdonlás az adatbázisban a felhasználói fiók által vezérelt. Felhasználói azonosítót a **db_owner** rögzített adatbázis szerep (lásd később) valamely tagja definiálhat.

Önmagában a felhasználói azonosító nem ad engedélyeket az adatbázisbeli objektumok kezeléséhez. A belépési azonosítót (login ID) össze kell kötni egy felhasználói azonosítóval (user ID) minden adatbázisban mielőtt bárki is csatlakozna a bejelentkezési azonosítóval, hogy kezelhesse az adatbázis objektumait. Ha a belépési azonosító nem kötődik közvetlenül felhasználóhoz, akkor a belépő a **vendég** (guest) felhasználó jogaival kezelheti az adatbázist. Ha az adatbázis nem rendelkezik vendég felhasználói fiókkal, akkor a belépő nem tudja kezelni az adatbázist anélkül, hogy a bejelentkezési azonosítót ne kapcsolnánk össze egy létező felhasználói névvel.

Az **sa** bejelentkezési azonosító használható a speciális **dbo** (adatbázis tulajdonos) felhasználói azonosítóhoz minden adatbázisban.

5.2.3. Szerepek (Roles)

A szerepek (szerepkörök) a felhasználókat összefogják egy olyan egységbe (csoportba), amit a jogosultságok kiosztásánál használunk. A szerepköröknek engedélyeket (jogokat) adhatunk, tilthatunk, vagy visszavonhatunk. Egy felhasználó több szerepkörhöz is tartozhat.

A telepítés során az SQL Server 2005 néhány rögzített szerepkört definiál. Ezekhez felhasználókat adhatunk adminisztrálási engedélyekkel. Az alábbi táblázat a *szerverszintű* rögzített szerepköröket tartalmazza:

Rögzített szerepek	Leírás
sysadmin	Bármilyen tevékenységet végrehajthat az SQL Server-en.
serveradmin	Beállíthatja a szerverre vonatkozó konfigurációkat, lekapcsolhatja a szervert.

5.7. táblázat.

Táblázat folytatás

Rögzített szerepek	Leírás
setupadmin	Kezelheti a csatolt szervereket és futtathat néhány rendszer tárolt eljárást.
securityadmin	Kezelheti a belépéseket, kezelheti a GRANT, DENY és REVOKE szerver-szintű engedélyeket. Olvashatja a hibanaplókat és módosíthat jelszavakat.
processadmin	Kezelheti (megszakíthatja) az SQL Serveren futó processzeket.
dbcreator	Létrehozhat, módosíthat, törölhet és visszaállíthat adatbázist.
diskadmin	Kezelheti a diszk fájlokat.
bulkadmin	Végrehajthatja a BULK INSERT utasítást.

Az **sp_helpsrvrole** rendszer tárolt eljárás segítségével kilistázhatjuk a rögzített szerver szerepeket, míg az **sp_srvrolepermission** tárolt eljárás a szerepenkénti engedélyeket listázza.

Minden adatbázis rögzített *adatbázis* szerepeket tartalmaz.

Rögzített adatbázis szerepkörök	Leírás
db_owner	Minden engedéllyel rendelkezik az adatbázisban.
db_accessadmin	Felvihet, törölhet felhasználói azonosítókat.
db_securityadmin	Kezelheti az összes engedélyt, objektumtulajdonlásokat, szerepeket és szereptagságokat.
db_ddladmin	Végrehajthatja az összes DDL utasítást, de nem hajthatja végre a GRANT, REVOKE, vagy DENY utasításokat.
db_backupoperator	Végrehajthatja a DBCC, CHECKPOINT, és BACKUP utasításokat.
db_datareader	Az adatbázis bármely felhasználói táblájából lekérdezhet adatokat.
db_datawriter	Az adatbázis bármely felhasználói táblájában módosíthat, felvehet, törölhet adatokat.

5.8. táblázat.

Táblázat folytatás

Rögzített adatbázis szerepkörök	Leírás
db_denydatareader	Az adatbázis egyetlen felhasználói táblájából sem kérdezhet le adatokat.
db_denydatawriter	Az adatbázis egyetlen felhasználói táblájában sem módosíthat adatokat.

Egy adatbázis minden felhasználója a **public** adatbázis szerepkörbe tartozik. Ha egy felhasználónak nincs megadva külön jogosultság egy objektumon, akkor a **public** szerepkörhöz megadott jogokkal kezelheti.

Az **alkalmazás szerepek** segítségével biztosíthatjuk, hogy egy egyéni alkalmazáson keresztül az adatbázishoz kapcsolódott felhasználó az engedélyezett módon kezelhesse az adatokat. Használat:

- Az alkalmazás indítása
- Az alkalmazás kapcsolódik az SQL server példányhoz a megfelelő felhasználói névvel
- Az alkalmazás futtatja az **sp_setapprole** tárolt eljárást a szerepnév és jelszó ismeretében (a jelszót a szerep létrehozásakor kell megadni)
- Ha a szerpnév és a jelszó megfelelő, akkor aktiválódik az alkalmazás szerep.
- Ezután a felhasználói névhez tartozó engedélyek helyett az alkalmazás szerephez tartozó engedélyek lesznek érvényben, amíg tart a kapcsolat.

5.2.4. Adattípusok és adattábla szerkezetek

Adattípusok (Data Types)

Egy oszlop a megfelelő objektum adott attribútumát reprezentálja, ezért az oszlop minden előfordulása azonos *adattípusú*. Az adattábla szerkezetének definiálásánál meg kell adni az oszlopok adattípusait.

Az alábbi táblázat az SQL Server 2005 adattípusait ismerteti röviden:

Adattípus	**Leírás**
binary[(n)]	Fix-hosszúságú, bináris adat, maximum 8000 bájttal.
bigint	Hosszú egész. Tartomány: -2^63 (-9.223.372.036.854.775.808) és 2^63-1 (9.223.372.036.854.775.807) között. Tárolási méret: 8 bájt.
bit	Bit adat 1 (true), 0 (false) vagy NULL értékkel.
char[(n)]	Fix-hosszúságú nem **Unicode** szöveges adat, maximum 8000 karakterrel.
cursor	Hivatkozás **cursor**-ra. A **cursor** olyan eredményhalmaz, amely egy időben egy sort tud visszaadni.
datetime	Dátum és idő 1753. január 1. és 9999. december 31. között. Pontosság: 3,33 ms. Tárolási méret: 8 bájt.
decimal(p [, s])	Fix pontosságú és hosszúságú (hossz, tizedesek száma adható meg) numerikus adat -10^38 +1 és 10^38 –1 között. Korlátok: 1<= p <= 38, 0 <= s <= p.
float[(n)]	Lebegőpontos szám -1.79E + 308 és 1.79E + 308 között. Tárolási méret n értékétől függ: 1<=n<=24 esetén 4 bájt, 25<=n<=53 esetén 8 bájt .
image	Változó hosszúságú, binárisadat. Maximális méret: 2^31 - 1 (2.147.483.647) bájt.
int	Egész szám -2^31 (-2.147.483.648) és 2^31 - 1 (2.147.483.647) között. Tárolási méret 4 bájt.
money	Pénzügyi érték -2^63 (-922.337.203.685.477,5808) és 2^63 - 1 (+922.337.203.685.477,5807) között. Pontosság: a pénzügyi egység tízezred része. Tárolási méret: 8 bájt.
nchar[(n)]	Fix-hosszúságú **Unicode** adat, maximálisan 4000 karakterrel.
ntext	Változó hosszúságú **Unicode** adat, maximálisan 2^30 - 1 (1.073.741.823) karakterrel.

5.9. táblázat.

Táblázat folytatás

Adattípus	Leírás
nvarchar[(n|max)]	Változó hosszúságú **Unicode** adat, maximálisan 4000 karakterrel. A tárolási méret **max** esetén 2^30 - 1 (1.073.741.823) bájt. Aktuális méret: adathossz + 2 bájt.
numeric	Ugyanaz, mint a **decimal** típus.
real	Lebegőpontos szám -3.40E + 38 és 3.40E + 38 között. Tárolási méret: 4 bájt.
smalldatetime	Rövid dátum és idő 1900. január 1. és 2079. június 6. között. Pontosság: 1 perc. Tárolási méret: 4 bájt.
smallint	Rövid egész -2^15 (-32.768) és 2^15 - 1 (32.767) között. Tárolási méret: 2 bájt.
smallmoney	Pénzügyi érték -214.748,3648 és +214.748,3647 között. Pontosság: a pénzügyi egység tízezred része. Tárolási méret: 4 bájt.
sql_variant	Az SQL Server által támogatott adattípusok tárolása, kivéve **text**, **ntext**, **timestamp**, és **sql_variant**.
sysname	Rendszertámogatott, felhasználói adattípus, ami megegyezik az **nvarchar(128)**. Adatbázisbeli objektumnévre hivatkozunk vele.
table	Speciális adattípus eredményhalmaz tárolására, későbbi feldolgozás céljából.
text	Változó hosszúságú nem **Unicode** adat, maximum 2^31 - 1 (2.147.483.647) karakterrel.
timestamp	Az adatbázison belül egyedi szám. A táblasor módosításakor mindig változik (időbélyegzés).
tinyint	0 és 255 közötti egész szám. Tárolási méret 1 bájt.
varbinary[(n|max)]	Változó hosszúságú binárisadat, maximum 8000 bájttal. A tárolási méret **max** esetén 2^31 - 1 (2.147.483.647) bájt. Aktuális méret: adathossz + 2 bájt.
varchar[(n|max)]	Változó hosszúságú nem **Unicode** adat, maximum 8000 karakterrel. A tárolási méret **max** esetén 2^31 - 1 (2.147.483.647) bájt. Aktuális méret: adathossz + 2 bájt.
uniqueidentifier	16 bájtos globális egyedi azonosító (GUID).
xml	XML adatok tárolására szolgáló típus.

Ideiglenes (Temporary) táblák

Az SQL Server támogatja az ideiglenes táblák használatát. Az ilyen táblák neve előtt a # jelet kell használni. Ha nem gondoskodunk törlésükről a lecsatlakozásig, akkor az SQL Server automatikusan törli az ideiglenes táblát. Az ideiglenes táblákat a **tempdb** adatbázis tárolja. Kétféle ideiglenes tábla létezik:

- **Helyi** (local) ideiglenes tábla

  A nevük a # jellel kezdődik. Csak abban a kapcsolatban látható, amely létrehozta.

- **Általános** (global) ideiglenes tábla

  A nevük a ## jelekkel kezdődik. Az ideiglenes globális táblákat az összes kapcsolat látja. Ha nem törlődnek közvetlenül a létrehozó kapcsolatának megszűnéséig, akkor a rendszer törli, amikor az összes feladat (task) befejeződik, amely rájuk hivatkozik.

5.2.5. Nézetek (Views)

A nézet lehet virtuális tábla vagy egy tárolt lekérdezés. A nézetben kezelt adatok nem tárolódnak külön adatbázisbeli objektumban. Az adatbázisban csak a SELECT utasítást tároljuk. A SELECT utasítás eredménytáblája adja a nézet virtuális tábláját. Az SQL utasításokban a nézettáblát ugyanúgy használhatjuk, mint az adattáblákat.

A nézetek kezelhetnek több adatbázisra, vagy **SQL Server 2005** példányra felosztott (**particionált**) adattáblákat is. Az eredeti adattábla több résztáblára kerül felosztása, ezek mindegyike az eredeti tábla sorainak részhalmazát adja. Minden résztábla más-más szerveren levő adatbázisba kerül. Minden szerveren **particionált nézetet** definiálunk.

A nézetek módosíthatók (az eredeti adattábla a nézeten keresztül módosítható) az UPDATE, DELETE, vagy INSERT utasítások használatával.

Indexelt nézetek segítségével növelhetjük a nézetek teljesítményét. Az **SQL Server 2005** támogatja **nyalábolt** (clustered) index létrehozását nézeten.

5.2.6. Tárolt eljárások

A tárolt eljárás egy egyszerű végrehajtási tervbe lefordított T-SQL utasítások sorozata. A tárolt eljárások négyféle módon adnak vissza adatot:

- Output paramétereket, amelyek valamilyen adatot adnak vissza (egész vagy szöveges értéket), vagy **cursor** adattípusú változót.
- Visszatérési kódot, amely mindig egész érték.
- A tárolt eljárásban, vagy az eljárás által hívott egyéb tárolt eljárásban levő minden SELECT utasításra egy eredményhalmazt.
- Egy globális **cursor**-t, amely az eljáráson kívülre hivatkozhat.

Sok feladat SQL utasítások sorozatára vezethető vissza. Az első SQL utasítások eredményéhez megadott feltétel-logika meghatározza, hogy mely rész utasításcsoport legyen végrehajtva. Ha ezeket az SQL utasítások és a feltétel-logikát egy tárolt eljárásba írjuk, akkor az SQL szerveren ez egy egyszerű végrehajtási tervként jelenik meg. Az eredményeket nem kell az ügyfélnek visszaküldeni, minden munka a kiszolgálón kerül végrehajtásra.

Tárolt eljárások és végrehajtási terv

Az **SQL Server 2005** nem menti a részlegesen lefordított végrehajtási tervet a tárolt eljáráshoz, amikor azt létrehozzuk. A tárolt eljárás futásidőben fordítódik, hasonlóan, mint a T-SQL utasítások. Az SQL Server megőrzi az összes SQL utasítás *végrehajtási tervét* az **eljárás bufferben** (procedure cashe), és a még nem tárolt eljárások végrehajtási tervét is. Ha az adatbázismotor úgy találja, hogy egy SQL utasítás megegyezik valamely végrehajtási terv utasításával, akkor felhasználja a tervet.

Az **SQL Server 2005** támogatja **ideiglenes** tárolt eljárások használatát is, amelyek hasonlóan az ideiglenes táblákhoz, automatikusan törlődnek, amint lecsatlakozunk az adatbázisról. Az ideiglenes tárolt eljárások a **tempdb** adatbázisban tárolódnak.

5.2.7. Felhasználói függvények

A programozási nyelvek függvényei olyan szubrutinok, amelyek az ismételten végrehajtandó tevékenységek megvalósítását végzik. A függvényben tárolt kód ismételt végrehajtásához csak a függvény meghívására van szükség. Az **SQL Server 2005** kétféle függvényt támogat:

- **Beépített** függvények
  A T-SQL leírásnak megfelelően működnek, nem módosíthatók. A függvényekre SQL utasításokban hivatkozhatunk, a szintaktikának megfelelően.
- **Felhasználói** függvények (UDF)

  A T-SQL CREATE FUNCTION utasítását használva saját függvényeket definiálhatunk. A felhasználói függvények 0, vagy több input paramétert fogadhatnak, és egy értéket adnak vissza. Általában a felhasználó függvények skaláris értéket adnak vissza.

Az **SQL Server 2005** olyan felhasználói függvényeket is támogat, amelyek **table** adattípust adnak vissza:

- Deklarálni tud függvény belső táblaváltozót, tud sorokat beszúrni a változóba, azután visszaadja a változót, mint visszatérési érték.
- A felhasználói függvények egy csoportja a SELECT utasítás eredménytábláját egy **table** típusú változóba tölti ki. Az ilyen függvényeket **in-line** függvényeknek nevezzük. Ezek a függvények azon a helyen használhatók, ahol tábla típusú kifejezés megadható.

A táblát visszaadó felhasználói függvények olyan helyeken használhatók, ahol a tábla és nézet kifejezések megengedettek egy SQL lekérdezésben. A nézetek egy egyszerű SELECT utasításra épülnek, míg a felhasználói függvények egyéb utasításokat is tartalmazhatnak, így sokkal bonyolultabb megvalósítást biztosíthatnak.

5.2.8. Megszorítások (Constraints)

A megszorítások (kényszerek) segítségével lehetőségünk van az adatbázis-integritás automatikus megőrzésére. A megszorítások szabályokat definiálnak, amelyek ellenőrzik az oszlopokban megengedhető értékeket. A **constraint**-ek használatát általában előnyben részesítik a triggerekkel, szabályokkal és alapértelmezett értékekkel szemben. A lekérdezés-optimalizáló is használ constraint definíciókat a nagyteljesítményű végrehajtási tervek létrehozásakor.

Az SQL Server 2005 öt constraint osztályt támogat:

NOT NULL

Az oszlopban nem engedhető meg a NULL érték, kötelező megadni adatot.

CHECK

A CHECK constraint kikényszeríti a tartományintegritást az oszlopban megadható értékek korlátozásával.

A CHECK constraint meghatároz egy logikai kifejezést, amely kiértékelődik, amikor meg-adunk egy oszlopértéket. Ha a logikai kifejezés értéke FALSE, akkor elveti a megadott értéket. Egy oszlophoz több CHECK constraint is tartozhat.

UNIQUE

A UNIQUE constraint biztosítja, hogy a nem NULL értékű oszlopértékek egyediek. Az elsődleges kulcs is biztosítja az egyediséget, de nem enged meg NULL értéket.

PRIMARY KEY

A PRIMARY KEY constraint-ek megadnak egy, vagy több oszlopot, amelyek értékei egyértelműen azonosítanak egy táblabeli sort. A PRIMARY KEY nem engedi meg a NULL érték használatát a kulcs oszlopaiban. Táblánként csak egy elsődleges kulcs lehet.

Egy tábla rendelkezhet egynél több olyan oszlopkombinációval, amelyek egyértelműen azonosítják a sort. Minden ilyen kombináció **jelölt** (candidate) kulcs. Az adatbázis adminisztrátor a jelölt kulcsok közül kiválaszt egyet elsődleges kulcsnak.

FOREIGN KEY

A FOREIGN KEY constraint kapcsolatot biztosít két tábla között. Az egyik tábla idegen kulcsa a másik tábla jelölt kulcsára mutat.

Az idegen kulcs megakadályozza olyan kulcsérték bevitelét, amely nem fordul elő a kapcsolódó jelölt kulcs értékei között.

Nem lehet beszúrni idegen kulcs értéket (kivéve NULL érték), ha az nem jelölt kulcs erre az értékre. Az ON DELETE klauzula megadja, hogy mi történjen, ha olyan sort akarunk törölni, amely létező idegenkulcsra mutat. Az ON DELETE klauzula két beállítással rendelkezik:

- NO ACTION megadja, hogy hibát eredményezzen az ilyen törlés.
- CASCADE megadja, hogy az összes sor az idegen kulcsokkal, amely a törölt sorra mutat, szintén törlődjön.

Az ON UPDATE klauzula megmondja, hogy mi történjen, ha módosítunk egy jelölt kulcsértéket, amely értéke létező idegen kulcsokra mutat. Szintén a NO ACTION és CASCADE beállításokat támogatja.

Oszlop (Column) és tábla (Table) megszorítások

A megszorítások lehetnek oszlop, illetve tábla megszorítások:

- Az oszlop megszorítás része az oszlopdefiníciónak, és csak az adott oszlopra vonatkozik.
- A tábla megszorítást oszloptól elkülönítetten definiáljuk, és a tábla több oszlopára vonatkozhat.

5.2.9. Üzleti szabályok (Rules)

A szabályok a korábbi verziókhoz való kompatibilitás miatt vannak a rendszerben. Helyette a CHECK megszorítást használjuk. A CHECK constraint sokkal tömörebb, mint a szabály. Ellentétben a szabállyal, egy oszlophoz több CHECK constraint is tartozhat. CHECK constraint a CREATE TABLE utasítás része, míg a szabályok külön objektumok, és kötni kell őket oszlophoz.

5.2.10. Alapértelmezett értékek (Defaults)

A **default** megadja, hogy milyen értéket tároljon az oszlopban, ha egy sor beszúrásakor nem adunk meg értéket. Az alapértelmezett érték bármi lehet, amit kiértékelve egy értéket (állandót) kapunk.
Kétféle módon használhatjuk az alapértelmezett értékeket:

- Definiálunk egy alapértelmezett értéket a CREATE TABLE utasítás DEFAULT kulcsszavával, ami az adott oszlop alapértelmezett értékét adja. Ez a javasolt, elsődleges módszer.
- Létrehozunk egy adatbázis default objektumot a CREATE DEFAULT utasítással és összekötjük az adott oszloppal az **sp_bindefault** rendszer tárolt eljárást használva. Ez felel meg a korábbi verziókkal való kompatibilitásnak.

5.2.11. Triggerek (Triggers)

A triggerek speciális tárolt eljárások, amelyek automatikusan végrehajtódnak, amikor egy adattáblán vagy nézettáblán UPDATE, INSERT, vagy DELETE utasítás kerül végrehajtásra. Egy táblához több triggert is létrehozhatunk. A CREATE TRIGGER utasítás használható a FOR

UPDATE, FOR INSERT, vagy FOR DELETE klauzulákkal, amely meghatározza, hogy a trigger milyen tevékenységgel kerüljön végrehajtásra. Ha FOR UPDATE a klauzula, akkor az IF UPDATE (oszlopnév) klauzula használható az adott oszlop módosítási tevékenységének figyelésére.

A triggerek tartalmazhatnak SQL utasításokat. A triggerek, hasonlóan, mint a tárolt eljárások, visszaadhatnak olyan eredményhalmazt, amit a triggerben használt SELECT utasítás eredményez. A triggerben nem ajánlott SELECT utasítást elhelyezni, kivéve, ha csak paramétereket tölt ki.

A FOR klauzulában megadjuk, hogy a trigger milyen funkció (INSERT, UPDATE, vagy DELETE) előtt kerüljön végrehajtásra.

AFTER

A trigger meghívásra kerül miután a figyelt utasítás (INSERT, UPDATE, vagy DELETE) le-futott. Ha az utasítás végrehajtása közben hiba keletkezik, akkor a trigger nem hajtódik végre. AFTER típusú triggereket nézetekhez nem definiálhatunk. Minden figyelt tevékenységre több AFTER trigger is definiálható. Ha egy adattáblához többszörös AFTER triggert definiáltunk, akkor az **sp_settriggerorder** tárolt eljárással adjuk meg az elsőre, és utoljára végrehajtandót. A többi trigger végrehajtási sorrendjét nem tudjuk befolyásolni.

Az AFTER az alapértelmezett.

INSTEAD OF

A trigger meghívásra kerül a figyelt tevékenység (INSERT, UPDATE, vagy DELETE) bekövetkezésekor. Az INSTEAD OF típusú trigger megadható adattáblához és nézethez is. Minden figyelt tevékenységre csak egy INSTEAD OF típusú trigger definiálható. Az INSTEAD OF triggereket elsősorban integritásellenőrzésre használjuk az INSERT, UPDATE utasításoknál.

5.2.12. Jelsorrendek (Collations)

Egy **jelsorrend** (collation) definiál egy bitsorozatot az egyes karakterek megjelenítésére és olyan szabályt, amely a karakterek rendezésére és összehasonlítására szolgál. Objektumonként, oszloponként különböző jelsorrendet adhatunk meg.

A karakteres adatok tárolási módja

A kódlapok bitsorozatot definiálnak a nagybetűs, kisbetűs karakterekre, számjegyekre, szimbólumokra és egyéb speciális (!, @, # stb.) karakterek-

re. Karakterenként 1 bájtot használva, 256 különböző karaktert tudunk reprezentálni egy-egy 8 bites bitsorozattal. 2 bájton (16 bit) 65.536 lehetőségünk adódik.

Minden európai nyelv saját egybájtos kódlappal rendelkezik. Bizonyos jeleknek (Latin ABC A-tól Z-ig, számjegyek, stb.) azonos a kódolása minden kódlapban, míg mások (például az ékezetes betűk) kódlaponként eltérőek. Ha az adatokat, egy másik kódlapot használó számítógépre visszük át, akkor konvertálni kell azokat az új kódlapnak megfelelően. Eközben előfordulhat, hogy „elvesznek" adataink. Még bonyolultabb a helyzet, ha egy nemzetközi adatbázishoz több országból csatlakoznak ügyfelek.

Az egybájtos karakterkészlet alkalmatlan az ázsiai nyelvek által használt nagyszámú karakterek tárolására. Az ilyen nyelvekre kétbájtos karakterkészletet definiáltak.

A fenti probléma megoldására az ISO szervezet és a Unicode Consortium csoport definiálta a **Unicode** standard karakterkészletet. A **Unicode** két bájtot használ minden karakter tárolására. A 65.536 lehetőség biztosítja, hogy nemzetközi hálózatokban sem kell karakterkonverziót végrehajtani, ha mindegyik a **Unicode** karakterkészletet használja.

A Windows operációs rendszerek telepítésekor meg kell adni a helyet, ami meghatározza, hogy az operációs rendszer milyen kódlapot használjon. A Windows alkalmazások a helynek megfelelő kódlapot használják az adatok megjelenítésére. A Windows alkalmazások támogatják a **Unicode** karakterkészlet használatát is.
Az SQL Server 2005 a szöveges adattípusok két kategóriáját támogatja:

- **Unicode** adattípusok: **nchar**, **nvarchar**, és **ntext**. Ezek az adattípusok a **Unicode** karakter reprezentációt használják, nincs kódlapjuk.
- **Nem-Unicode** adattípusok: **char**, **varchar**, és **text**.

Rendezés

A **rendezési sorrend** meghatározza az SQL Server által használt szabályt a szöveges adatok értelmezésére, jelsorrendjére, összehasonlítására és megjelenítésére. Például, a rendezési sorrend definiálja, hogy az 'a' karakter kisebb, egyenlő, vagy nagyobb, mint a 'b' karakter. A rendezési sorrend megadja, hogy a jelsorrend *case-sensitive* (megkülönböztetjük-e a kis- és nagybetűket), vagy nem. Például az 'm' karakter megegyezik, vagy nem az

'M' karakterrel. Definiálja, hogy a jelsorrend ékezet érzékeny-e (*accent-sensitive*). Például az 'á' karakter egyenlő, vagy nem az 'ä' karakterrel.

5.2.13. Indexek

Az **SQL Server 2005** index egy struktúra, amely adattáblához, vagy nézethez kapcsolódik. Megnöveli a tábla, vagy nézetbeli sorok visszahozásának sebességét. A strukturált tárolás biztosítja a kulcshoz tartozó sorok gyors és eredményes megtalálását.

Tábla indexek

Az adattábla bármely oszlopára definiálhatunk indexet, még a számított oszlopra is. Az SQL Server két indextípust különböztet meg:

- **Nyalábolt** (clustered) indexet
  A nyalábolt indexek az adattáblában levő sorokat az indexkulcsaik alapján tárolják és rendezik. Értelemszerűen, táblánként csak egy nyalábolt index lehetséges. Az adatsorok a nyalábolt indexstruktúra legalsó szintjén helyezkednek el. Ha egy táblában nincs nyalábolt index, akkor az adatsorok tárolása kupacban (heap) történik.
- **Nem nyalábolt** (nonclustered) indexet
  A nem nyalábolt indexek az adatsoroktól elkülönült struktúrában tárolódnak. A nem nyalábolt indexstruktúra legalsó sora tartalmazza az indexkulcs értéket, és minden kulcsértékhez tartozik egy mutató, amely arra az adatsorra mutat, ami tartalmazza a kulcsértéket. Az adatsorok nem a kulcsnak megfelelő rendezettségben tárolódnak.

Az indexsorban levő mutatót **sorlokátornak** nevezzük. A sorlokátor szerkezete függ attól, hogy az adatlapok kupacban, vagy nyaláboltan tárolódnak. Kupac esetén a sor lokátor a sorra mutat. Nyalábolt indexű tábla esetén a sorlokátor a nyalábolt index kulcs.

Az indexek lehetnek egyediek, ami azt jelenti, hogy nincs két olyan sor a táblában, amely ugyanazt az értéket adná az indexkulcsra. Egyébként az index nem egyedi, és több sorhoz is ugyanolyan indexkulcs tartozhat.

Kétféleképpen definiálhatunk indexeket. A CREATE INDEX utasítás létrehoz és elnevez egy indexet. A CREATE TABLE utasítás az alábbi **constraint**-eket támogatja, amelyek indexet hoznak létre:

- PRIMARY KEY egyedi, elsődleges indexet hoz létre,
- UNIQUE egyedi indexet hoz létre,

- CLUSTERED nyalábolt indexet hoz létre,
- NONCLUSTERED nem-nyalábolt indexet hoz létre.

A rendezettség iránya lehet növekvő, vagy csökkenő.

Az SQL Server **index kitöltési tényező** (fill faktor) tulajdonsága szabályozza, hogy létrehozáskor milyen sűrűen legyen becsomagolva az index. Az alapértelmezett kitöltési tényező általában jó teljesítményt eredményez, de néhány esetben célszerű lehet módosítani. Sok módosítás, beszúrás esetén célszerű kisebbre állítani az értéket, hogy több hely maradjon az új kulcsok számára. A kitöltési faktor csak az index létrehozásakor adható meg.

Nézet indexek

Indexeket létrehozhatunk nézeten is. Egy nem indexelt nézet eredményhalmaza nem tárolódik az adatbázisban. A nézetre hivatkozva az SQL Server 2005 előveszi a definíciót és létrehozza az eredménytáblát. A nézet eredménytáblájának létrehozási folyamatát *materializálásnak* nevezzük.

Amikor egy *egyedi nyalábolt indexet* hozunk létre nézeten, a nézet végrehajtódik, és az eredménytábla az adatbázisban tárolódik, hasonlóan, mint a nyalábolt indexű adattábláknál. A nyalábolt index létrehozásakor az eredménytábla azonnal létrejön. Automatikusan tükrözi az adattáblákban történő módosításokat.

A nézeten létrehozott első indexnek egyedi nyalábolt indexnek kell lennie. Miután létrehoztuk, már egyéb nem nyalábolt indexeket is definiálhatunk.

Ugyanúgy, mint az adattáblákhoz tartozó nyalábolt indexben, a nyalábolt index **B-fa** struktúrája csak a kulcsoszlopokat tartalmazza. Az adatsorok a nézet eredménytáblájában tárolódnak.

5.3. Példa adatbázis az utasítások használatához

Egy strandkölcsönző az alábbi információkat kezeli egy számítógépes nyilvántartási rendszerben:

Adatok, információk	Leírás
A kölcsönző személyi igazolványának száma	8 karakter. Az első kettő nagybetű, a további 6 számjegy karakter. A kölcsönző személy egyedi azonosítója. Egy szezonban, napon belül többször, több eszközt is kölcsönözhet egy személy.
A kölcsönző neve.	Megadása kötelező.
A kölcsönző lakcíme.	Irányítószám, település, utca, házszám. Csak a település kitöltése kötelező.
A kölcsönzés dátuma (év, hónap, nap).	A kölcsönzéskor (új rekord felvitele) a rendszer-dátumot automatikusan adja a program, amit módosíthatunk.
A kikölcsönzött eszköz.	Lehetséges eszközök: napozóágy, gumimatrac, csónak, vízi bicikli. Legkelendőbb a napozóágy.
A kölcsönzés kezdő időpontja (óra, perc).	A kölcsönzéskor (új rekord felvitele) a rendszeridőt automatikusan adja a program, amit módosíthatunk.
A kölcsönzés tervezett időtartama (óra) eszközönként.	A visszahozáskor módosítható adat.
A kölcsönzési díj összege eszközönként, óránként.	50 Ft-nál nagyobb összeg.

5.10. táblázat.

Kölcsönzési szabályok:

- Személyi igazolvánnyal lehet csak kölcsönözni.
- Egy kölcsönzéssel több eszközt (azonos típusból is és különböző típusból is) elvihetünk.
- A kölcsönzéskor fizetni kell a megadott időtartamra. Az időtartam eszköz fajtánként azonos.
- Egy kölcsönzés a nap végén (legkésőbb záráskor) befejeződik.

- Késedelmes visszahozáskor fizetendő a késedelmi díj (az időtartam módosítandó).
- Ha eszköz fajtánként többet kölcsönöztünk, akkor azokat együtt kell visszavinni, ezzel befejezetté válik a kölcsönzés.

Az alábbi táblázatok tartalmazzák az egyedhalmazok, attribútumok neveit, az adatbázisban használt kódok jegyzékét:

Egyedhalmaz	**Attribútum**		**Tulajdonság típusa**
	név	**szöveges értelmezés**	
Ugyfelek	USZIGSZ	Személyi ig. száma	azonosító
	UNEV	Kölcsönző neve	leíró + index
	UIRSZ	Lakcím - irányítószám	leíró
	UTELEP	Lakcím - település	leíró
	UUTCA	Lakcím - utca, házszám	leíró
Eszkozok	EKOD	Eszköz sorszám	azonosító
	ENEV	Eszköz megnevezés	leíró
	EDIJ	Kölcsönzési díj (Ft/óra)	leíró
Kolcsfej	KAZON	Kölcsönzésazonosító	azonosító
	KUSZIGSZ	Kölcsönző szig. szám	leíró + kapcs.
	KDATUM	Kölcsönzés dátuma	leíró + index
	KKIDO	Kölcsönzés kezdete	leíró
Kolcstet	KAZON	Kölcsönzésazonosító	azonosító 1
	KEKOD	Eszköz sorszám	azonosító 2
	KTARTAM	Kölcsönzés időtartam (óra)	leíró
	KMENNY	Mennyiség (db)	leíró
	KRENDB	Befejezettség jelző	leíró

5.11. táblázat.

Kódjegyzék			
Tulajdonság-típus neve	**A kód szöveges értelmezése**	**Felvehető értékek vagy kódjegyzék hivatkozás**	**Előfordulás helye**
EKOD KEKOD	Eszköz sorszám	”1” - napozóágy ”2” - gumimatrac ”3” - vízi bicikli ”4” - motorcsónak	Eszkozok Kolcstet

5.12. táblázat.

A *StrandKolcs* adatbázis *sémája* (primary key: aláhúzott, foreign key: *dőlt*)
Ugyfelek (USZIGSZ, UNEV, UIRSZ, UTELEP, UUTCA)
Eszkozok (EKOD, ENEV, EDIJ)
Kolcsfej (KAZON, *KUSZIGSZ*, KDATUM, KKIDO)
Kolcstet (*KAZON*, *KEKOD*, KTARTAM, KMENNY, KRENDB)

A következő táblázat az attribútumok adattípusait tartalmazza (PK – primary key, FK – foreign key, I – általános index):

Reláció	**Attribútum**		
	név	**adattípus**	**Indexek**
Ugyfelek	USZIGSZ	CHAR(8)	PK
	UNEV	VAR CHARYING(30)	I
	UIRSZ	CHAR(4)	
	UTELEP	VAR CHARYING(20)	
	UUTCA	VAR CHARYING(25)	
Eszkozok	EKOD	CHAR(1)	PK
	ENEV	VAR CHARYING(20)	
	EDIJ	SMALLINT	
Kolcsfej	KAZON	INTEGER	PK
	KUSZIGSZ	CHAR(8)	FK
	KDATUM	DATE	I
	KKIDO	CHAR(5)	

5.13. táblázat.

Táblázat folytatás

Reláció	Attribútum		
	név	**adattípus**	**Indexek**
Kolcstet	KAZON	INTEGER	PK1
	KEKOD	CHAR(1)	PK2
	KTARTAM	SMALLINT	
	KMENNY	SMALLINT	
	KRENDB	BIT	

5.4. Lekérdező utasítások

5.4.1. A SELECT utasítás

Szintaktika:

```
SELECT [ALL | DISTINCT] [TOP nKif [PERCENT]]
        [Alias.] Oszlop_1 [Oszlop_név1]
        [, [Alias.] Oszlop_2 [Oszlop_név2] ...]
FROM [[adatbázis.]tulajdonos.]táblanév_1 [Alias_1]
        [{CROSS|NATURAL|INNER |
        {LEFT | RIGHT | FULL} [OUTER]}]
        JOIN [[adatbázis.]tulajdonos.]táblanév_2 [Alias_2]
        {ON JoinFeltétel … | USING (Oszlop_1[, Oszlop_2 ...)}
[WHERE JoinFeltétel [AND JoinFeltétel ...] [AND | OR SzűrőFeltétel
                        AND | OR SzűrőFeltétel ...]]]
[GROUP BY CsoportOszlop_1[, CsoportOszlop_2 ...]]
[HAVING SzűrőFeltétel]
[ORDER BY RendezőOszlop_1 [ASC | DESC]
                [, RendezőOszlop_2 [ASC | DESC] ...]];
```

A SELECT és FROM záradékok használata kötelező, a többi opcionális.

A SELECT záradék

A SELECT záradékban definiáljuk az eredménytábla oszlopait. Az oszlopok jöhetnek a FROM záradékban megadott táblából vagy nézetből. Speciálisan a csillag (*) használatával a tábla minden oszlopát kijelöljük. A következő utasítás az *Ugyfelek* tábla minden oszlopát megjeleníti az eredménytáblában:

```
SELECT * FROM Ugyfelek;
```

Ezen kívül megadhatunk ún. *számított oszlopokat* is. Számított oszlop esetén megadjuk azt a kifejezést, amellyel számítódik az oszlopbeli érték. A kifejezésben előfordulhatnak:

- mezőnevek,
- literálok,
- operátorok,
- belső függvények,
- összesítő (aggregáló) függvények.

Ha a mező több táblában is megtalálható, a mező neve elé beírjuk a tábla nevét, illetve a FROM záradékban megadott tábla alias-nevet (másodlagos név), és a **.**(pont) operátort.

Az adott oszlop nevét, oszlopfeliratát módosíthatjuk:

```
Oszlop_1 [AS] Oszlop_név_1
```

Az AS kulcsszó elhagyható. Számított oszlop esetén, ha nem adunk meg nevet, akkor a rendszer automatikusan ad nevet, pl. Exp1 (kifejezés 1).

ALL, DISTINCT, TOP predikátumok

Az eredménytábla sorainak számát befolyásolhatjuk:

Prédikátum	Magyarázat
ALL	Alapértelmezés. Minden sor megjelenik az eredménytáblában.
DISTINCT	Kihagyja az ismétlődő sorokat. Az ismétlődésekből csak egy sor jelenik meg az eredménytáblában.

5.14. táblázat.

Táblázat folytatás

Prédikátum	Magyarázat
TOP n [PERCENT]	A lekérdezés eredményébe a megadott számú sor (TOP n), vagy az eredménytábla sorainak adott százaléka (TOP n PERCENT) kerül az eredménytábla elejéről vagy végéről, az ORDER BY záradéknak megfelelően.

NULL-érték kezelés

Az alábbi CASE-szerkezet segítségével a NULL-érték helyett az eredménytáblában kifejezés-értéket jeleníthetünk meg.

```
CASE
        WHEN kifejezés1 IS [NOT] NULL THEN kifejezés_a
        WHEN kifejezés2 IS [NOT] NULL THEN kifejezés_b
        ….
        ELSE kifejezés_x
END [oszlopfelirat]
```

Néhány általánosan használt belső függvény

Szövegkezelő függvények	
LEFT(KKif, NKif)	Az első szöveges típusú argumentumból baloldalról függvényértékként annyi karaktert ad vissza, mint amennyi a második numerikus argumentum értéke.
SUBSTRING(KKif, NKif1, NKif2)	Az első szöveges típusú argumentum belsejéből függvényértékként annyi karaktert ad vissza, mint amennyi a harmadik numerikus argumentum értéke, a második numerikus argumentum értékének megfelelő kezdő pozíciótól.

5.15. táblázat.

Táblázat folytatás

Szövegkezelő függvények	
RIGHT(KKif, NKif)	Az első szöveges típusú argumentumból jobboldalról függvényértékként annyi karaktert ad vissza, mint amennyi a második numerikus argumentum értéke.
LOWER(KKif)	A szöveges típusú argumentum karaktereit kisbetűsre konvertálja.
UPPER(KKif)	A szöveges típusú argumentum karaktereit nagybetűsre konvertálja.
LTRIM(KKif)	Levágja a szöveges típusú argumentum baloldali szóköz karaktereit.
RTRIM(KKif)	Levágja a szöveges típusú argumentum jobboldali szóköz karaktereit.
LEN(KKif)	A szöveges típusú argumentum karaktereinek számát adja függvényértékként, mint numerikus adat.
STR(NKif)	A numerikus típusú argumentumot szöveges típusra konvertálja.
Dátum és idő függvények	
DATEADD(dátumrész, NKif, DKif)	A dátumrész lehet: day, month, week, year. A harmadik argumentumban megadott dátumhoz hozzáad az első argumentumban megadott dátumrészből a második argumentumban levő numerikus értéknek megfelelő mennyiséget. A függvényérték dátum típusú.
DATEDIFF(dátumrész, DKif1, DKif2)	A dátumrész lehet: day, month, week, year. A harmadik és második argumentumban levő dátumok különbségét adja függvényértékként az első argumentumban levő dátumrésznek megfelelő egységben. A függvényérték numerikus.

Táblázat folytatás

Dátum és idő függvények	
DATEPART(dátumrész, DKif)	A dátumrész lehet: day, month, week, year. A második argumentumban levő dátumból az első argumentumnak megfelelő dátumrészt adja vissza numerikus típusú függvényértékként.
DAY(DKif)	A dátum típusú argumentum nap részét adja függvényétéként, mint numerikus adat.
MONTH(DKif)	A dátum típusú argumentum hónap részét adja függvényétéként, mint numerikus adat.
YEAR(DKif)	A dátum típusú argumentum év részét adja függvényétéként, mint numerikus adat.
DATE()	A rendszerdátumot adja függvényértékként, mint dátum típusú adat.
TIME()	A rendszeridőt adja függvényértékként.

Összesítő függvények

Az összesítő függvények az argumentumukban szereplő oszlopon fejti ki hatásukat. Ha a SELECT utasítás tartalmaz GROUP BY záradékot, akkor a művelet csoportonként hajtódik végre.

Az alábbi táblázat az 5 legfontosabb összesítő függvényt tartalmazza:

Függvény	**Leírás**
SUM(<kif>)	Az argumentumban levő kifejezést kiszámítja minden soron és összegzi az értékeket. A kifejezés értéke kötelezően numerikus adat. Ha NULL-érték adódik, akkor azt nem veszi figyelembe.
AVG(<kif>)	Az argumentumban levő kifejezést kiszámítja minden soron és összegzi az értékeket, számlálja az összegzésbe bevont értékek darabszámát. A kifejezés értéke kötelezően numerikus adat. Ha NULL-érték adódik, akkor azt nem veszi figyelembe. Eredményként átlagot számol.

5.16. táblázat.

Táblázat folytatás

Függvény	Leírás
MIN(<kif>)	Az argumentumban levő kifejezést kiszámítja minden soron. Függvényértékként a legkisebb értéket adja. A kifejezés értéke lehet szöveges, numerikus, dátum típusú.
MAX(<kif>)	Az argumentumban levő kifejezést kiszámítja minden soron. Függvényértékként a legnagyobb értéket adja. A kifejezés értéke lehet szöveges, numerikus, dátum típusú.
COUNT(*|<kif>)	A COUNT(*) a sorok számát adja vissza függvényértékként. A COUNT(<kif>) szintaktika esetén a NULL-értékű sorok nem kerülnek be a számlálásba.

A FROM záradék

A FROM záradékban jelennek meg a lekérdezésben résztvevő táblák, nézetek.

Több tábla sorainak összekapcsolása (régebbi szintaktika):

```
FROM tábla1 [aliasnév1], tábla2 [aliasnév2]
        [WHERE tábla1.kulcs_i operátor tábla2.kulcs_x
                [AND|OR tábla1.kulcs_j operátor tábla2.kulcs_y ….]]
```

Ha hiányzik a WHERE záradék, akkor a két tábla Descartes-szorzata lesz az eredménytábla. Összekapcsolás esetén (természetes, théta) a WHERE záradékban adjuk meg a kiválasztási feltételt.

Több tábla sorainak összekapcsolása:

```
FROM tábla1 [aliasnév1] [{CROSS | NATURAL |INNER |
 {LEFT | RIGHT | FULL} [OUTER]}]  JOIN tábla2 [aliasnév2]
{ON JoinFeltétel …| USING (Oszlop_1[, Oszlop_2 ...)}
```

Az összekapcsolási kulcsszavak jelentése:

Kulcsszó	Jelentés
CROSS JOIN	Descartes-szorzat (keresztszorzat).
NATURAL JOIN	Természetes összekapcsolás.
INNER JOIN	Belső összekapcsolás. Csak azok a sorok kerülnek be az eredménytáblába, ahol a tábla1-hez tartozik sor az ON feltételnek megfelelően a tábla2-ből.
LEFT [OUTER] JOIN	Baloldali külső összekapcsolás. A tábla1-ből (baloldali tábla) minden sor bekerül az eredménytáblába. Ha nincs kapcsolódó sor a tábla2-ben, akkor NULL-értékek jelennek meg az adott oszlopokban.
RIGHT [OUTER] JOIN	Jobboldali külső összekapcsolás. A tábla2-ből (jobboldali tábla) minden sor bekerül az eredménytáblába. Ha nincs kapcsolódó sor a tábla1-ben, akkor NULL-értékek jelennek meg az adott oszlopokban.
FULL [OUTER] JOIN	Mindkét oldali (bal és jobb) külső összekapcsolás.

5.17. táblázat.

Az ON összekapcsolási feltétel helyett használható a USING, ha a két táblában azonosak a kapcsolatot biztosító mezőnevek.

A FROM záradékban megadott másodlagos (alias) név minősítheti a SELECT záradék oszlopait. Ha egy táblát önmagával kell összekapcsolnunk, akkor kötelező a használata.

A WHERE záradék

A WHERE záradék szolgál az eredménytábla sorainak kiválasztására, szűrésére. A logikai típusú kifejezésben a szokásosak mellett az alábbi speciális operátorokat is használhatjuk:

A LIKE művelet

Kettő karaktersorozatot hasonlít össze. Igaz értéket ad, ha a LIKE előtti szöveges típusú kifejezés megfelel a LIKE utáni mintának. A NOT kulcsszóval vizsgálhatjuk a LIKE tagadását is.

szöveges kifejezés [NOT] LIKE minta

A minta szöveges típusú literál, amely az alábbi karaktereket tartalmazhatja:

Minta elem	**Leírás**	**Példa**
_	Tetszőleges karakter az adott pozíción.	"A__A" mintának pl. megfelel az "ABBA" karaktersorozat
%	Tetszőleges számú (akár egy sem) karakter.	"B%" mintának pl. megfelel a "Balogh", "Bécs", stb. karaktersorozat.
#	Bármely számjegy karakter az adott pozíción.	"Verzió:#" mintának pl. megfelel a "Verzió:1" karaktersorozat.
[karakter lista]	Bármely a listának megfelelő karakter az adott pozíción.	"[A-Z]#" mintának pl.: megfelel a "K5" karaktersorozat.
[!karakter lista]	Bármely, a listában nem szereplő karakter az adott pozíción.	"[!X,Y,Z]%" mintának pl.: megfelel a "Kató néni" karaktersorozat.
-	Karakter lista intervallummal történő megadásakor az alsó és felső érték elválasztó jele.	
,	Karakter lista felsorolással történő megadásakor a karakterek elválasztó jele.	

5.18. táblázat.

Táblázat folytatás

Minta elem	Leírás	Példa
Egyéb, eddig fel nem sorolt karakter.	Az adott pozíción kötelezően előforduló karaktert jelöl.	

Az IN művelet

Logikai igaz értéket ad vissza, ha a kifejezés értéke megegyezik az IN zárójelében felsorolt valamely értékkel, egyébként hamisat. A NOT kulcsszóval vizsgálhatjuk az IN tagadását is.

kifejezés [NOT] IN (érték1, érték2, . . .)

A BETWEEN művelet

Logikai igaz (True) értéket ad, ha a kifejezés értéke egy intervallumba esik (a szélső értékek is az intervallumhoz tartoznak). Az ellenkező értelmű vizsgálatot a NOT logikai művelet beillesztésével végezhetjük el (ekkor azt vizsgáljuk, hogy a kifejezés értéke kívül esik-e a megadott tartományon).

kifejezés [NOT] BETWEEN alsó érték AND felső érték

A GROUP BY záradék

A (rész)eredménytábla sorait csoportokba szervezhetjük. Erre szolgálnak a csoportképző oszlopok. Azok a sorok tartoznak egy csoportba, ahol a csoportképző oszlopok értékei azonosak. Több oszlop esetén a csoportosítás balról jobbra történik.

Alapvető szabályok a csoportképzésre:

- A csoportképzésben résztvevő oszlopoknak kötelezően meg kell jelenniük a SELECT záradékban.
- A SELECT záradékban a nem csoportképző oszlopok csak összesítő függvények argumentumában fordulhatnak elő.

Az összesítő függvények csoportonként végzik el a műveleteiket.

A HAVING záradék

A HAVING záradékot csak a GROUP BY záradékkal együtt használjuk. A csoportosított eredménytáblára adhatunk meg kiválasztási feltételt. Ha

nincs csoportképzés, akkor a kiválasztási feltételt írjuk a WHERE záradékba.

Az ORDER BY záradék

Az eredménytábla soraira adhatunk meg rendezési előírást. Megadhatjuk oszlopnévvel, vagy a SELECT záradékban szereplő oszlop sorszámával. A rendezés iránya oszloponként lehet növekvő (ASC), vagy csökkenő (DESC). Alapértelmezésben növekvő a rendezés, az ASC kulcsszó elhagyható. A második, harmadik, stb. rendezési szempont csak akkor lép életbe, ha az előzőekben azonos értékek szerepelnek.

Alkérdések használata

Az alkérdés egy SELECT utasítás (al-SELECT) egy SELECT, SELECT...INTO, INSERT...INTO, DELETE vagy UPDATE utasításon, illetve másik alkérdésen belül.

A SELECT-en belüli alkérdés (belső SELECT) szintaktikája:

```
SELECT ...
        FROM ...
        WHERE [kifejezés] operátor (SELECT ... FROM ... [WHERE])
[{AND|OR} ...]
        [GROUP BY ...]
        [HAVING ...]
        [ORDER BY ...] ;
```

A belső SELECT szolgáltatja azt a konkrét értéket (vagy értékeket), amelye(ke)t a WHERE vagy a HAVING záradék operátora a kifejezés alapján kiértékel. Az operátor lehet: théta-operátor, LIKE, IN, BETWEEN. Ha a belső SELECT több értéket ad vissza egy oszlopban, akkor kötelező az ANY, SOME vagy ALL minősítők használata.

Háromféle szintaktika szerint állíthatunk össze alkérdést:

- összehasonlítás [NOT]{ANY | ALL | SOME} (al-SELECT)
- kifejezés [NOT] IN (al-SELECT)
- [NOT] EXISTS (al-SELECT)

Az ALL predikátum használatával választhatjuk ki a fő lekérdezésben (külső SELECT) azokat a rekordokat, amelyek kielégítik az összehasonlító feltételt az alkérdés összes rekordjával szemben.

Az ANY vagy a SOME (ugyanazt jelentik) beállításokkal válogathatjuk ki a főlekérdezésben azokat a rekordokat, amelyek eleget tesznek a megadott összehasonlító feltételnek az alkérdés valamely értékére.

Az IN predikátum segítségével a főlekérdezés eredményéből kiválaszthatjuk csak azokat a rekordokat, amelyekhez az alkérdés eredményében azonos értékkel rendelkező rekordokat találunk.

Az EXISTS és a NOT EXISTS állításokkal dönthetjük el, hogy az alkérdés eredményezett-e legalább egy értéket.

Alkérdésben táblanév_alias révén is hivatkozhatunk egy alkérdésen kívüli FROM záradékban szereplő táblákra.

5.4.2. Halmazműveletek

Az egyesítés, metszet, különbség relációs műveletek végrehajtására szolgálnak. Szintaktika:

```
<al-SELECT utasítás>
{UNION [ALL] | INTERSECT [ALL] | MINUS [ALL] }
<al-SELECT utasítás>
[ORDER BY RendezőOszlop_1 [ASC | DESC] [, RendezőOszlop_2 [ASC | DESC] ...]];
```

A műveletek a két al-SELECT eredménytábláira alkalmazandók. Az ALL használata esetén ismétlődő sorok is előfordulhatnak. A művelet eredményeként kapott táblázat rendezhető az ORDER BY záradék alkalmazásával.

5.4.3. Lekérdezési példák a példa adatbázisból

1. példa: Adjuk meg azt a lekérdezést, amely megjeleníti azokat az ügyfeleket (kölcsönzőket), akiknek a neve 'K'-val kezdődik!

Megoldás:

```
SELECT * FROM Ugyfelek
    WHERE UNEV LIKE 'K%';
```

2. példa: Adjuk meg azt a lekérdezést, amely megjeleníti a napi bevételeket eszközönkénti bontásban!

Megoldás:

```
SELECT KDATUM, KEKOD,
            SUM(KMENNY*KTARTAM*EDIJ) AS Bevétel
    FROM Kolcsfej
            INNER JOIN (Kolcstet INNER JOIN Eszkozok
                    ON KEKOD = EKOD)
                    ON Kolcsfej.KAZON = Kolcstet.KAZON
    GROUP BY KDATUM, KEKOD;
```

3. példa: Adjuk meg azt a lekérdezést, amely megjeleníti adott hónapban (pl. július) a kölcsönzési adatokat! Eszközönként jelenjen meg a kölcsönzési összeg is! Az adatbázis csak egy szezon adatait tartalmazza.

Megoldás:

```
SELECT  F.*, KEKOD, KMENNY, KTARTAM, EDIJ,
                    KMENNY*KTARTAM*EDIJ AS Összeg
    FROM Kolcsfej F, Kolcstet T, Eszkozok E
    WHERE F.KAZON =T.KAZON AND T.KEKOD=E.EKOD
                    AND MONTH(KDATUM) = 7;
```

4. példa: Adjuk meg azt a lekérdezést, amely megjeleníti azt az 5 napot, amikor legtöbben kölcsönöztek!

Megoldás:

```
SELECT TOP 5 KDATUM, COUNT(*) AS Kölcsönzésszám
    FROM Kolcsfej
    GROUP BY KDATUM
    ORDER BY 2 DESC;
```

5. példa: Adjuk meg azt a lekérdezést, amely megjeleníti, hogy adott időpontban melyek a kikölcsönzött eszközök!

Megoldás:

```
SELECT KEKOD, ENEV, KMENNY
    FROM Kolcsfej F, Kolcstet T, Eszkozok E
    WHERE F.KAZON =T.KAZON AND T.KEKOD=E.EKOD
                AND KRENDB=B'0';
```

5.5. Adat karbantartási utasítások

5.5.1. Felvitel

Egy sor felvitele az adattáblába:

```
INSERT INTO [[adatbázis.]tulajdonos.]táblanév [(mező_1[, mező_2 ...])]
{VALUES (érték_1[, érték_2 ...]) | SELECT ... };
```

A VALUES záradékban levő mezőértékeket a sémának megfelelő sorrendben kell megadni. Ha nem adunk meg minden értéket, akkor meg kell adni a mezőnevek listáját, ami mutatja, hogy melyek azok az oszlopok, amelyek kapnak értéket.

Alkérdés a VALUES záradékban (egy sor bevitele):

Egy oszlopos lekérdezés

```
INSERT INTO [[adatbázis.]tulajdonos.]táblanév [(...)]
VALUES (..., mező_i = ( SELECT oszlop FROM ... WHERE ...), ...) ;
```

Több oszlopos lekérdezés

```
INSERT INTO [[adatbázis.]tulajdonos.]táblanév [(...)]
VALUES (..., (mező_i, mező_j[, ...]) = ( SELECT oszlop_1, oszlop_2
[, ... ] FROM ... WHERE ...), ...) ;
```

Tábla feltöltés alkérdéssel (több sor bevitele):

```
INSERT INTO [[adatbázis.]tulajdonos.]táblanév [(mező_1[, mező_2 ...])]
SELECT ... ;
```

Példa: Adja meg azt a lekérdező utasítást, amely a megadott dátumhoz (2005.07.19) tartozó kölcsönzésekre kiírja a Kolcsdij táblába (hozzáfűzi) az azonosító (KAZON), és a kölcsönzés összege sorokat!

Megoldás:

```
SELECT Kolcstet.KAZON,
            SUM(EDIJ*KMENNY*KTARTAM) AS kdij
      INTO Kolcsdij
      FROM Eszkozok INNER JOIN (
            Kolcsfej INNER JOIN Kolcstet
                  ON Kolcsfej.KAZON = Kolcstet.KAZON)
                  ON Eszkozok.EKOD = Kolcstet.KEKOD
      WHERE KDATUM = '2005-07-19'
      GROUP BY Kolcstet.KAZON;
```

5.5.2. Módosítás

Az adattábla egy, vagy több oszlopát módosíthatjuk, a módosításban résztvevő sorokat a WHERE záradék feltétele választja ki:

```
UPDATE [[adatbázis.]tulajdonos.]táblanév
        SET mező_i = kifejezés_i [, mező_j = kifejezés_j ...]
        [WHERE feltétel];
```

Az új értékek megadására alkérdést is használhatunk a SET záradékban.

Egy oszlopos lekérdezés

```
UPDATE [[adatbázis.]tulajdonos.]táblanév
        SET ..., mező_i = ( SELECT oszlop FROM ... WHERE ...)
                [, ...]
        [WHERE feltétel] ;
```

Több oszlopos lekérdezés

```
UPDATE [[adatbázis.]tulajdonos.]táblanév
        SET ..., (mező_i, mező_j[, ...]) =
                (SELECT oszlop_1, oszlop_2[, ... ] FROM ... [WHERE ...])
        [, ...]  [WHERE feltétel] ;
```

5.5.3. Törlés

A WHERE feltételnek megfelelő sorokat töröljük az adattáblából:

```
DELETE FROM [[adatbázis.]tulajdonos.]táblanév
        [WHERE feltétel] ;
```

5.6. Adatdefiníciós utasítások

5.6.1. Adatbázis definiálása MS SQL Server 2005-ben

Létrehozás

Új adatbázis és fájlok létrehozása, vagy meglévő adatbázishoz fájlok hozzákapcsolása.

```
CREATE DATABASE adatbázisnév
[ ON  [ PRIMARY ]
    [ <fájlspecifikáció> [ ,...n ] ]
    [ , <fájlcsoport> [ ,...n ] ]
]
[ LOG ON {<fájlspecifikáció> [ ,...n ] } ]
[ COLLATE jelsorrend_név]
```

A fájlspecifikáció:

```
( [ NAME = logikai_fájlnév , ]
   FILENAME = 'OS_fájlnév'
   [ , SIZE = fájlméret ]
   [ , MAXSIZE = {maximális_méret | UNLIMITED} ]
   [ , FILEGROWTH = növekmény ] ) [ ,...n ])
```

A felhasználói fájlcsoport specifikáció:

```
FILEGROUP fájlcsoport_név [ DEFAULT ] <fájlspecifikáció> [ ,...n ]
```

A következő táblázat az utasításban előforduló kulcsszavak, argumentumok rövid magyarázatát tartalmazza:

Argumentum	Leírás
adatbázisnév	Maximum 128 karakterből állhat, ha van megadva logikai fájlnév, ha nincs, akkor max. 123 karakter.
ON	Az adattárolásra közvetlenül definiált fájlokat használunk. Ezek a fájlok tartozhatnak az elsődleges fájlcsoporthoz, vagy felhasználói fájlcsoportokhoz. Ha hiányzik, vagy ON DEFAULT van megadva, akkor az alapértelmezett felhasználói fájlcsoporthoz fog tartozni az adott fájl, amit az ALTER DATABASE utasítással adhatunk meg. Kezdetben az elsődleges fájlcsoport (PRIMARY) az alapértelmezett.
n	A specifikáció n-szer ismételhető.
LOG ON	A napló lemezfájlok közvetlenül lesznek definiálva. Ha hiányzik, akkor a rendszer automatikusan generál egy naplófájlt az adatbázisnév alapján, a telepítéskor meghatározott alapértelmezések alapján.
jelsorrend_név	Az adatbázis alapértelmezett jelsorrendjének beállítását adja. Ha hiányzik, akkor a Server példánynál beállított jelsorrendet használja.
PRIMARY	Megadja, hogy az utasításban szereplő fájlok az *elsődleges fájlcsoporthoz* tartoznak. Kötelezően itt van az *elsődleges adatfájl*, illetve a felhasználói fájlcsoporthoz nem kötött egyéb adatbázis objektumok. Az elsődleges adatfájl tartalmazza az adatbázis logikai kezdetét és a rendszertáblákat.
NAME	A fájlspecifikációban levő fizikai (OS) fájlhoz tartozó logikai név megadását jelöli.
logikai_fájlnév	A Transact-SQL utasításokban használható logikai fájlnév. Az adatbázisban egyedinek kell lennie.
FILENAME	Operációsrendszerbeli fájlnév megadás következik a fájlspecifikációban.

5.19. táblázat.

Táblázat folytatás

Argumentum	Leírás
'OS-fájlnév'	Fizikai fájlnév útvonallal együtt. A mappa nem lehet tömörített. Nyers (formázatlan) partíció esetén a betűje és **:** kerül megadásra (pl.: F☺. Minden fájlt külön partícióra (legalább kettő) kell definiálni. A fájl nem lehet automatikus növekedésű, így nincs értelme MAXSIZE és FILEGROWTH paramétereknek.
SIZE	Az adott fájl méretét definiálhatjuk. Ha hiányzik, akkor elsődleges adatfájl esetén a **model** adatbázisból veszi, másodlagos adatfájl és naplófájl esetén az alapértelmezett 1 MB méretet veszi.
Fájlméret	Az adott fájl kezdeti mérete. A mértékegység lehet: KB, MB, GB, TB. Ha hiányzik a mértékegység, akkor az alapértelmezés szerint megabájt (MB). A minimális méret 512 KB. Az alapértelmezett érték: 1 MB. Az elsődleges adatfájl méretének nagyobb egyenlőnek kell lennie, mint a **model** adatbázis elsődleges adatfájljának mérete.
MAXSIZE	Megadhatjuk a fájl maximális méretét, ha növekedőnek definiáltuk.
Maximális_méret	Az adott fájl maximális mérete. A mértékegység lehet: KB, MB, GB, TB. Ha hiányzik a mértékegység, akkor az alapértelmezés szerint megabájt (MB). Ha nincs megadva, akkor a lemezmeghajtó mérete a korlát.
UNLIMITED	Korlátlanul növekedhet (amíg a lemezmeghajtó nem telik be).
FILEGROWTH	A fájlnövekedés mértékét adhatjuk meg.
Növekmény	A növekmény értéke (az üres hely mennyisége) MB, KB, GB, TB mennyiségi egységben. A 0 érték jelzi, hogy nem növekedhet az adott fájl. Ha a mértékegység helyén % jel van, akkor a növekedés százalékos, az aktuális méret adott százalékával kell megnövelni a fájlméretet. Ha hiányzik a FILEGROWTH paraméter, akkor alapértelmezésben 10%-os a növekedés, de minimum 64 KB.

Az adatbázis *tulajdonosa* (tulajdonos) az a felhasználó, aki létrehozta azt. A tulajdonos megváltoztatható az **sp_changedbowner** tárolt eljárással.

Engedélyek

Az utasítás kiadásához a **sysadmin** és **dbcreator** rögzített szerver szerep tagság szükséges. A **sysadmin** és **securityadmin** rögzített szerver szerep tagok adhatnak egyéb belépőknek (logins) jogot a CREATE DATABASE utasítás kiadásához.

Módosítás

A módosítással az alábbi tevékenységek hajthatók végre egy adatbázisban:

- hozzáadhatunk-, törölhetünk fájlokat, fájlcsoportokat,
- módosíthatjuk fájlcsoportok-, fájlok attribútumait (pl. név, méret),
- módosíthatjuk az adatbázis-, fájlcsoport nevét,
- módosíthatjuk adatfájlok-, naplófájlok logikai nevét,
- megadhatjuk az adatbázis különböző beállításait.

```
ALTER DATABASE adatbázisnév
{ADD FILE <fájlspecifikáció> [ ,...n ] [ TO FILEGROUP fájlcsoport_név | DEFAULT]
| ADD LOG FILE <fájlspecifikáció> [ ,...n ]
| REMOVE FILE logikai_fájlnév
| ADD FILEGROUP fájlcsoport_név
| REMOVE FILEGROUP fájlcsoport_név
| MODIFY FILE <fájlspecifikáció>
| MODIFY NAME = új_adatbázisnév
| MODIFY FILEGROUP fájlcsoport_név {fájlcsoport_tulajdonság | NAME = új_ fájlcsoport_név}
| SET < beállítás_specifikáció> [ ,...n ] [ WITH < megszakítás> ]
| COLLATE < jelsorrend_név>
}
```

Az egyes specifikációk, beállítások leírása az SQL Server 2005 dokumentációban megtalálható.

Törlés

Egy vagy több adatbázist, illetve a hozzájuk tartozó lemezfájlokat törölhetjük a DROP DATABASE utasítással.

```
DROP DATABASE adatbázisnév [ ,...n ]
```

Megjegyzések:

- Használatban levő adatbázis nem törölhető.
- Törlés után a **master** adatbázis mentését elkészíti a rendszer.
- A rendszer adatbázisok nem törölhetők.
- Csak olvasható-írható módban levő adatbázis törölhető.
- Az adatbázist a tulajdonosa, illetve a **sysadmin** és **dbcreator** szerepköri tagok törölhetik.
- A törlési jog nem adható át.

5.6.2. Jelkészletek definíciói

Jelkészlet

```
CREATE CHARACTER SET név [AS] GET bázis_név
        COLLETE jelsorrend_neve |
        COLLATION FROM jelsorrend_forrása];
```

Jelsorrend

```
CREATE COLLATION név FOR jelkészlet_neve
        FROM jelsorrend_forrása [NO PAD] [PAD SPACE];
```

Jelsorrend forrásának megadása

```
DEFAULT  jelsorrend_neve | DESC (jelsorrend_neve) |
        EXTERNAL ('külső_jelsorrend_neve') |
        TRANSLATION leképezés_neve
        [THEN COLLATION jelsorrend_neve];
```

Jelkészlet leképezés

```
CREATE TRANSLATION leképezés_neve FOR bázis_név TO cél_név
        FROM {IDENTITY | leképezés_neve |
            EXTERNAL ('külső_leképezés_neve')};
```

5.6.3. Oszloptípusok definíciói

Az alap adattípusokra építkezve definiálhatunk saját oszloptípust, amit a tábla struktúra megadásakor (CREATE TABLE) tudunk felhasználni. A definícióban megadhatunk alapértelmezett értéket (DEFAULT) és mező ellenőrzési szabályt (CHECK CONSTRAINT).

Létrehozás

```
CREATE DOMAIN oszloptípus_név AS adattípus [DEFAULT állandó]
[CONSTRAINT ...];
```

Módosítás

```
ALTER DOMAIN oszloptípus_név
{SET DOMAIN DEFAULT állandó |
DROP DOMAIN DEFAULT |
ADD CONSTRAINT feltételnév CHECK (...) |
        DROP CONSTRAINT feltételnév};
```

A CHECK szintaktikája:

```
CHECK (oszlopnév {IS NOT NULL | [NOT] IN ( felsorolás ) |
[NOT] BETWEEN alsó korlát AND felső korlát |
reláció {ANY | ALL} ( SELECT oszlop FROM ... ) })
```

Törlés

```
DROP DOMAIN oszloptípus_név;
```

Példa: Hozzunk létre oszloptípust személyi igazolványszám oszlopokhoz!

Megoldás:

```
CREATE DOMAIN azon_szigsz AS CHAR(8)
        CONSTRAINT azon_szigsz_ell
        CHECK(azon_szigsz IS NOT NULL);
```

5.6.4. Adattáblák definíciói

Létrehozás

```
CREATE [{GLOBAL | LOCAL} TEMPORARY] TABLE
[[adatbázis.]tulajdonos.]táblanév
( oszlopnév1 adattípus1 [DEFAULT állandó] [CONSTRAINT ...]
[, CONSTRAINT ...]
[, oszlopnév2 adattípus2 [DEFAULT állandó] [CONSTRAINT ...]
[, CONSTRAINT ...]]
.
[, CONSTRAINT feltételnév1 ...]
[, CONSTRAINT feltételnév2 ...]
…);
```

A TEMPORARY kulcsszó segítségével ideiglenes adattábla hozható létre. Az ideiglenes tábla törlődik az alkalmazás befejezésekor (GLOBAL), vagy a munka (tranzakció) befejezésekor (LOCAL).

Származtatott oszloptípus esetén csak oszlopnév és oszloptípus_név (DOMAIN) adható meg!

Ha a megszorítás több mezőt érint, akkor a CONSTRAINT megadása csak a meződefiníciók után történhet (pl.: összetett index, rekordszintű ellenőrzés). A CONSTRAINT záradékkal az alábbi kényszerek végezhetők:

- NULL kifejezések kizárása (NOT NULL),
- engedélyezett értékek megadása vagy mások kizárása (CHECK (feltétel)),

- egyedi kulcsok definiálása:

```
CONSTRAINT feltételnév
[PRIMARY KEY | UNIQUE] (oszlopnév1 [, oszlopnév2, ...])
```

- idegen kulcsok definiálása:

```
CONSTRAINT feltételnév
FOREIGN KEY (oszlopnév1[, oszlopnév2, ...])
      REFERENCES táblanév
[ON {DELETE|UPDATE}{CASCADE|SET NULL|
       SET DEFAULT|NO ACTION}]
```

- beviendő oszlopérték, vagy abból származtatott kifejezés előfordulásának vizsgálata más helyeken:

```
CONSTRAINT feltételnév
CHECK (kifejezés1 [NOT] {IN | reláció {ANY | ALL}}
       (SELECT kifejezés2 FROM ... ) )
INITIALLY {IMMEDIATE [NOT DEFERRABLE] | DEFERRED}
```

vagy

```
CONSTRAINT feltételnév
CHECK ([NOT] EXISTS (SELECT ... FROM ... ) ) ...
```

Az ellenőrzés hatályba lépése (INITIALLY) történhet azonnal (IMMEDIATE), vagy a tranzakció végén (DEFERRED). A NOT DEFERRABLE tiltja a DEFERRED használatát.

Az alábbi utasítások létrehozzák a példa adatbázis tábláit. Csak a mezőneveket, adattípusokat, NOT NULL kényszert, DEFAULT értéket adtuk meg. Az egyéb beállításokat módosítással hajtjuk végre.

```
CREATE TABLE Eszkozok
(
        EKOD char(1) NOT NULL ,
        ENEV var charying (20) NOT NULL ,
        EDIJ smallint NOT NULL ) ;
CREATE TABLE Ugyfelek
(
        USZIGSZ char(8) NOT NULL ,
        UNEV var charying(30) NOT NULL ,
        UIRSZ char(4) DEFAULT ” ,
        UTELEP var charying(20) NOT NULL ,
        UUTCA var charying(25) NULL ) ;

CREATE TABLE Kolcsfej
(
        KAZON integer NOT NULL ,
        KUSZIGSZ char(8) NOT NULL ,
        KDATUM date NOT NULL ,
        KKIDO char(5) NOT NULL ) ;

CREATE TABLE Kolcstet
(
        KAZON integer NOT NULL ,
        KEKOD char(1) NOT NULL ,
        KTARTAM smallint NOT NULL ,
        KMENNY smallint NULL ,
        KRENDB bit NOT NULL ) ;
```

Módosítás

ALTER TABLE opciók;
Lehetőségek:

- oszlop adattípus módosítás,
- oszlop törlés (DROP COLUMN),
- új oszlop definiálás (ADD COLUMN),
- új értékvizsgálatok (ADD CONSTRAINT),
- új épségi vizsgálatok (ADD CONSTRAINT),
- értékvizsgálat törlés (DROP CONSTRAINT),
- épségi vizsgálat törlés (DROP CONSTRAINT),

- új alapértelmezési érték megadása (SET DEFAULT),
- alapértelmezési érték törlése (DROP DEFAULT),
- NULL-kifejezés engedélyezése (NULL),
- NULL-kifejezés tiltása (NOT NULL),
- tárolás körülményeinek módosítása.

Példáinkban az MS SQL Server 2005 táblastruktúra módosító utasítását használjuk.

Példa: Definiálja az előző adattáblák primary key megszorítását MS SQL Server 2005 szintaktikát használva!

Megoldás:

```
ALTER TABLE Eszkozok
ADD CONSTRAINT PK_Eszkozok PRIMARY KEY (EKOD);

ALTER TABLE Ugyfelek
ADD CONSTRAINT PK_Ugyfelek PRIMARY KEY (USZIGSZ);

ALTER TABLE Kolcsfej
ADD CONSTRAINT PK_Kolcsfej PRIMARY KEY (KAZON);

ALTER TABLE Kolcstet ADD CONSTRAINT PK_Kolcstet
PRIMARY KEY (KAZON,  KEKOD);
```

Példa: Definiálja az előző adattáblák foreign key megszorítását!

Megoldás:

```
ALTER TABLE Kolcsfej ADD
        CONSTRAINT FK_Kolcsfej_Ugyfelek FOREIGN KEY
        (KUSZIGSZ) REFERENCES Ugyfelek
    ON DELETE CASCADE  ON UPDATE CASCADE;
```

```
ALTER TABLE Kolcstet ADD
        CONSTRAINT FK_Kolcstet_Eszkozok FOREIGN KEY
        (KEKOD) REFERENCES Eszkozok
    ON DELETE CASCADE  ON UPDATE CASCADE ,
        CONSTRAINT FK_Kolcstet_Kolcsfej FOREIGN KEY
        (KAZON) REFERENCES Kolcsfej
    ON DELETE CASCADE  ON UPDATE CASCADE;
```

Példa: Definiálja az előző adattáblák check megszorításit!

Megoldás:

```
ALTER TABLE Eszkozok ADD
        CONSTRAINT CK_Eszkozok_edij CHECK  (EDIJ > 50),
       CONSTRAINT CK_eszkozok_ekod CHECK  (EKOD >= '1'
                                          AND EKOD <= '4');

ALTER TABLE Ugyfelek ADD
        CONSTRAINT CK_ugyfelek_uirsz
CHECK (UIRSZ LIKE '[0-9][0-9][0-9][0-9]' OR uirsz ='),
        CONSTRAINT CK_ugyfelek_uszigsz
      CHECK (USZIGSZ LIKE '[A-Z][A-Z][0-9][0-9][0-9][0-9][0-9][0-9]'
          OR USZIGSZ LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][A-Z][A-Z]');
ALTER TABLE Kolcsfej ADD
        CONSTRAINT CK_kolcsfej_kkido CHECK
(KKIDO LIKE '[0-9][0-9]:[0-9][0-9]' AND (LEFT(KKIDO, 2) >= '09'
AND LEFT(KKIDO, 2) <= '19') AND (RIGHT(KKIDO, 2) >= '00'
AND RIGHT(KKIDO, 2) <= '59'));
```

Törlés

```
DROP TABLE táblanév;
```

5.6.5. Önálló feltételek

Táblák közötti összefüggések ellenőrzésére szolgál.

Létrehozás

```
CREATE ASSERTION név CHECK ([NOT] feltétel );
```

Törlés

```
DROP ASSERTION név;
```

Feltételek hatálybaléptetése

```
SET CONSTRAINTS {ALL | lista} {DEFERRED | IMMEDIATE};
```

5.6.6. Indexek

Egyszerű vagy összetett indexeket hozhatunk létre a táblázatainkon. A rendezés iránya lehet növekvő (ASC), vagy csökkenő (DESC). Az alapértelmezés: csökkenő.

Létrehozás

```
CREATE [UNIQUE] INDEX indexnév ON táblanév
 (oszlop1 [{ASC | DESC}] [, oszlop2 ...]);
```

Példa: Hozza létre a példa adatbázis általános indexeit!

Megoldás:

CREATE INDEX IX_Ugyfelek_unev ON Ugyfelek (UNEV);

CREATE INDEX IX_Kolcsfej_kdatum ON Kolcsfej (KDATUM DESC);

Törlés

```
DROP INDEX indexnév;
```

5.6.7. Szinonimák

Másodlagos nevet definiálhatunk adattáblához. Ezeket a neveket, ugyanúgy használhatjuk, mint a táblaneveket.

Létrehozás

```
CREATE SYNONYM szinonima_név
        FOR [[adatbázis.]tulajdonos.]tábla_név;
```

Törlés

```
DROP SYNONYM szinonima_név;
```

5.6.8. Nézettáblák

Adattáblákra, korábban definiált nézetekre építhetünk új nézetet. Az adatbázis a CREATE VIEW utasítást tárolja. Az utasítás SELECT záradéka mutatja, hogy a nézetet hogyan kell benépesíteni adatokkal. Nézeten keresztül az alaptáblák adatait is módosíthatjuk (INSERT, UPDATE, DELETE).

Létrehozás

```
CREATE VIEW [[adatbázis.]tulajdonos.]nézettábla_név AS SELECT ...;
```

Példa: Hozza létre a kölcsonzfej nevű nézetet, amely a kolcsfej és ugyfelek adattáblák alapján megadja a kölcsönzéshez tartozó ügyfél adatokat is! A kolcsfej tábla minden sora jelenjen meg, azok is, amelyekhez nincs kapcsolódó rekord az ugyfelek táblában!

```
CREATE VIEW kolcsonzfej AS
SELECT Kolcsfej.KAZON, Kolcsfej.KUSZIGSZ,
        Ugyfelek.UNEV, Ugyfelek.UIRSZ,.Ugyfelek.UTELEP,
        Kolcsfej.KDATUM, Kolcsfej.KKIDO
        FROM Kolcsfej LEFT OUTER JOIN Ugyfelek
                ON Kolcsfej.KUSZIGSZ = Ugyfelek.USZIGSZ;
```

Példa: Hozza létre a tetertek nevű nézetet, amely a kolcstet és eszkozok adattáblák mezői alapján megadja a tétel kölcsönzési értékét is!

Megoldás:

```
CREATE VIEW tetertek AS
SELECT Kolcstet.KAZON AS Azonosító,
    Eszkozok.EKOD AS Eszközkód,
    Kolcstet.KTARTAM AS [Tartam (óra)],
    Kolcstet.KMENNY AS [Mennyiség (db)],
    Eszkozok.EDIJ AS [Kölcsönzési díj (Ft/db/óra)],
    Kolcstet.KTARTAM * Kolcstet.KMENNY*Eszkozok.EDIJ AS Érték
        FROM  Eszkozok INNER JOIN   Kolcstet
                ON Eszkozok.EKOD =Kolcstet.KEKOD;
```

Törlés

```
DROP VIEW [[adatbázis.]tulajdonos.]nézettábla_név ;
```

5.6.9. Adatbázissémák

A CREATE SCHEMA utasítással teljes adatbázist, illetve al sémákat definiálhatunk. Megadhatjuk a felhasználók bejelentkezési, hitelesítési paramétereit, az adatbázisból mit láthatnak, kezelhetnek, stb.

Létrehozás

```
CREATE SHEMA séma_név
        [AUTHORIZATION felhasználói_név/jelszó]
        [DEFAULT CHARACTER SET jelkészlet_név]
        {CREATE, ALTER, ... utasítások};
```

5.7. Adat felügyeleti utasítások

5.7.1. Hozzáférések szabályozása

Egy adatbázis adatainak eléréséhez először a felhasználónak kapcsolódnia kell az adatbázishoz. A felhasználó ismert kell, hogy legyen az adatbázisban. Az adatbázis kitüntetett felhasználója, a rendszergazda adminisztrálja a felhasználókat. A kapcsolatot létrehozó utasítás:

```
CONNECT TO <adatbázis> [USER <felhasználó>] ;
```

Egyszerre több adatbázishoz is csatlakozhatunk. Aktív adatbázis kiválasztása:

```
SET CONNECTION {DEFAULT | <adatbázisnév>};
```

A kapcsolat megszüntetése:

```
DISCONNECT {<adatbázis> | ALL | CURRENT};
```

5.7.2. A felhasználói jogok szabályozása

Jogokat adhatunk az adatbázis egészére, illetve az objektumokon végrehajtható különböző műveletekre:

```
GRANT <műveleti jog> ON <objektum> TO {PUBLIC | <felh_1>
[, <felh_2> ...]} [WITH GRANT OPTION];
```

vagy

```
GRANT <adatbázis jog> TO {PUBLIC | <felh_1> [, <felh_2> ...]}
```

Az objektum lehet:

- CHARACTER SET
- COLLATION
- DOMAIN
- TABLE
- TRANSLATION
- tárolt eljárás

A műveleti jog lehet:

- ALL [PRIVILEGES]
- ALTER
- DELETE
- EXECUTE – adatbázis eljárás hívása
- INDEX
- INSERT
- SELECT [(oszlop_1[, oszlop_2 ...])]
- UPDATE [(oszlop_1[, oszlop_2 ...])]
- USAGE karakterkészlet használata

Az adatbázis egészére vonatkozó jogok:

CONNECT	kapcsolódás, adatkezelési utasítások
RESOURCE	objektumdefiniálás, megszüntetés, erőforrás kezelés
DBA	adatbázis egészére vonatkozó műveletek: jogok adása, megvonása, adatbázis megszüntetése

A korábban kiosztott jogokat a REVOKE utasítással vehetjük vissza:

```
REVOKE <műveleti jog> ON <objektum> FROM {PUBLIC | <felhasználónév_1> [, <felhasználónév_2> ...]};
```

vagy

```
REVOKE <adatbázis jog> FROM {PUBLIC | <felhasználónév_1>
 [, <felhasználónév_2> ...]};
```

5.7.3. Hozzáférési zárak kezelése

A felhasználók is létrehozhatnak zárakat kizárólagos (EXCLUSIVE), vagy osztott (SHARE) használatra adattáblákon, illetve a kiadott zárat megszüntethetik:

```
LOCK TABLE [[adatbázis.]tulajdonos.]táblanév
        IN {SHARE | EXCLUSIVE} MODE;
UNLOCK TABLE [[adatbázis.]tulajdonos.]táblanév;
```

5.7.4. Tranzakciók kezelése

Korábbi szintaktika:

```
BEGIN [WORK];
...
COMMIT [WORK];
...
ROLLBACK [WORK];
```

Szabványos (SQL2) szintaktika

```
...
[CONNECT ...]
...
COMMIT [WORK];
...
ROLLBACK [WORK];
...
```

A tranzakciók nem ágyazhatók egymásba a szintaktika szerint! A kapcsolódás elindítja az első tranzakciót, majd az érvényesítés (COMMIT) zárja, és azonnal indul a következő tranzakció. Hiba esetén jöhet a visszagörgetés (ROLLBACK). Hogy kevesebbet kelljen visszagörgetni meghibásodások esetén, kijelölhetünk olyan pontokat (SAVEPOINT), ahol még minden rendben volt.

A visszamenőleges érvényesítés határpontjainak kijelölése

```
SAVEPOINT <név>;
ROLLBACK [WORK] TO <mentésipont_név>;
```

A változtató (módosító) utasításokat a SAVEPOINT mögé csoportosítjuk, majd vizsgálat következik. Ha hiba van, akkor javítunk, majd visszaállítunk, és újból végrehajtjuk az utasításokat. Ha minden rendben, akkor érvényesítés, COMMIT.

A tranzakciók naplózása biztosítja a visszagörgetési lehetőséget.

Tranzakciók lefutása és a változtatások láthatósága

Az egyszerre futó tranzakciók elválasztási szintjének beállítására a SET TRANSACTION utasítás szolgál. Szintaktika:

```
SET TRANSACTION <elhatárolási szint> [, <hozzáférés jellege>];
```

Elhatárolási szint

- READ UNCOMMITTED – olvasás érvényesítés nélkül: az olvasott sorokat mások nem változtathatják meg

- READ COMMITTED – érvényesített olvasás: érvényesítésig mindenki csak a saját változatát látja
- REPEATABLE READ – ismételhető olvasás: 2 változat van
- SERIALIZABLE – sorba rendezhető (SQL2 alapértelmezés)

Hozzáférés jellege

- READ ONLY
- READ WRITE (törlés is)

Piszkos olvasás (DIRTY READ) esetén nincs zár, bárki, bármikor hozzáférhet az adatokhoz!

5.8. Ellenőrző kérdések

1. Ismertesse a NULL-érték fogalmát, használatát!
2. Adja meg a NULL-értékkel bővített logikai AND művelet igazságtáblázatát!
3. Adja meg a NULL-értékkel bővített logikai OR művelet igazságtáblázatát!
4. Adja meg a NULL-értékkel bővített logikai NOT művelet igazságtáblázatát!
5. Sorolja fel az SQL2 objektumait!
6. Ismertesse a következő SQL2 objektumokat: adatséma, adattípus, oszloptípus!
7. Ismertesse a következő SQL2 objektumokat: jelkészlet, jelsorrend, jelkészlet leképezés!
8. Ismertesse a következő SQL2 objektumokat: adattábla, indextábla, nézettábla!
9. Ismertesse a következő SQL2 objektumokat: adatérték szabály, hivatkozási függőségi szabály!
10. Csoportosítsa az alapvető SQL2 utasításokat!
11. Ismertesse a katalógus szerepét!
12. Ismertesse az egytáblás lekérdezés lehetőségeit!
13. Ismertesse a többtáblás lekérdezés lehetőségeit!
14. Ismertesse az általánosan használható szövegkezelő SQL2 belső függvényeket!
15. Ismertesse az általánosan használható dátum- és időkezelő SQL2 belső függvényeket!
16. Ismertesse az SQL2 alapvető összesítő függvényeit!

17. Ismertesse a WHERE záradék használatát!
18. Ismertesse a GROUP BY záradék használatát!
19. Ismertesse a HAVING záradék használatát!
20. Ismertesse az ORDER BY záradék használatát!
21. Ismertesse az alkérdések (al-SELECT) használatát!
22. Ismertesse röviden az SQL2 adat karbantartási utasításait!
23. Ismertesse röviden az SQL2 adatdefiníciós utasításait!
24. Ismertesse az SQL2 megszorítások definiálását, használatát!
25. Ismertesse röviden az SQL2 adat felügyeleti utasításait!

6. Lekérdezések feldolgozása

6.1. A lekérdezés fordítás áttekintése

A felhasználó lekérdezéseit, adatmódosító utasításait a **lekérdezés feldolgozó** *lefordítja* adatbázis-műveletekre, és végre is hajtja a műveleteket. A lekérdezés fordítás három fontosabb lépésre osztható fel:

a) **Elemzés**: a lekérdezést és annak szerkezetét jellemző *elemző fát* építünk fel.

b) **Lekérdezés átírás**: az elemző fát átkonvertáljuk egy kezdeti lekérdezés tervvé, amely rendszerint a lekérdezésnek egy *algebrai* megvalósítása. A kezdeti tervet később átalakítjuk egy olyan ekvivalens tervvé, amelynek végrehajtási költsége várhatóan kisebb lesz.

c) **Fizikai terv előállítása**: a *b)* pontban megkapott *logikai lekérdezés tervnek* átalakítjuk *fizikai lekérdezés tervvé*. A logikai terv valamennyi operátorának megvalósítására kiválasztunk egy algoritmust, és meghatározzuk ezen operátorok végrehajtási sorrendjét. A fizikai terv a végrehajtáshoz tartozó információkat is tartalmaz pl.: a relációkhoz történő hozzáférés, relációt kell-e rendezni stb.

A *b)* és *c)* részeket együtt gyakran nevezzük *lekérdezés optimalizálásnak*.

6.2. Kifejezésfák

Az operátorok egymás utáni alkalmazását egy *kifejezésfa* formájában rajzolhatjuk fel. A fa *leveleit* relációk nevei alkotják, és a belső csúcsokat olyan operátorok alkotják, amelyek akkor nyernek értelmet, amikor alkalmazzuk a gyermeke vagy gyermekei által reprezentált relációkra.

Példa: Tegyük fel, hogy adottak a Kiküldetések adatbázis alábbi relációi. Készítsük el a kifejezésfát ahhoz a lekérdezéshez, amely megjeleníti a ’0001’ azonosítóval bíró dolgozó közlekedési költségű (tkod = 1) kiküldetéseit közlekedési eszköz rendezettségben!

Dolgozok (<u>dkod</u>, dnev, dirsz, dtelep, dutca)

Kikuld (<u>kikaz</u>, *dkod*, kikhely, kikdat, ekod, kikcel)

KtgTipus (tkod, tnev)

Koltseg (*kikaz*, *tkod*, km, osszeg)

Az aláhúzott attribútum elsődleges kulcsot, a dőlt betűtípusú idegen kulcsot jelöl az adott relációban. A Koltseg reláció összetett elsődleges kulcsát a két idegen kulcsú attribútum együtt adja. Egy kiküldetéshez egy adott típusú költségből csak egy sor tartozhat.

Az SQL lekérdezés:

```
SELECT D.dkod, D.dnev, Ki.kikhely, Ki.kikdat, Ki.ekod, Ki.kikcel
        FROM Dolgozok D, Kikuld Ki, Koltseg Ko
        WHERE D.dkod = Ki.dkod AND Ki.kikaz=Ko.kikaz
                AND D.dkod = '0001' AND Ko.tkod = 1
        ORDER BY Ki.ekod;
```

Az SQL-lekérdezést az *elemző* logikai lekérdezés tervvé alakítja, amely a következő lépéseket tartalmazza:

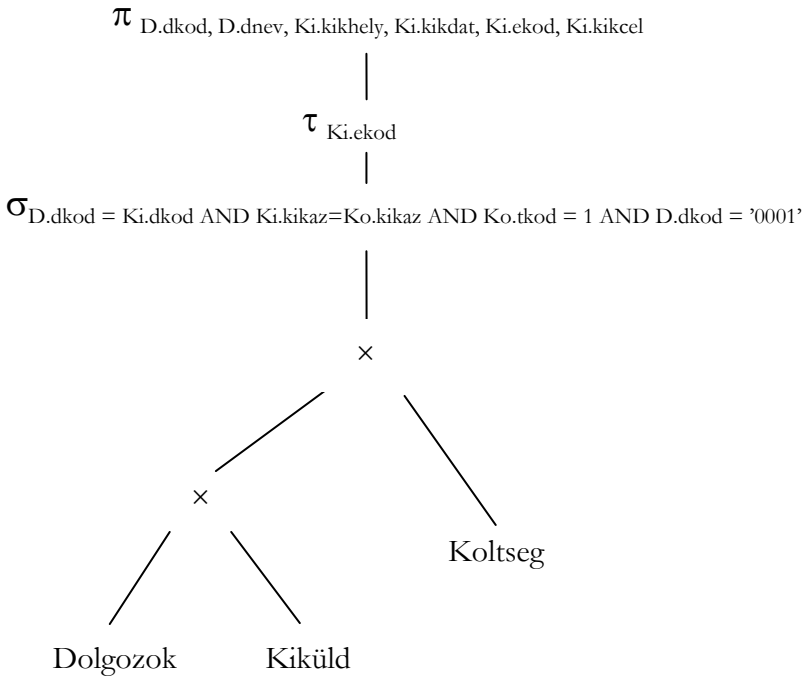
Első lépés: a FROM utáni relációk összekapcsolása a Descartes-szorzat operátor felhasználásával.

Második lépés: a WHERE záradéknak megfelelő kiválasztás végrehajtása.

Harmadik lépés: a rendezés végrehajtása.

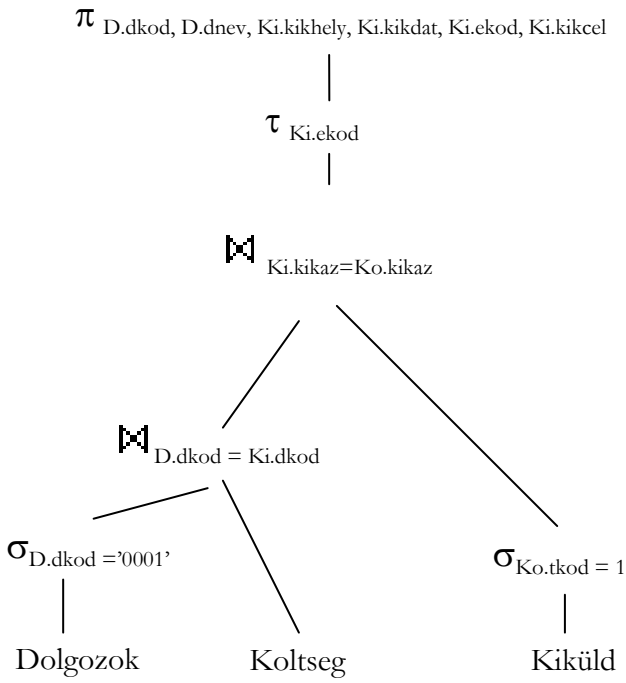
Negyedik lépés: vetítés a SELECT záradékban szereplő listára.

A fenti lekérdezésnek megfelelő kifejezésfát a 6.1. ábra mutatja:



6.1. ábra. Logikai lekérdezés terv

A **6.2. ábra** egy kiválasztás és egy szorzat összekapcsolássá történő átalakítását alkalmaztuk. Az összekapcsolások általában kevesebb sort eredményeznek.



6.2. ábra. Módosított logikai lekérdezés terv

A WHERE záradékban szereplő két feltételt szétválasztottuk két σ-műveletté, és a műveleteket lejjebb „csúsztatjuk" a fában, a megfelelő relációkig. Célszerű a kiválasztást minél előbb elvégezni, amivel a relációk mérete jelentősen csökkenhet.

6.3. Algebrai szabályok lekérdezés tervek javítására

6.3.1. Kommutatív és asszociatív szabályok

A különféle kifejezések egyszerűsítésére használt legáltalánosabb szabályok a *kommutatív* és *asszociatív* szabályok. Egy operátorra vonatkozó *kommutatív* szabály azt mondja ki, hogy nem számít, hogy milyen sorrendben adjuk meg az operátor argumentumait, az eredmény ugyanaz lesz. Pl. a +

és × művelet kommutatív, mert x + y = y + x és x × y = y × x tetszőleges x és y esetén.

Egy operátorra vonatkozó *asszociatív* szabály azt mondja ki, hogyha az operátort kétszer használjuk, akkor egyaránt csoportosíthatunk balról vagy jobbról. A + és x például asszociatív aritmetikai operátorok, ami azt jelenti, hogy (x + y) + z = x + (y + z) és (x × y) × z = x × (y × z). A relációs algebra néhány operátora egyszerre *kommutatív* és *asszociatív*:

$R \times S = S \times R$; $(R \times S) \times T = R \times (S \times T)$,

$R \bowtie S = S \bowtie R$; $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$,

$R \cup S = S \cup R$; $(R \cup S) \cup T = R \cup (S \cup T)$,

$R \cap S = S \cap R$; $(R \cap S) \cap T = R \cap (S \cap T)$.

Az *egyesítésre* és a *metszetre* vonatkozó szabályok egyaránt érvényesek halmazokra és multihalmazokra.

6.3.2. Kiválasztással kapcsolatos szabályok

A kiválasztások lényegesen csökkenthetik a *relációk méretét*, ezért a kiválasztásokat vigyük lefelé a kifejezésfában. Ha egy kiválasztás feltétele összetett (azaz AND vagy OR által összekapcsolt feltételekből áll), akkor a feltételt szétvágjuk. A kiválasztásra vonatkozó első két szabályt *szétvágási szabálynak* nevezzük:

$\sigma_{F1 \text{ AND } F2}(R) = \sigma_{F1}(\sigma_{F2}(R))$

$\sigma_{F1 \text{ OR } F2}(R) = (\sigma_{F1}(R)) \cup_H (\sigma_{F2}(R))$

A **σ** operátor tetszőleges sorozata esetén a sorrend felcserélhető:

$\sigma_{F1}(\sigma_{F2}(R)) = \sigma_{F2}(\sigma_{F1}(R))$

A kiválasztásokat a szorzat, egyesítés, metszet, különbség és összekapcsolás bináris operátorokon áttolhatjuk. Háromféle szabály van, attól függően, hogy opcionális vagy kötelező a kiválasztást az egyes argumentumokhoz odavinni:

1. *Egyesítés* esetén a kiválasztást mindkét argumentumra alkalmazni kell.

2. *Különbség* esetén a kiválasztást az első argumentumra alkalmazni kell, a másodikra pedig lehet.
3. A többi operátor esetében csak azt követeljük meg, hogy a kiválasztást egy argumentumra alkalmazzuk.

Az *egyesítésre* vonatkozó szabály:

$$\sigma_F(R \cup S) = \sigma_F(R) \cup \sigma_F(S)$$

Itt kötelezően le kell vinni a kiválasztást a fa mindkét ágán.

A *különbségre* vonatkozó szabály egyik változata:

$$\sigma_F(R \setminus S) = \sigma_F(R) \setminus S$$

Az is megengedett azonban, hogy mindkét argumentumhoz odaviszszük a kiválasztást:

$$\sigma_F(R \setminus S) = \sigma_F(R) \setminus \sigma_F(S)$$

A következő szabályoknál feltételezzük, hogy az **R** relációban megvan az összes F-ben szereplő attribútum.

$$\sigma_F(R \times S) = \sigma_F(R) \times S$$
$$\sigma_F(R \bowtie S) = \sigma_F(R) \bowtie S$$
$$\sigma_F(R \bowtie_F S) = \sigma_F(R) \bowtie_F S$$
$$\sigma_F(R \cap S) = \sigma_F(R) \cap S$$

Ha az F-ben csak **S**-beli attribútumok szerepelnek, akkor:

$$\sigma_F(R \times S) = R \times \sigma_F(S).$$

Hasonlóan írhatók át a $\bowtie$, $\bowtie_F$ és $\cap$ operátorok szabályai.

Ha az **R** és **S** relációk mindegyikében szerepel az összes F-beli attribútum, akkor használható az alábbi szabály:

$$\sigma_F(R \bowtie S) = \sigma_F(R) \bowtie \sigma_F(S)$$

Nem alkalmazható ilyen szabály, ha az operátor × vagy $\bowtie_F$, ezekben az esetekben ugyanis **R**-nek és **S**-nek nincsenek közös attribútumai. A $\cap$ esetében a szabály mindig érvényes lesz, hiszen ekkor az **R** és **S** sémája ugyanaz kell, hogy legyen.

6.3.3. Vetítéssel kapcsolatos szabályok

A kiválasztáshoz hasonlóan a vetítéseket is „tolhatjuk lefelé", át más operátorokon. A vetítések tologatása abban különbözik a kiválasztások tologatásától, hogy amikor vetítést tolunk, akkor a vetítés általában ott is megmarad, ahol van. A vetítés „tolása" valójában egy új vetítés bevezetését jelenti valahol a létező vetítés alatt.

6.3.4. Összekapcsolásra és szorzatra vonatkozó szabályok

Közvetlenül az összekapcsolás definíciójából következő szabályok:

- $R \bowtie_F S = \sigma_F(R \times S)$
- $R \bowtie S = \pi_L(\sigma_F(R \times S))$, ahol **F** az a feltétel, amely az **R**-ből és **S**-ből származó azonos nevű attribútum párok egyenlőségét vizsgálja, az **L** pedig olyan lista, amely tartalmazza az összes egyenlővé tett attribútum pár egyikét, valamint az **R** és **S** minden maradék attribútumát.

6.3.5. Ismétlődések elhagyására vonatkozó szabályok

Sok operátoron keresztül lehet tolni a δ operátort. A δ lefelé történő mozgatása a fában csökkenti a köztes relációk méretét. A δ néha olyan helyre vihető, ahol elhagyható, mert a relációra nem tartalmaz ismétlődéseket:

- $\delta(R) = R$, ha **R**-ben nincsenek ismétlődések. Ilyen fontos esetekről van szó például, ha **R** a következő:

a) Egy reláció elsődleges kulccsal.

b) Egy γ művelet eredményeként kapott reláció.

Az alábbi néhány szabály a δ operátort más operátorokon „tolja" keresztül:

- $\delta(R \times S) = \delta(®\times \delta(S)$
- $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
- $\delta(R \bowtie_F S) = \delta(R) \bowtie_F \delta(S)$

• $\delta(\sigma_F(R)) = \sigma_F(\delta(R))$

A δ odavihető egy *metszet* egyik vagy mindkét argumentumához is:

• $\delta(R \cap_M S) = \delta(R) \cap_M S = R \cap_M \delta(S) = \delta(R) \cap_M \delta(S)$

A $\boldsymbol{\delta}$ általában nem vihető át a $\cup_M$, $\setminus_M$ vagy $\boldsymbol{\pi}$ operátorokon.

6.3.6. Csoportosításra és összesítésre vonatkozó szabályok

A transzformáció alkalmazhatósága a használt összesítő operátor részleteitől függ. Emiatt kevés általános szabály van:

• $\delta(\gamma_L(R)) = \gamma_L(R)$

• $\gamma_L(R) = \gamma_L(\pi_M(R))$, ahol M az **R** azon attribútumainak listája, amelyek L-ben előfordulnak.

Egy γ_L operátort ismétlődés érzéketlennek nevezünk, ha L-ben csak MIN és/vagy MAX összesítések szerepelnek. Ha γ_L ismétlődés érzéketlen, akkor:

• $\gamma_L(R) = \gamma_L(\delta(®.$

6.4. Ellenőrző kérdések

1. Ismertesse a lekérdezés fordítás lépéseit!
2. Adja meg a kifejezésfa fogalmát!
3. Mit jelent a kiválasztás művelet lejjebb csúsztatása?
4. Mit jelent a vetítés művelet csúsztatása?
5. Ismertesse a kommutatív és asszociatív szabályok használatát!
6. Ismertesse a kiválasztással kapcsolatos szabályokat!
7. Ismertesse a vetítéssel kapcsolatos szabályokat!
8. Ismertesse az összekapcsolásra és szorzatra vonatkozó szabályokat!
9. Ismertesse az ismétlődések elhagyásának szabályait!
10. Ismertesse a csoportosításra és összesítésre vonatkozó szabályokat!

7. Tranzakció-kezelés

A *tranzakció-kezelő* tipikus feladatai:

- **Több felhasználós működés (konkurenciavezérlés)**: egyidejű hozzáférést kell biztosítani több felhasználónak úgy, hogy az adatbázis konzisztens maradjon.
- **Rendszerhibák utáni helyreállítás biztosítása**: hiba esetén a *helyreállítás-kezelő* az adatbázist újra konzisztens állapotba hozza a naplóbejegyzések segítségével.

7.1. Konkurenciavezérlés

A *lekérdezés-feldolgozó* a felhasználók SQL utasításait elemi utasítások sorozatává alakítja, pl.:

1. felhasználó: $u_1, u_2, \ldots, u_{10}$
2. felhasználó: $v_1, v_2, \ldots, v_{10}$

A végrehajtás során a két utasítássorozat nem elkülönítve (egymás után) hajtódik végre, hanem összefésülődnek pl. az alábbi módon:

$u_1, v_1, v_2, u_2, u_3, v_3, \ldots, v_{10}, u_{10}$

A saját sorrend megmarad mindkettőn belül. Így lesz lehetséges a több felhasználó egyidejű kiszolgálása. Kialakulhat olyan állapot, ami nem jött volna létre, ha egymás után futnak le a tranzakciók. Például, legyen mindkét felhasználó feladata ugyanaz: olvassunk be a memóriába a háttértárról egy numerikus adattételt az **A** változóba, növeljük meg az **A** értékét 1-gyel, majd írjuk ki a háttértárra. A végrehajtandó feladat:

1. felhasználó: Read A, A=A+1, Write A

2. felhasználó: Read A, A=A+1, Write A

Ha az összefésülés az alábbi (az index a felhasználó sorszámot jelöli):

$(\text{Read A})_1, (\text{Read A})_2, (\text{A=A+1})_1, (\text{A=A+1})_2, (\text{Write A})_1, (\text{Write A})_2$

akkor csak 1-gyel nő az A értéke, míg egymás utáni végrehajtás esetén kettővel nőtt volna.

A *konkurenciavezérlés* biztosítja az **ütemezőn** keresztül, hogy az egyidejűleg végrehajtandó tranzakciók megőrizzék konzisztenciájukat. Az ütemező fogadja a *tranzakció-kezelő* olvasási/írási kérését. Az ütemező olvassa/írja a kért adattételt a pufferből. Szükség esetén utasítja a *pufferkezelőt* az adattétel pufferbe hozására. Az ütemező késleltetheti az adott kérés végrehajtását, megszakíthatja a problémás tranzakciót.

7.1.1. Soros és sorba rendezhető (serializability) ütemezések

A *korrektségi-elv* alapján, ha a tranzakciókat egymástól elkülönítve futtatjuk, akkor az adatbázist konzisztens állapotból konzisztens állapotba alakítjuk. A gyakorlatban egyidejűleg több tranzakció fut. Végrehajtásukat úgy kellene „ütemezni", hogy olyan eredményt kapjunk, mintha a tranzakciókat külön-külön hajtottuk volna végre.

Ütemezések

Az *ütemezés* a tranzakciók által végrehajtott műveletek időrendi megadása. Feltesszük, hogy a lényeges olvasási és írási műveletek a központi memória puffereiben történnek. A pufferbe behozott adatbáziselemekhez más tranzakciók is hozzáférhetnek. A READ és WRITE műveletek a puffer tartalmára vonatkoznak, szükség esetén meghívják az adatbeviteli (INPUT), illetve az adatkiírási (OUTPUT) utasításokat is. Konkurenciavizsgálatnál csak a READ és WRITE műveletek számítanak.

Soros ütemezések

Egy ütemezés *soros*, ha bármely két T_1 és T_2 tranzakcióra: ha T_1-nek van olyan művelete, amely megelőzi a T_2 valamelyik műveletét, akkora T_1 összes művelete megelőzi a T_2 minden műveletét. Minden soros ütemezés megőrzi az adatbázis-állapot konzisztenciáját (tranzakció korrektségi-elv).

Az alábbi táblázatban a T_1 tranzakció megelőzi a T_2 tranzakciót, az **A** adatelem kezdőértéke **x**, a **B** adatelemé **y**.

T_1	T_2	A	B
		x	y
Read(A, t)			
t := t + 100		x +100	
Write(A, t)			

7.1. táblázat.

Táblázat folytatás

T_1	T_2	A	B
Read(B, t)			
t := t + 100			
Write(B, t)			y +100
	Read(A, s)		
	s := 2 * s		
	Write(A, s)	2*(x +100)	
	Read(B, s)		
	s := 2 * s		
	Write(B, s)		2*(y +100)

7.1. táblázat.

A Read(A, t) művelet beolvassa pufferből (esetleg háttértárról) az A adattételt a t változóba. Az ütemezés jelölése: (T_1, T_2).

Sorba rendezhető ütemezések

Egy ütemezés *sorba rendezhető* (sorosítható), ha ugyanolyan hatással van az adatbázis állapotára, mint valamely soros ütemezés, függetlenül az adatbázis kezdeti állapotától.

Az alábbi táblázat sorba rendezhető (de nem soros) ütemezést mutat:

T_1	T_2	A	B
		x	y
Read(A, t)			
t := t + 100		x +100	
Write(A, t)			
	Read(A, s)		
	s := 2 * s		
	Write(A, s)	2*(x +100)	
Read(B, t)			
t := t + 100			
Write(B, t)			y +100
	Read(B, s)		
	s := 2 * s		
	Write(B, s)		2*(y +100)

7.2. táblázat.

A tranzakciók és ütemezések egyszerűbb jelölése

Egy tranzakció által végrehajtott számítások tetszőlegesek lehetnek, ezért nem szükséges a helyi számításokat megadni, csak a tranzakciók által végrehajtott olvasások és írások számítanak. Ezért a tranzakciókat és az ütemezéseket rövidebben jelölhetjük. Az $r_T(X)$ és $w_T(X)$ tranzakció műveletek azt jelentik, hogy a T tranzakció olvassa, illetve írja az X adatbáziselemet. Az $r_i(X)$ és $w_i(X)$ jelölje ugyanazt, mint $r_{Ti}(X)$, illetve $w_{Ti}(X)$.

Az előző tranzakciók az alábbi módon írhatók fel:

T_1: $r_1(A)$; $w_1(A)$; $r_1(B)$; $w_1(B)$;

T_2: $r_2(A)$; $w_2(A)$; $r_2(B)$; $w_2(B)$;

7.1.2. A sorba rendezhetőség biztosítása zárakkal

Az *ütemező* feladata az olyan műveleti sorrendek megakadályozása, amelyek nem sorba rendezhető ütemezésekhez vezetnek. A tranzakciók *zárolják* azokat az adatbáziselemeket, amelyekhez hozzáférnek, hogy biztosítsák a sorba rendezhetőséget.

Tegyük fel, hogy egy zárolási sémában csak egyféle zár van, amelyet a tranzakcióknak meg kell kapniuk az adatbáziselemre, az adott művelet végrehajtásához.

Legális ütemezés, egyszerű tranzakció modell

Az ütemező fogadja a tranzakcióktól a kéréseket, és vagy megengedi a műveleteket az adatbáziselemen, vagy addig *késlelteti,* amikor már biztonságosan végre tudja hajtani őket, vagy megszakítja (ABORT).

A tranzakcióknak zárakat kell kérniük, és feloldaniuk az adatbáziselemekre. A zárak használatának helyesnek kell lennie, mind a tranzakciók szerkezetére, mind pedig az ütemezők szerkezetére alkalmazva.

Tranzakciók konzisztenciája: A műveletek és a zárak az alábbi elvárások szerint kapcsolódnak egymáshoz:

1. A tranzakció csak akkor olvashat, vagy írhat egy elemet, ha már korábban zárolta az elemet, és még nem oldotta fel a zárat.
2. Ha egy tranzakció zárolt egy elemet, akkor később azt fel kell szabadítania.

Az ütemezések jogszerűsége: Nem zárolhatja két tranzakció ugyanazt az elemet, csak úgy, ha az egyik előbb már feloldotta a zárat.

Az ütemező *zártáblát* használ a zárolások nyilvántartására. A zártábla (X, T) párokból álló zárolások (elem, tranzakció) táblázata, ahol a T tranzakció zárolja az X adatbáziselemet. Új jelöléseket vetetünk be a *zárolás* és a *feloldás* műveletekre:

$l_i(X)$: T_i tranzakció az X adatbáziselemre zárolást kér (lock).

$u_i(X)$: T_i tranzakció az X adatbáziselem zárolását feloldja (unlock).

Adottak a T_1, T_2 tranzakciók. A T_1 hozzáad az **A** és **B** adatbáziselemekhez 100-at, a T_2 pedig *megduplázza* az értéküket.

T_1: $l_1(A)$; $r_1(A)$; A:=A+100; $w_1(A)$; $u_1(A)$; $l_1(B)$; $r_1(B)$; B:=B+100; $w_1(B)$; $u_1(B)$;

T_2: $l_2(A)$; $r_2(A)$; A:=A*2; $w_2(A)$; $u_2(A)$; $l_2(B)$; $r_2(B)$; B:=B*2; $w_2(B)$; $u_2(B)$;

Mindkét tranzakció konzisztens, mert felszabadítják az **A**-ra és **B**-re kiadott zárakat, csak akkor végeznek műveleteket az **A**-n és a **B**-n, amikor már zárolták az elemet, és még nem oldották fel a zár alól.

Az alábbi táblázatban a két tranzakció egy jogszerű ütemezése található:

T_1	T_2	**A**	**B**
		x	y
$l_1(A)$; $r_1(A)$;			
A:=A+100;			
$w_1(A)$; $u_1(A)$;		x+100	
	$l_2(A)$; $r_2(A)$;		
	A:=A*2;		
	$w_2(A)$; $u_2(A)$;	2*(x+100)	
	B:=B*2;		
	$w_2(B)$; $u_2(B)$;		2*y
$l_1(B)$; $r_1(B)$;			
B:=B+100;			
$w_1(B)$; $u_1(B)$;			2*y+100

7.3. táblázat.

Helymegtakarítás miatt több műveletet írtunk egy sorban. Az ütemezés jogszerű, mégsem sorba rendezhető.

A kétfázisú zárolás (2PL) protokoll

Minden tranzakcióban minden zárolási művelet megelőzi az összes zárfeloldási műveletet.

Az első fázisban csak zárolásokat adunk ki (lock), a második fázisban pedig csak megszüntetünk zárolásokat (unlock).

Azokat a tranzakciókat, amelyek eleget tesznek a 2PL feltételnek, kétfázisú zárolású tranzakcióknak vagy *2PL tranzakcióknak* nevezzük.

Holtpont

Az ütemező úgy kényszeríti ki az ütemezést, hogy *várakoztatja* a tranzakciókat. Problémák lehetnek, ha a tranzakciók egymásra várnak.

Holtpont (deadlock, patt) keletkezik, amikor a tranzakciók közül egyik sem tud tovább futni, mert vár egy másikra.

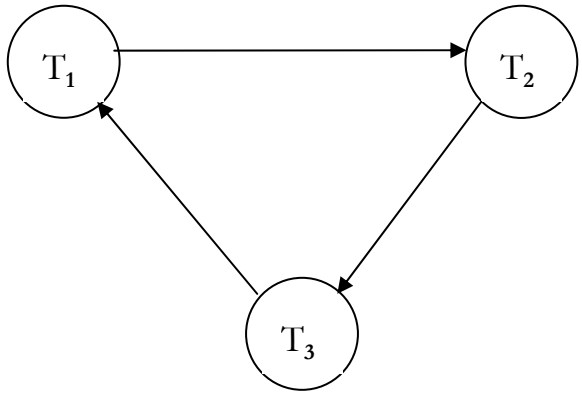
Például: $l_1(A)$; l_2 (B); $l_3(C)$; $l_1(B)$; $l_2(C)$; $l_3(A)$

sorrendben érkező zárkérések esetén egyik tranzakció se tud tovább futni. Az ilyen helyzeteket el kell kerülni, illetve ha már kialakultak, akkor fel kell ismerni és meg kell szüntetni.

Várakozási gráf

A holtpont felismerésében segít a zárkérések sorozatához tartozó *várakozási gráf*: Csúcsai a tranzakciók, és akkor van él T_i-ből T_j-be, ha T_i vár egy olyan zár elengedésére, amit T_j tart éppen.

Például az előbbi, holtponthoz vezető zárkérés-sorozat várakozási gráfja a hat zárkérés után:



7.1. ábra. Várakozási gráf

A várakozási gráf változik az ütemezés során, ahogy újabb zárkérések érkeznek, vagy zár elengedések történnek.

Holtpont felismerése

Az ütemezés során egy adott pillanatban pontosan akkor nincs holtpont, ha az adott pillanathoz tartozó várakozási gráf DAG (Directed Acyclic Graph: nincs benne irányított kör).

Holtpontok elkerülése:

1. Rajzoljuk folyamatosan a *várakozási gráfot*. Ha holtpont alakul ki, akkor megszakítjuk (ABORT) az irányított kör egyik tranzakcióját.
2. *Időkorlát* alkalmazása: ha egy tranzakció kezdete óta túl sok idő telt el, akkor megszakítjuk.

Éhezés

Többen várnak ugyanarra a zárra, de amikor felszabadul, mindig elviszi valaki a tranzakció orra elől.

Megoldást jelenthet éhezés ellen az adategységenként *várakozási lista* a zárkiosztásra.

7.1.3. Sorosítási gráf az egyszerű tranzakciómodellben

A *sorosítási gráf* csúcsai tranzakciók, és akkor van él T_i-ből T_j-be, ha az ütemezésben van olyan $u_i(A) \ldots l_j(A)$ rész, ahol $u_i(A)$ (T_i elengedi A zárját) és $l_j(A)$ (T_j megkapja A zárját) között A-ra senki se kap zárat. Ekkor minden olyan *soros ütemezésben*, ami ekvivalens lehet a miénkkel, biztos, hogy T_j-nek T_i után kell jönnie. Ez azért van így, mert feltettük, hogy T_i is és T_j is bármit csinálhat A-val, amíg nála van a zár, és ha pl. T_i írja, T_j meg olvassa A-t, akkor már csak a $T_i, \ldots, T_j$ sorrend lesz a jó.

Egy csak zárkéréseket és zárelengedéseket tartalmazó egyszerű tranzakció modellbeli ütemezés pontosan akkor sorosítható, ha az előbbi módszerrel felrajzolt *sorosítási gráf* DAG.

Az ütemező lehetőségei a sorosíthatóság kikényszerítésére:

1. **Figyeli a sorosítási gráfot** (amit a zárkérések alapján készít) és ha kör keletkezne, akkor az egyik körbeli tranzakciót ABORT-álja.
2. **Protokollt ír elő a tranzakciók számára**, amit minden egyes tranzakciónak be kell tartania: 2PL.

Ha az egyszerű tranzakció modellbeli legális ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó sorosítási gráf DAG, azaz az ütemezés sorba rendezhető.

7.1.4. Bonyolultabb zármodellek

Többfajta zár van, aszerint, hogy a tranzakciók mit akarnak tenni az adattal (pl.: olvasni vagy írni).

Kompatibilitási mátrix

Egy mátrix segítségével adjuk meg, hogy különböző tranzakcióknak milyen zárai lehetnek egyszerre egy adategységen.

Kompatibilitási mátrix: A sorok és az oszlopok is a lehetséges záraknak felelnek meg. A Z_i sor Z_j oszlopában pontosan akkor van I (Igen), ha egy tranzakció megkaphatja egy adategységre a Z_i zárat akkor, ha egy másik tranzakció Z_i zárat tart fenn ezen az adategységen. Ha nem kaphatja meg, akkor N (Nem) áll a Z_i sor Z_j oszlopában.

Akkor lehet két különböző tranzakciónak Z_i és Z_j zárja ugyanazon az adategységen, ha mindegy, hogy a két zárnak megfelelő műveletek milyen sorrendben hajtódnak végre.

A kompatibilitási mátrix használata

1. Ez alapján dönti el az ütemező, hogy egy ütemezés/zárkérés legális-e, illetve ez alapján várakoztatja a tranzakciókat. Minél több az I a mátrixban, annál kevesebb lesz a várakoztatás.
2. A mátrix alapján keletkező várakozásokhoz elkészített *várakozási gráf* segítségével az ütemező kezeli a *holtpontot* (ami tetszőleges zármodell esetén ugyanazt jelenti, és a gráfot is ugyanúgy kell felépíteni).
3. A mátrix alapján készíti el az ütemező a *sorosítási gráfot* egy zárkérés-sorozathoz: a sorosítási gráf csúcsai a tranzakciók, és akkor van él T_i-ből T_j-be, ha van olyan A adategység, amelyre az ütemezés során Z_k zárat kért és kapott T_i, ezt elengedte, majd ezután A-ra legközelebb T_j kért és kapott Z_l zárat és a mátrixban a Z_k sor Z_l oszlopában N áll. Olyankor lesz él, ha a két zár nem kompatibilis egymással, nem mindegy a két művelet sorrendje.

7.1.5. Összefüggések az adategységek között

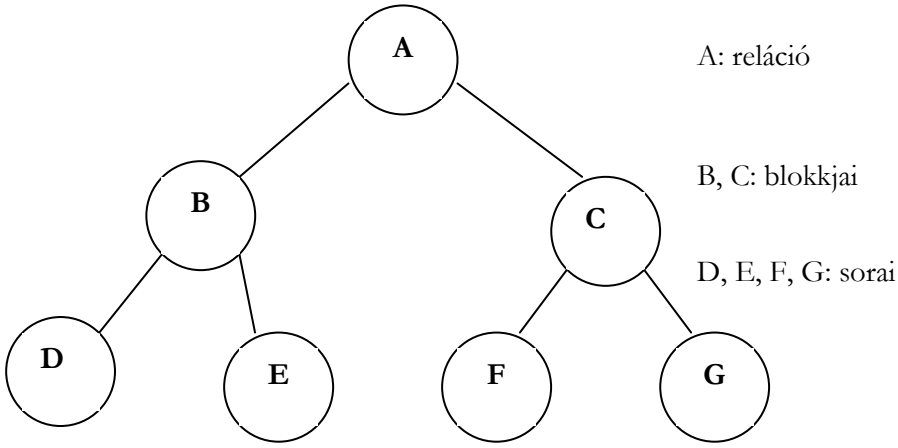
Eddig feltételeztük, hogy az adategységek között nincs összefüggés, kapcsolat. Nem alkalmazható ez a feltevés:

1. Ha az adategységek *egymásba ágyazottak* (pl. reláció, blokk, rekord).
2. Ha tudjuk, hogy egymáshoz képest hogyan helyezkednek el az adategységek a tárolási struktúrában (például a *B-fában* tárolt adatok).

7.1.6. Adatbáziselemekből álló hierarchiák

A valóságban zárolhatunk teljes relációkat, de zárolhatjuk külön-külön ezek egyes blokkjait, sőt sorait is. Minél nagyobb egységre rakunk zárat, annál könnyebb lesz a záradminisztráció, de annál több lesz a zárfeloldásra várás is és ezzel együtt a holtpont. Alkalmazástól függ, hogy mi éri meg jobban, de egy valami mindig közös: Elvárjuk azt, hogy ha az A adategység egy reláció, B pedig ennek egy blokkja, akkor az A-ra rakott zár zárolja B-t is, azaz pl. az egyszerű tranzakció modellben ne lehessen $l_j(B)$-t kapni, ha $l_i(A)$ után még nem volt $u_i(A)$. Ezt az eddigi technika még nem biztosítja, eddig ilyen összefüggéseket nem is vettünk figyelembe.

Egy ilyen lehetséges hierarchikus helyzet:



7.2. ábra. Hierarchikus adatszerkezet

Figyelmeztető zármodell

Olyan sorosítható ütemezést szeretnénk, ami az adategységek közötti hierarchiát is figyelembe veszi. Ez a hierarchia egy fával adott. Egyszerű változat esetén háromféle zárművelet lesz:

1. $LOCK_i(A)$: T_i zárolja A-t (explicit lock) és minden leszármazottját (implicit lock), kizárólagosan, azaz ezek után más tranzakció se A-ra, se ennek leszármazottjára nem kaphat zárat.
2. $WARN_i(A)$: T_i figyelmeztetést rak A-ra (gyerekeire nem), ez annak jelzésére szolgál, hogy T_i majd zárat akar kapni A valamely leszármazottjára.
3. $UNLOCK_i(A)$: felszabadítja az A-ra rakott LOCK-ot és WARN-t, az implicit zár is lekerül A leszármazottjairól.

A zárak használata:

1. Az i-edik tranzakció, T_i, csak akkor olvashatja, vagy írhatja az A adategységet, ha előtte zárat kért és kapott rá ($LOCK_i(A)$ vagy $LOCK_i$ A valamelyik ősén), és ezt a zárat még azóta nem engedte fel.
2. $LOCK_i(A)$ és $WARN_i(A)$ után mindig van $UNLOCK_i(A)$.

3. Ha $LOCK_i$ van A-n, akkor se $WARN_j$ se $LOCK_j$ nem kerülhet már rá (ha $j \neq i$), de két különböző tranzakciónak lehet WARN-ja ugyanott. A kompatibilitási mátrix:

	LOCK	WARN
LOCK	N	N
WARN	N	I

A figyelmeztető protokoll

A T_i tranzakció követi a *figyelmeztető protokollt*, ha zárkérései a fentieken kívül még a következőket is tudják:

(b) T_i első zárkérése $WARN_i$ vagy $LOCK_i$ a gyökérre.
(c) Ezután $LOCK_i$ vagy $WARN_i$ csak akkor kérhető egy adategységre, ha $WARN_i$ már van az apján.
(d) $UNLOCK_i$ csak akkor kérhető egy adategységre, ha már nincs sem explicit $LOCK_i$, sem $WARN_i$ az adategység leszármazottjain.
(e) Kétfázisú zárkérés van: $UNLOCK_i$ után nincs se $LOCK_i$, se $WARN_i$. Az (a) és (b) pontok miatt a zárkérések felülről lefelé kúsznak a fában, a zárelengedések pedig a (c) miatt alulról felfele mennek az egyes tranzakciók esetén.

Ha a figyelmeztető zármodellben, egy legális ütemezésben minden tranzakció követi a figyelmeztető protokollt, akkor az ütemezés sorosítható, és nem lesz egyszerre két különböző tranzakciónak zárja ugyanazon az adategységen.

B-fában tárolt adategységek

A zárolható adategységek egy fa csúcsaiban helyezkednek el. A fa azt mutatja, hogy hogyan lehet elérni az adatokat. Például a B-fa esetén, a levelekhez csak úgy juthatunk el, ha a gyökértől indulva végigjárunk egy lefele vezető utat.

Faprotokoll

A T_i tranzakció követi a faprotokollt, ha

1. Az első zárat bárhova elhelyezheti.
2. Ezután csak akkor kaphat zárat A-n, ha zárja van A apján.
3. Zárat bármikor fel lehet oldani (nem 2PL).

4. Nem lehet újrazárolni, azaz ha T_i elengedte egy A adategység zárját, akkor később nem kérhet rá újra (még akkor sem, ha A apján még megvan a zárja).

Ha minden tranzakció követi a faprotokollt egy legális ütemezésben, akkor az ütemezés sorosítható lesz, noha nem feltétlenül lesz 2PL.

7.1.7. Sorosíthatóság időbélyegekkel

Minden tranzakciónak van egy időbélyege: $t(T_i)$ a T_i tranzakcióé. Az időbélyegek egyediek, növekvő sorrendben adja ki őket az ütemező, ahogy indulnak a tranzakciók.

Az ütemező az időbélyegek növekvő sorrendjéhez tartozó soros ütemezéssel azonos hatású ütemezést enged csak lefutni, minden olyan kérést letilt (és a megfelelő tranzakciót ABORT-álja), ami ez ellen tesz.

Például, ha $t(T_1) = 120$, $t(T_2) = 90$ és $t(T_3) = 130$, akkor a cél a $T_2T_1T_3$ soros sorrenddel azonos hatású ütemezés.

7.2. Ellenőrző kérdések

1. Mit jelent a konkurenciavezérlés, miért van rá szükség?
2. Ismertesse az ütemező feladatát, szerepét!
3. Adja meg a soros, sorba rendezhető ütemezések fogalmát!
4. Ismertesse az egyszerű tranzakció modellt!
5. Ismertesse a következő fogalmakat: zártábla, 2PL protokoll!
6. Ismertesse a következő fogalmakat: holtpont, várakozási gráf!
7. Ismertesse a következő fogalmakat: éhezés, sorosítási gráf!
8. Adjon elégséges feltételt egy ütemezés sorosíthatóságára egyszerű tranzakció modellt használva!
9. Ismertesse a figyelmeztető zármodell használatát!
10. Ismertesse a figyelmeztető protokoll használatát!
11. Ismertesse a fa protokoll használatát!
12. Ismertesse a következő fogalmakat: piszkos adat, lavina!
13. Ismertesse a szigorú 2PL protokollt!

Irodalom

[1] J. D. Ullman – J. Widom: *Adatbázisrendszerek,* Panem, 1998.
[2] H. Garcia-Molina – J. D. Ullman – J. Widom: *Adatbázisrendszerek megvalósítása,* Panem, 2001.
[3] Halassy Béla: *Adatmodellezés,* Nemzeti Tankönyvkiadó, 2002.
[4] Czenky Márta: *Adatmodellezés,* ComputerBooks, Budapest, 2002.
[5] Stolnicki Gyula: *SQL programozóknak,* ComputerBooks, 2003.
[6] Gajdos Sándor: *Adatbázisok,* Műegyetemi Kiadó, 2004.
[7] Katona Gyula: *Adatbázisok előadás diák,* BME, 2005.
[8] *Microsoft SQL Server 2005 Books Online*

Név- és tárgymutató