Adatbázisok I

Előadás és gyakorlat jegyzet

Készült Szalai-Gindl János Márk előadásai
és Vörös Péter gyakorlatai alapján

Sárközi Gergő, 2021-22-2. félév
Nincsen lektorálva!

Tartalomjegyzék

1. Bevezetés

- Adatbázis: Database Management System által kezelt adatok együttese
- DBMS feladatai:
  - Adatbázis létrehozása, adatbázis séma definiálása
  - Adatok lekérdezése, módosítása (hatékonyan)
  - Nagyméretű adatok hosszú időn történő tárolása
  - Autentikáció és meghibásodás ellen védelem
  - Egyszerre több felhasználó egyidejű hozzáférésének biztosítása
- Több adatbázist egy adattárház (middleware) fog össze

2. Relációs adatmodell

- Adatmodell: információ/adatok leírására szolgáló jelölés
  - Adatok struktúrája
  - Adatokon végezhető műveletek
  - Adatokra vonatkozó megszorítások (pl. unique)
- Legfontosabb adatmodellek: relációs, félig strukturált (XML)
- Adatok 2D-s táblákban (relációkban) vannak
  - Egy reláció az sorok halmaza, ennek a halmaznak neve: előfordulás
  - Halmaz $\implies$ nem számít a sorrend, egy sor max egyszer szerepelhet
- Reláció fejrészében attribútumok vannak
  - Relációséma: attribútumok halmaza; adatelemek adatainak formája
  - attribútumok sorrendje nem fix, nevük alapján hivatkozhatók
  - attribútumok meghatározzák az értékkészletet
- Adatbázis: relációk halmaza
- Egy sor elemei: mezők (komponensek)
  - Csak atomi értékeket vehetnek fel: szám, szöveg, referencia, stb.

3. Relációs algebra

- Algebra: műveletek és atomi operandusok
  - Eredmény: reláció
  - Operandusok: konstansok, változók (relációkból)
- Monoton operátor: egy reláción értelmezett operátor akkor monoton, ha bármely argumentumrelációhoz egy újabb sort hozzávéve az eredmény tartalmazza az összes eddigi sort, és esetleg újabbakat is

3.1. Megszorítások

- Megjegyzés: algebrai műveletek lejjebb vannak, de kellenek ide
- Relációra vonatkozó megszorítás: $R = \emptyset$ vagy $R \subseteq S$
  - Kifejezőkészségük azonos, pl. R sorai részhalmaza S sorainak
- Hivatkozási épség megszorítás: ha itt megjelenik, ott is meg kell
  - Kifejezés: $\Pi_A(R) \subseteq \Pi_B(S)$
- Kulcs: attribútumok halmaza egy kulcs, ha a reláció bármely (mostani és jövőbeli) előfordulásában nincs két olyan sor, amelyek a kulcs összes attribútumának értékein megegyeznek.
  - Példa: (a, b) kulcs: $\sigma_{R.a=S.a \wedge R.b=S.b \wedge R.c \neq S.c}(R \times S) = \emptyset$
  - Aláhúzással jelöljük (általában?)

3.2. Algebrai műveletek

- Projekció, vetítés: attribútumok szűkítése, jelölés: $\Pi_{A,B}(R)$
- Szelekció, kiválasztás: feltétel alapján sorok szűrése
  - Jelölés: $\sigma_{A=a \wedge C=d}(R)$
  - Minden tag 1 vagy 2 attribútumból áll (vagy 2, vagy 1 és 1 konstans)
  - A tagokban ezek a műveletek lehetnek: $=, <, >, \neq, \leq, \geq$
  - A tagokat ezek kapcsolhatók: $\neg, \wedge, \vee$
- Unió, metszet, különbség
  - Feltétel: operandusok attribútumai megegyeznek
- Descartes-szorzat, jelölés: $R \times S$
  - Azonos nevű attribútumok a reláció nevét kapják prefixnek
  - Asszociatív (szabadon zárójelezhető), kommutatív (felcserélhető)
- Átnevezés: $\rho_{S(C,D,E)}(R)$ vagy $\rho_S(R)$
- Théta-összekapcsolás: $R \bowtie_F S \Leftrightarrow \sigma_F(R \times S)$ (ahol F egy feltétel)
  - Példa: $R \bowtie_{R.A=S.A} S$
  - Kommutatív
  - Asszociatív, ha a feltétel nem használ olyan attribútumot, ami a "távolabbi" (csak megfelelő zárjelezés esetén elérhető) relációból származik
    - $E_1 \bowtie_{F_1} (E_2 \bowtie_{F_2} E_3) \equiv (E_1 \bowtie_{F_1} E_2) \bowtie_{F_2} E_3$
      ha $attr(F_1) \subseteq attr(E_1) \cup attr(E_2)$
      és $attr(F_2) \subseteq attr(E_2) \cup attr(E_3)$
- Egyen-összekapcsolás, equi join:
  - Théta-összekapcsolás, ahol egyenlőség a feltétel
- Természetes összekapcsolás: Théta, F= minden azonos attribútum egyezik
  - Jelölés: $R \bowtie S$
  - Alternatíva: Théta-összekapcsolás, majd közös attribútumokból 1
  - Kommutatív, asszociatív (mindig)

4. Relációs algebrai lekérdezések optimalizációja

- Ekvivalens lekérdezések: tetszőleges előfordulás esetén is azonos eredmény
- Lépések (mindezt egy fán használjuk):
  - Műveletek szétbontása (pl. konjunkció, összekapcsolás)
    - Természetes összekapcsolásból Descartes-szorzat és kiválasztás
  - Műveletek a levélhez közel helyezése (minél előbbre)
  - Projekciók behelyezése (nem szükséges oszlopok levágása)
    - Akkor nagy a nyereség, ha az oszlopok vannak külön-külön fájlban tárolva (nem pedig a sorok)
    - Nem kötelező kiválasztás elé: elég kiválasztás után, de összekapcsolás előtt
  - Természetes és théta összekapcsolásokra cserélni, amit lehet
- Diszjunkció bonyolultabb, mint a konjunkció
- Van arra példa, hogy egy kiválasztást először felfelé kell csúsztatni, hogy aztán le lehessen tolni.

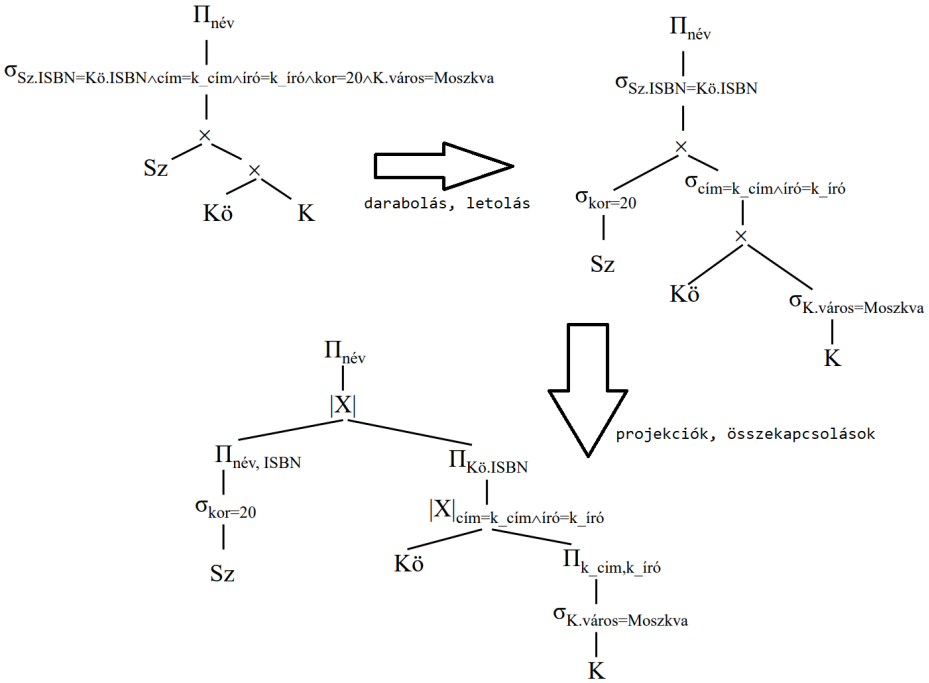
4.1. Műveletek optimalizációja

- Projekció sorozat: $X \subseteq Y \implies \Pi_X(\Pi_Y(E)) \equiv \Pi_X(E)$
- Kiválasztás és feltételek...
  - konjunkciója: $\sigma_{F1}(\sigma_{F2}(E)) \equiv \sigma_{F1 \wedge F2}(E)$
  - diszjunkciója: $\sigma_{F1}(E) \cup \sigma_{F2}(E) \equiv \sigma_{F1 \vee F2}(E)$
- Kiválasztás elé projekció beillesztése:
  $Y = attr(F) \cup X \implies \Pi_X(\sigma_F(E)) \equiv \Pi_X(\sigma_F(\Pi_Y(E)))$
- Kiválasztás és Descartes-szorzat/összekapcsolás:
  - Legyen Δ vagy $\times$ vagy $\bowtie$
  - Ha $F = F_1 \wedge F_2$ és $attr(F_i) \subseteq attr(E_i)$
    Akkor $\sigma_F(E_1 \Delta E_2) \equiv \sigma_{F_1}(E_1) \Delta \sigma_{F_2}(E_2)$
  - Ha $F = F_1 \wedge F_2$ és $attr(F_1) \subseteq attr(E_1)$ de $attr(F_2) \not\subseteq attr(E_i)$
    Akkor $\sigma_F(E_1 \Delta E_2) \equiv \sigma_{F_2}(\sigma_{F_1}(E_1) \Delta E_2)$

- Projekció és Descartes-szorzat/összekapcsolás:
  - Legyen Δ vagy $\times$ vagy $\bowtie$
  - Ha $X = Y \cup Z$ és $Y \subseteq attr(E_1)$ és $Z \subseteq attr(E_2)$
  - Akkor $\Pi_X(E_1 \Delta E_2) \equiv \Pi_Y(E_1) \Delta \Pi_Z(E_2)$
- Kiválasztás és halmazműveletek: disztributív
  Legyen Δ vagy $\cap$ vagy $\cup$ vagy $\setminus$: $\sigma_F(E_1 \Delta E_2) \equiv \sigma_F(E_1) \Delta \sigma_F(E_2)$
- Projekció és unió disztributivitás: $\Pi_X(E_1 \cup E_2) \equiv \Pi_X(E_1) \cup \Pi_X(E_2)$

4.2. Optimalizációra példa

- Feladat: Kik azok a 20 évesek, akik moszkvai kiadású könyvet kölcsönöztek?
  - Személy(név, kor, város, ISBN)
  - Könyv(cím, író, ISBN, ár)
  - Kiad(k-cím, k-író, város, ország)



5. SQL bevezetés

- Magas szintű nyelv: "hogyan" helyett "mit"-et írjuk le
- Adatbázis kezelő optimalizál: kitalálja a leggyorsabb végrehajtási módot
- Itt már multihalmaz van: lassú mindig kitörölni az ismétlődést

5.1. Lekérdezés jelentése

- Kezdjük a FROM záradékban megadott relációval
- Alkalmazzuk a WHERE záradékban megadott kiválasztási feltételt
- Levetítjük az eredményt a SELECT-ben megadott oszlopokra

5.2. SQL érdekességek

- SELECT *
- SELECT név AS sör; SELECT ár*300 AS árForint
- SELECT 'konstans' as valami (pl. adattárházban integráció miatt)
- WHERE alapműveletek: AND, OR, NOT, =, <>, <, >, <=, >=
- WHERE attribútum (NOT) LIKE minta
  - Mintában %: 0 vagy több akármilyen karakter
  - Mintában _: pontosan egy akármilyen karakter
  - Mintában bármilyen más karakter: önmagát jelenti
- WHERE attr BETWEEN x AND y (ahol x kötelezően kisebb, mint y)
- Több táblából lekérdezés: FROM után több tábla
  - Ezután duplikált attribútum nevek neve: TÁBLA.OSZLOP
- Azonos táblából többször: FROM tábla t1, tábla t2
- Alapból multihalmaz, ha ne: SELECT DISTINCT

5.3. NULL értékek

- Mezők NULL értéket is tartalmazhatnak
- Lehetséges jelentés: hiányzik, értelmetlen
- SQL 3 értékű logikát használ: TRUE, FALSE, UNKNOWN
  - Legyen TRUE=1, FALSE=0, UNKNOWN=0.5
  - Ekkor AND=MIN, OR=MAX, NOT(x)=1-x
- NULL-al hasonlítás (akár NULL-t): eredmény UNKNOWN
  - Helyesen: IS (NOT) NULL
- Egy sor WHERE-ben UNKNOWN, akkor nem kerül be az eredménybe

5.4. Alkérdések

- FROM és WHERE záradékban SELECT-FROM-WHERE-t is használhatunk
  - FROM esetén az ideiglenes táblának nevet kell adni általában
- Egy sort visszaadó alkérdések
  - Konstans értéknek lehet kezelni
  - Általában csak egy oszlopa van
  - Futásidejű hiba keletkezik, ha több sor van
- Tábla rendes kérdésben és alkérdésben is be van húzva: oszlopnév alapból a legközelebbi FROM-ra vonatkozik (azaz az alkérdésre)
- Alkérdéses műveltek:
  - Tartalmazás: sor (NOT) IN alkérdés
  - Alkérdés (nem) üres: (NOT) EXISTS alkérdés
    - Korrelált alkérdés: minden sorra külön lefut, mert függ tőlük
  - Halmazműveletek: INTERSECT, UNION, MINUS (=EXCEPT)
    - Nem multihalmaz, hanem halmaz szemtikát halmaz
    - Hogy mégis multihalmaz legyen: <UNION/...> ALL
  - ANY, ALL: pl. x > ANY(alkérdés); x <> ALL(alkérdés)

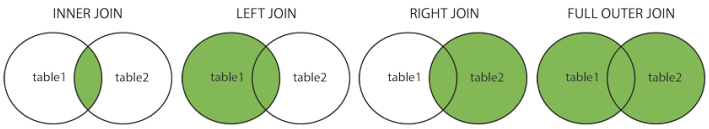
5.5. Összekapcsolás (join)

- Emlékeztető: $NULL \neq NULL$
- Természetes összekapcsolás: a NATURAL JOIN b
  - SQL optimalizál: nincs SELECT-elve az oszlop $\implies$ nem működik
  - Önmagával: $NULL \neq NULL$, tehát $NULL$-os sorok eltűnnek
- Descartes-szorzat: a CROSS JOIN b (ugyan az, mint a vessző: a, b)
- Théta-összekapcsolás: a JOIN b ON aa = bb
  - Ugyan az, mint az INNER JOIN
  - Ugyan az, mint: a, b WHERE aa = bb

6. SQL haladó

6.1. Összekapcsolás (join)

- Félig összekapcsolás (SEMI JOIN, '$\ltimes$'): olyan természetes összekapcsolás, ami csak az első relációból veszi az attribútumokat
  - SELECT R.* FROM R WHERE EXISTS(... R.A = S.A)
- Anti összekapcsolás (ANTI JOIN, '$\triangleright$'): $R \triangleright S \equiv R \setminus R \ltimes S$
  - SELECT R.* FROM R WHERE NOT EXISTS(... R.A = S.A)
- Külső összekapcsolás:
  - SQL-ben: R <típus> (OUTER) JOIN S (ON ...)
    - Típus: LEFT (⟕_C), RIGHT (⟖_C), FULL (⟗_C or ${}_L\text{⟗}_{R\,C}$)
    - Vagy NATURAL, vagy ON szerepel (vagy semmi): kizáróak
  - Lógó sor: eredmény része, de nincs hozzárendelt sor másik relációban
    - Külső összekapcsolás lényege: ilyen sorokat megőrzi
    - Ilyenkor a hiányzó oszlopok NULL értékű lesznek



6.2. Összesítések (aggregációk)

- SELECT záradékban alkalmazhatjuk egy-egy oszlopra
- SUM, AVG, COUNT, MIN, MAX
- Összesítő függvényen belül DISTINCT: COUNT(DISTINCT ár)
- NULL érték nem számít: mintha nem is lenne ott a sor
- Üres halmaz eredménye: NULL, kivéve COUNT: ott 0
- Ha GROUP BY nélkül van: mintha az egész reláció egy csoport lenne

6.3. Csoportosítás (grouping)

- WHERE után záradék: GROUP BY oszlop
- SELECT sör, AVG(ár) FROM x GROUP BY sör;
- SELECT-ben csak bizonyos dolgok lehetnek:
  - Összesítések, amelyek így csoportonként kiértékelődnek
  - Attribútumok, amik alapján GROUP BY-t csináltunk
- Csoport szűrés: HAVING ("megfelelője": WHERE)
  - Értelemszerűen meg van szorítva, mint a SELECT

6.4. Sorba rendezés

- ORDER BY = ORDER BY ASC; másik: ORDER BY DESC

6.5. Adatbázis módosítások

- Nem query: nem ad vissza eredményt
- Insert, Update, Delete
- Data Manipulation Language

6.5.1. Beszúrás

- INSERT INTO <tábla> VALUES (...)
  - Itt minden oszlopnak értéket kell adni
- Tábla neve után megadhatók az attribútumok:
  - INSERT INTO sör(név, ár) VALUES ('Bud', 100)
  - Így a sorrendet "lokálisan" definiáljuk, nem kell emlékezni
  - A hiányzó attribútumok default (NULL/beállított) értéket kapnak
- Több sor beszúrása: INSERT INTO <reláció> (alkérdés);

6.5.2. Törlés

- DELETE FROM <reláció> (WHERE ...)
- Nincs WHERE: összes sor törlése
  - Alternatíva: TRUNCATE TABLE <név> (ez séma módosítás féleség)
- Két lépésben hajtódik végre:
  - Először törlendő sorok kijelölése, majd kijelölt sorok kitörlése
  - Emiatt determinisztikus: nem számít melyik tűnik el előbb

6.5.3. Módosítás

- UPDATE <reláció> SET oszlop1=v1, oszlop2=v2, ... [WHERE ...]

6.6. Adatbázis sémák

- Data Definition Language
- Létrehozás: CREATE TABLE <név> (<elemek, megszorítás listája>);
  - Elem: <név> <típus> [sor megszorítás] [DEFAULT <érték>]
  - Megszorítás: CONSTRAINT <név_FK> <FK szintaxis lejebb>
- Törlés: DROP TABLE <név>;

6.6.1. Típusok

- Szám: INT=INTEGER, REAL=FLOAT
  - Tizedesvessző: . (pont)
- Szöveg: CHAR(n) (mindig n), VARCHAR(n) (legfeljebb n)
  - VARCHAR2(n): Oracle valami, $\sim$ VARCHAR
  - Aposztrófok közé kell rakni
  - Két aposztróf = egy igazi aposztróf (escape)
- DATE, megadás: DATE 'yyyy-mm-dd'
  - Gyakon megadás: TO_DATE('2000.01.01', 'YYYY.MM.DD')
- TIME, megadás: TIME 'hh:mm:ss' vagy TIME 'hh:mm:ss.tizedmp'

6.6.2. Tábla módosítás

- ALTER TABLE <név> ....
  - DROP COLUMN/CONSTRAINT <név>
  - ADD <név> <típus> [megszorítások, stb]
  - MODIFY <meglévő név> <új típus> [megszorítások, stb]
- Oszlopot nem lehet töröli, ha van hozzá CONSTRAINT

6.7. Megszorítások

- Adatelemek közötti kapcsolat
- Adatbázis rendszer tartja fel őket
- Relációs algebrában: egy lekérdezés értéke legyen pl. üres vagy részhalmaz
- Gyakori megszorítás: NOT NULL

6.7.1. Kulcsok

- Attribútum vagy attribútum lista lehet kulcs
  - Egy esetén: create table-ben, attribútum sora végén is lehet
  - Több esetén: csak az attribútum lista után lehet
- Relációnak nem lehet két sora, amiben megegyeznek
- PRIMARY KEY vagy UNIQUE
  - PRIMARY KEY-ből csak 1db lehet, nem lehet NULL
  - UNIQUE-ból több is lehet, lehet NULL
- Oracle SQL: nincs AUTO INCREMENT, helyette hack:
  INSERT INTO x(y) VALUES ((SELECT MAX(y)+1 FROM x))

6.7.2. Idegen kulcsok, hivatkozási épség megszorítás

- Relációs algebrában: $\Pi_{beer}(Serve) \subseteq \Pi_{name}(Beer)$
- Megadható attribútum után vagy séma végén
  - név típus REFERENCES reláció(attribútum)
  - FOREIGN KEY (attribútumok...) REFERENCES reláció (attribútumok...)
- Csak kulcs (PRIMARY KEY / UNIQUE) attribútumokon működik
- Sértheti: beszúrás vagy a másik oldalon törlés/módosítás
  - Rossz beszúrást csak egyféleképpen kezel az SQL: nem engedi
  - Rossz törlés/módosítás kezelése:
    - Default: szerintem ezt jelenti: nem engedi a módosítást (RESTRICT)
    - Továbbgyűrűzés: másik táblában is töröljük/módosítjuk sor(ok)at
    - Set NULL: másik táblában NULL értéket kap
    - ... KEY ... ON [UPDATE/DELETE] [SET NULL / CASCADE]

6.7.3. Attribútum alapú (érték-alapú) megszorítás

- Egy adott oszlopra vonatkozik a megszorítás, annak értékeit ellenőrzi
- Csak beszúrásnál, módosításnál van végrehajtva
  - Azaz törlésnél nem
  - Azaz foreign-key féle koncepció nincsen
- Megadás: attribútum után CHECK(feltétel)
  - Alapból csak a saját attribútumra hivatkozhatunk
  - Alkérdés segítségével bármi hivatkozható

6.7.4. Sor-alapú megszorítás

- Sor mezei közötti kapcsolatot definiál
- Csak beszúrásnál és módosításnál van végrehajtva
- Megadás: reláció séma után CHECK(feltétel)
  - Saját relációban bármelyik attribútumra hivatkozhatunk
  - Alkérdés segítségével bármilyen reláció hivatkozható

6.7.5. Globális megszorítás

- Adatbázissémához tartozik (nem pedig relációhoz)
- Bármira hivatkozhat (tetszőleges tábla, tetszőleges oszlop)
- Minden módosítás előtt (insert, update, delete) ellenőrizve van
  - Adatbázis rendszer optimalizálhat: nem nézi meg mindig
- CREATE ASSERTION név CHECK (feltétel)

6.8. Trigger

- Kód végrehajtódik, amikor egy esemény történik (pl. sor beszúrása)
- Más név: ECA szabály (event-condition-action)
- Példa: beszúrás visszautasítása helyett (foregin key miatt), beszúrhatunk egy sort a másik táblába is automatikusan (pl. NULL értékkel kiegészítve)
- CREATE TRIGGER név ... (nagyon sok lehetőség van)

```
CREATE TRIGGER SörTrig
BEFORE INSERT ON Felszolgál
REFERENCING NEW ROW AS ÚjSor
FOR EACH ROW
WHEN (ÚjSor.sör NOT IN (SELECT név FROM Sörök))
INSERT INTO Sörök(név) VALUES(ÚjSor.sör);
```

6.9. Tranzakciók

- Egység, amit osztatlanul kell végrehajtani
- Lehet lekérdezés és/vagy módosítás
- Minden SQL utasítás az: 1 elemes tranzakció
- Perzisztálni kell: végrehajtás közben rendszerhiba $\implies$ ne legyen baj
- Részei:
  - Konkurenciakezelő: atomicitás megvalósítása, versenyhelyzetek kizárása
  - Naplózás- és helyreállítás-kezelés: tranzakciók tartóssága (perzisztencia)
- Műveletek végrehajtási sorrendje fontos (race condition)
- Valamilyen szinten párhuzamosítva van

6.9.1. ACID tranzakciók

- Atomicity: vagy az összes, vagy semelyik utasítás hajtódik végre
- Consistency: adatbázis megszorítások megőrződnek
- Isolation: úgy tűnik, mintha a folyamatok egymás után futnának le
- Durability: befejeződött tranzakció módosításai nem vesznek el
- Opcionálisan gyengébb feltételek is megadhatóak
  - pl. osztott adatbázisoknál kötelező

6.9.2. SQL utasítások

- COMMIT: tranzakció módosításai megőrződnek
- ROLLBACK: tranzakció abortál, összes utasítás vissza van görgetve
  - 0-val való osztás, stb. automatikus ezt eredményezi

6.9.3. Elkülönítési szintek

- Egy tranzakció milyen állapotot láthat (részeredmények, stb.)
- Négy elkülönítési szint (ACID-ban I betű finomhangolása)
- SET TRANSACTION ISOLATION LEVEL <level>
  - A pillatnyi connection-re állítja be
- SERIALIZABLE (magyarul: sorbarendezhető)
  - Tranzakciók nem befolyásolják egymást, nincs átfedés.
- REPEATABLE READ
  - Csak COMMIT-olt adatot látni, külön-külön olvasásoknál extra sorok megjelenhetnek (ha közben van módosítás). Sorok soha nem módosulhatnak, nem tűnhetnek el.
  - Többletsor neve: fantomsor, fantomadat.
- READ COMMITTED
  - Csak COMMIT-olt adatot látni, de a tranzakción belül minden művelet lehet, hogy más adaton dolgozik: közben lehet módosítás.
  - "Nem ismételhető olvasás"
- READ UNCOMMITTED
  - A nem COMMIT-olt változtatásokat is látja (piszkos adatokat)

6.10. Nézettáblák

- Nézettáblát alaptáblákból és más nézettáblákból definiáljuk
- Virtuális nézettábla: nem tárolódik adat, csak a relációt megadó lekérdezés
- Materializált nézettábla: maga az adat tárolódik
  - Ha sokat lekérdezzük a nézettáblát, akkor gyorsabb lehet
  - Minden módosítás esetén frissül automatikusan, ami idő
    - Beállítható: X időn belül vagy periodikus frissítés
    - Azaz nem kötelező a legfrissebb adatnak benne lennie
  - Előnye új táblához képest:
    - Adatbáziskezelő helyettünk frissíti
    - Lekérdezés optimalizáció akár ki tudja használni ezeket a táblákat
- CREATE (MATERIALIZED) VIEW <név> AS <lekérdezés>
- Állítólag pár egyszerű módosítást nézettáblán keresztül is lehet csinálni
- NEM nézettábla, hanem másolás: CREATE TABLE <név> AS <lekérdezés>

6.11. Indexek

- Adatszerkezet, amivel egy-egy relációt gyorsabban lekérdezhetünk egy/több attribútum alapján
- Megvalósítás lehet hash table, de általában bináris fa
- SQL-ben deklarálás
  - Nincs szabvány/standard megoldás
  - CREATE INDEX <IndexNév> ON Tábla(Oszlop)
  - CREATE INDEX <IndexNév> ON Tábla(Oszlop1, Oszlop2, ...)
- Használatuk: automatikus, csak a megfelelő attribútum alapján szűrjünk
- Hangolás: indexek bevezetése gyorsíthat, de nem feltétlenül jó
  - Indexek mellett: felgyorsítja a lekérdezéseket
  - Indexek ellen: módosítások lassulnak; indexeket is frissíteni kell
  - Menete: query load $\Longrightarrow$ indexek készítése $\Longrightarrow$ hatás vizsgálata

7. SQL jogosultságkezelés

- 9 féle jogosultság van, némelyik oszlop szinten is megadható
- Pár relációra vonatkozó jogosultság:
  - SELECT (lehet, hogy csak adott attribútumra vonatkozik)
  - INSERT (lehet, hogy csak adott attribútumra vonatkozik)
  - DELETE
  - UPDATE (lehet, hogy csak adott attribútumra vonatkozik)
- Jogosultságokat lehet adni még:
  - Nézettábla (virtuális és materializált) lekérdezés/módosítás
  - Relációhoz index, trigger, nézettábla létrehozása
- Jogosultsági azonosító (authorization ID)
  - Általában a bejelentkezési név
  - Egy másik: PUBLIC
- Jogosultságok megadása
  - Általunk létrehozott dolgokra minden jogunk megvan
  - GRANT <jogosultságok...> ON <reláció/valami> TO <JogAzonosítók...>
  - ... WITH GRANT OPTION: jogosultságot kapó tovább is adhatja
- Jogosultságok visszavonása
  - REVOKE <jogosultságok...> ON <reláció/valami> FROM <JogAzonosítók...>
  - Több helyről kapták a jogot $\implies$ megmaradhat utána is
  - REVOKE-hoz meg kell adni egyet:
    - CASCADE: a továbbadott jogosultságokat is megvonjuk
    - RESTRICT: addig nem működik a REVOKE, amíg érvényben van még továbbadott jog
  - REVOKE GRANT OPTION ... is létezik

7.1. Grant diagrammok

- Jogosultság gráf
- Pontok: felhasználó/jogosultság/grant option-e/tuladonos-e
  - Külön pontnak számít: UPDATE ON R és UPDATE ON R(a)
  - AP nevű pont: A azonosítójú P jogát jelenti
  - AP*: P-hez van grant jog
  - AP**: tulajdonosságból származik a jog
- $X \to Y$ él: X pontot használtuk Y megadására
- Alapvető szabály:
  - Legyen C és X felhasználó (akár azonos)
  - Legyen Q egy jogosultság, amit P magába foglal (akár azonos)
  - C-nek addig van joga Q-hoz, amíg vezet XP**-ból az egyikbe út: CQ, CQ*, CQ**
- Gráf frissítéskor: pont nem érhető el egy **-ból se $\implies$ törölni kell

8. Relációs adatbázis tervezés

- Minden adat egyetlen relációban: kényelmes, de sok a felesleges adat
  - Rossz tárolási hatékonyság
  - Ellentmondásossá válhat (nem változtatjuk meg mindenhol az adatot)
- Jobb megoldás: felhasználói eset fogalmait, kapcsolatait modellezni kell
- Jelölés: X,Y,Z attribútum halmaz; A,B,C pedig attribútum

8.1. Kulcsok

- K szuperkulcs R relációra, ha K funkcionálisan meghatározza R attribútumait
- K kulcs R-en, ha K szuperkulcs, de egyetlen valódi részhalmaza sem az
- Azaz például $user\_id, address$ szuperkulcs, de $user\_id$ kulcs
- Kulcs megkapása: paraszti ésszel vagy FF-ek alapján következtetjük ki

8.2. Funkcionális függőségek (FF)

- Redundancia: egy/több oszlopból ki tudunk következtetni egy harmadikat
- Funkcionális függőség ($X \to Y$): ha két sor megegyezik X összes attribútumán, akkor Y-jain is
- Jobboldalak szétvágása (ff): $X \to A_1A_2... \Leftrightarrow X \to A_1 \wedge X \to A_2 \wedge ...$

8.2.1. Armstrong-axiómák 1

- Reflexivitás: ha $Y \subseteq X \subseteq R$ akkor $X \to Y$ (triviális függőség)
- Bővítés: $X \to Y$ akkor tetszőleges $Z \subseteq R$ esetén $XZ \to YZ$
- Tranzitivitás: $X \to Y \wedge Y \to Z \implies X \to Z$
- Felhasználás példa:
  - Feladat: ha $X \to Y$ és $XY \to Z$ akkor $X \to Z$
  - $X \to Y$ bővítés után $X \to XY$
  - $X \to XY$ és $XY \to Z$ reflexivitás miatt $X \to Z$

8.2.2. Lezárás

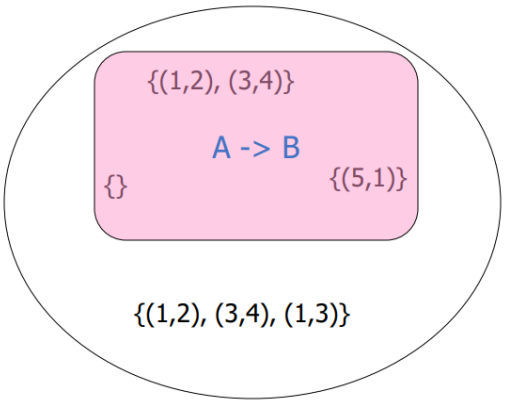
- Y lezártja (Y^+): attribútum halmaz, ami következik Y-ból
  - Azaz: $\forall A \in Y^+ : Y \to A$
- Ha Y szuperkulcs, akkor Y^+ az összes attribútum lesz
- Kiszámítás: iteratívan
  - Első lépés: $Y^+ = Y$
  - Indukció: ha $X \subseteq Y^+$ és $X \to A$, akkor A-t belerakjuk Y^+-ba
  - Iteráció vége: ha utolsó lépésben Y^+ nem változott
- Bizonyítás: $B \in Y^+ \implies Y \to B$
  - Indukcióval (valószínűleg nem kéri számon)
- Bizonyítás: nem lett egy B sem kihagyva
  - Indirekt módon (valószínűleg nem kéri számon)

8.2.3. Összes következmény FF, szétbontás, FF projekció

- Motiváció: egy relációsémát több sémára bontunk
- Ha tudunk attribútumokat és FF-eket, akkor az attribútumok részhalmazán milyen FF-ek lesznek? (normalizálás)
- Példa: eredeti: $AB \to C, C \to D, D \to A$
  ABC részhalmazon teljesül: $AB \to C$ és $C \to A$
- Exponenciális algoritmus:
  - Minden lehetséges X halmazhoz számítsuk ki X^+-ot
  - Függőségekhez adjuk hozzá $\forall X \to A$-t, ahol $A \in X^+ \setminus X$
  - Dobjuk ki $XY \to A$-t, ha $X \to A$ is teljesül
  - Végül csak azokat az FF-eket nézzük, amikben csak a projektált attribútumok vannak
  - Trükk: $\emptyset^+ = \emptyset$ és $R^+ = R$
  - Trükk: $X^+ = R \wedge X \subseteq Y \implies Y^+ = R$

8.2.4. FF-ek geometriaia reprezentációja

- Minden pont egy lehetséges előfordulás (olyan sorhalmaz, ami létezhetne)
- Minden FF egy régió: előfordulások részhalmaza, amire igaz
- Triviális FF: a régió a teljes tér
- Több FF is teljesül: régiók metszete
- Legyen $A \rightarrow B$ és $B \rightarrow C$. Ekkor az $A \rightarrow C$ régi tartalmazza a két másik régió metszetét, de bővebb nála (valódi tartalmazás, nem egyenlőség).



8.3. Relációs sémák tervezése

8.3.1. Relációk felbontása

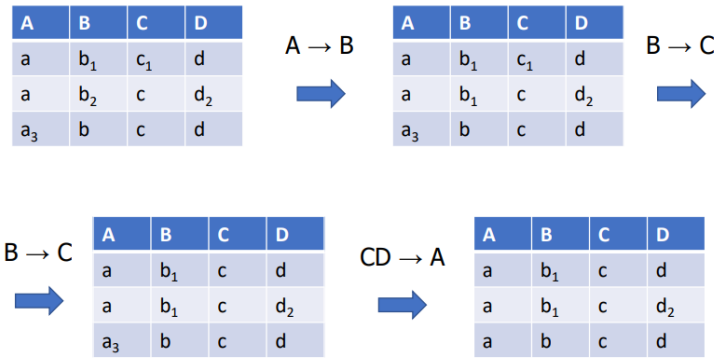
- Bemenet: $R(A_1, ..., A_n)$; kimenet: $S(B_1, ..., B_m), T(C_1, ..., C_k)$ relációk
  - $\{A_1, ..., A_n\} = \{B_1, ..., B_m\} \cup \{C_1, ..., C_k\}$
  - $S = \Pi_{B1,...,Bm}(R)$ és $T = \Pi_{C1,...,Ck}(R)$
- Cél: veszteségmentes felbontás: $r = \Pi_{R1}(r) \bowtie ... \bowtie \Pi_{Rk}(r)$
  - r: egy R sémájú reláció
  - $\Pi_{Ri}(r)$: r sorai az Ri attribútumaira (több van!) projektálva
  - Példa nem veszteségmentesre: ABC-t AB-re és BC-re bontjuk, de B oszlop minden eleme azonos, A és C elemei viszont nem
- Cél: funkcionális függőségek megőrzése
  - Nem mindig sikerül
- Cél: redundancia, anomáliák (felsorolva alul) megszüntetése
  - (Nem mindig sikerül)
  - Módosítási: nem mindenhol módosul az adat (redundancia miatt)
  - Törlési anomália: olyan is törlődik, aminek nem kéne
    pl. ha senki nem szeret egy sört, akkor tudjuk, hogy ki gyártja?
  - Beszúrási: nem tudunk beszúrni, mert olyan adatot kéne megadni, ami nincs értelmezve egy adott esetben (pl. kedvenc söre anyámnak)

8.3.2. Boyce-Codd normálforma (BCNF)

- R reláció BCNF formában van, ha $\forall X \to Y$ nemtriv FF-re X szuperkulcs
  - Nemtriviális: Y nem része X-nek
  - Szuperkulcs: tartalmaz kulcsot (akár ő maga egy kulcs)
- Veszteségmentes, de FF-ek nem feltétlenül őrződnek meg, de FF-ek okozta anomália nem lehet benne
- BCNF-re való felbontás
  - Legyen R reláció, F FF-ek halmaza
  - Van-e olyan $X \to Y$ FF, ami sérti a BCNF-et?
    - Elég F-et végignézni: ha F^+-ban valami sérti, akkor F-ben is
    - Hogyan: X^+-ban nincs minden attribútum $\implies$ nem szuperkulcs
  - R dekomponálása $X \to Y$ alapján
    - $R_1 = X^+$ és $R_2 = R - (X^+ - X)$
    - Projektáljuk F-beli FF-eket a két új relációra
    - Példa (F: név $\to$ cím, név $\to$ kedvenc, kedvelt $\to$ gyártó)
      - Bemenet: Főnökök(név, cím, kedveltSörök, gyártó, kedvenc)
      - név $\to$ cím: R1(név, cím, kedvenc) R2(név, kedvelt, gyártó)
      - Ez még mindig nem BCNF: $R2$-t ketté kell szedni
- Belátása, hogy a BCNF-re való felbontás veszteségmentes
  - Tranzitivitás féleség: ha $R_1, ..., R_k$ egy veszteségmentes felbontása R-nek, és S_1, S_2 egy veszteségmentes felbontása R_1-nek, akkor $S_1, S_2, R_2, ..., R_k$ egy veszteségmentes felbontása R-nek
  - Tehát csak azt kell belátni, hogy a dekomponálás veszteségmentes
  - Ez pedig triviális (legalábbis az EA diákon ez a magyarázat)

8.3.3. Chase-teszt veszteségmentességhez

- Megmondja, hogy veszteségmentes-e egy felbontás
- Példa eset:
  - Eredeti reláció: $R(A, B, C, D)$
  - FF-ek: $F = \{A \to B, B \to C, CD \to A\}$
  - Felbontás: $R_1(A, D)$, $R_2(A, C)$, $R_3(B, C, D)$
- Egy táblázatot használ az algoritmus:
  - Oszlopok az eredeti R oszlopai
  - Sorok: minden R_i-hez egy sor
  - Cellák: oszlop R_i része akkor ismert (x) egyébként ismeretlen (x_i)
- A cél az általános sor (a, b, c, d) kialakítása. Az FF-ek használjuk fel.
  - Két egyenlővé teendő szimbólum közül...
    - Egyik index nélküli: másik is azt az értéket kapja
    - Mindkét indexes: a kisebbik indexet kapja mindkettő
  - A lecserélt szimbólum minden előfordulását le kell cserélni
- Egyszer nem feltétlenül elég végigmenni az FF-eken: tetszőleges sorrendben vettük végig őket, ami lehet, hogy nem optimális. Addig kell újra-újra lefuttatni az "algoritmust", amíg változik az eredmény.
  - Ha megkaptuk az általános sort, akkor biztosan veszteségmentes. Ismételni csak akkor kell, ha nem kaptuk meg.



8.3.4. Minimális bázis

- Bázis: F-fel ekvivalens FF halmaz
- Minimális bázis
  - Jobb oldalon mindig csak egy attribútum van
  - Bármely függőség elhagyása esetén már nem bázis
  - Bármely bal oldal csökkentése esetén már nem bázis
- Minimális bázis létrehozása algoritmikusan: (legyen F az FF-ek halmaza)
  - Kezdetben $G = \emptyset$
  - Függőségek $X \to A$ alakra hozása (jobboldalak szétvágása):
    $\forall\, X \to Y \in F,\ A \in Y \setminus X$: az $X \to A$ FF-eket tegyük G-be
  - Nem szükséges függőségek elhagyása:
    $X \to A \in G$ elhagyása, ha $X \to A$ következik $(G \setminus \{X \to A\})$-ból
  - Baloldalak minimalizálása:
    Ha $X \to A \in G$ és $\exists B \in X : A \in (X \setminus \{B\})^+$ (G szerint), azaz G ekvivalens marad: G-ben $X \to A$ lecserélése $(X \setminus \{B\}) \to A$-ra

8.3.5. Harmadik normálforma (3NF)

- Motiváció: BCNF felbontáskor veszíthetünk el függőségeket
  - Példa: AC,BC sémákkal nem lehet az $AB \to C$ egy FF
- Veszteségmentes, megőrzi az FF-eket, de anomália maradhat
- Prím (elsődleges) attribútum: legalább egy kulcsnak az eleme
- Definíció: $X \to A$ nem triviális FF csak akkor sérti meg 3NF-et, ha X nem szuperkulcs és A nem prím
  - BCNF-től eltérés: FF akkor is szabályos, ha jobboldal prím
- Minimális bázisból 3NF-re bontás
  - Minimális bázis minden FF-jének bal- és jobboldalának uniója egy séma, pl.: $X \to A$ esetén XA egy séma
  - Ha az így kapott sémák között nincs szuperkulcs, akkor egy extra séma: valami, ami egy kulcs az R reláción
    - Ez a veszteségmentességhez kell

- Helyesség belátása:
  - Függőségeket megőrizni a minimális bázis miatt
  - Veszteségmentes, amit a Chase-tesztel be lehet látni

8.4. Többértékű függőségek

8.4.1. Többértékű függőség (TÉF)

- Jelölés: $X \to\to Y$
- X minden értékei esetén Y értékei függetlenek az $R \setminus X \setminus Y$ értékeitől
  - Azaz ha két sorban X értékei egyeznek $\implies$ Y értékei felcserélhetők
- Példa: Alkesz(név, tel, kedveltSör): telefonszám független a sörtől
  - név $\to\to$ tel és név $\to\to$ kedveltSör
  - Azaz egy adott névhez tartozó telefonok, sörök minden kombinációja létezik a sorok között
- Triviális TÉF: $X \to\to Y$ ahol $Y \subseteq X$ vagy $X \cup Y = R \quad (X, Y \subseteq R)$

8.4.2. Többértékű függőség szabályok

- Minden FF egyben TÉF: $X \to Y \implies X \to\to Y$
  - Ha két sor egyezik X-en, akkor Y-on; felcserélve ugyan azt kapjuk
- Komplementálás: $X \to\to Y \implies X \to\to (R \setminus X \setminus Y)$
  - X megegyezik, akkor mindegy: Y-t cseréljük fel / minden mást
- FF-ekkel ellentétben a jobboldalak nem darabolhatók szabadon
- Bővíthetőség: ha $X \to\to Y$ és $V \subseteq W$ akkor $XW \to\to YV$
- Tranzitivitás: $X \to\to Y \wedge Y \to\to S \implies X \to\to S \setminus Y$

8.4.3. Negyedik normálforma (4NF)

- Motiváció: TÉF-ek okozta redundanciát BCNF nem szünteti meg
- 4NF: TÉF-eket dekomponáláskor FF-ként kezeljük
  (kulcs kereséskor a TÉF-ek nem számítanak)
- R akkor van 4NF-ben, ha $\forall$ nemtriv. $X \to\to Y$ esetén X szuperkulcs
- R 4NF-ben van $\implies$ BCNF-ben is (fordítva nem igaz)
  - Minden FF, ami BCNF-et megsérti, az a 4NF-et is megsérti
- 4NF dekompozíció, ha $X \to\to Y$ sérti 4NF-et: (hasonló BCNF-hez)
  $R_1 = XY$ és $R_2 = R \setminus (Y \setminus X)$

8.4.4. TÉF-ek és FF-ek együttes következtetése

- Feladat: egy TÉF, FF halmazból következik-e egy adott TÉF/FF?
- Megoldás: tabló, Chase ötlet kiterjesztése
- Chase-teszt kiterjesztése
- TÉF esetén írjuk be a szükséges sorokat, hogy a TÉF igaz legyen
- Vagy ha $X \to\to Y$ bizonyítása a feladat, akkor kezdjünk 2 X-ben egyező és $R \setminus X$-ben különböző sorral és lássuk be (FF-ek és TÉF-ek segítségével), hogy a két sor megegyezik
- Nem sikerült felfognom a példák levezetését, így ezen témakör jegyzete hiányos.

9. Egyed-kapcsolat modell, E/K diagram

9.1. Cél, motiváció

- Adatbázisséma felvázolása
- Később relációs adatbázissémává alakítható
- Tervezést segíti (főbb részek felvázolása)

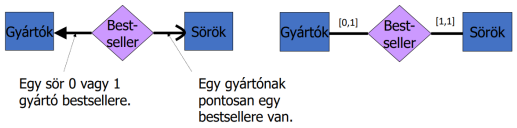
9.2. Alafogalmak

- Egyed
- Egyedhalmaz (téglalap): mint OOP-ben egy osztály
  - Csak struktúrát tekintünk, végezhető műveleteket nem
  - Értéke: hozzá tartozó egyedek halmaza
- Attribútum (ovális): atomi érték (szám, szöveg) (tömb/rekord NEM!)
  - Össze vannak kötve egyedhalmazzal
- Kapcsolat (rombusz): 2 (bináris) vagy több egyedhalmazt köt össze
  - Értéke a kapcsolathalmaz: sorok halmaza, minden sorban a kapcsolatban résztvevő egyedhalmazokból van 1-1 darab

9.3. Kapcsolatokról bővebben

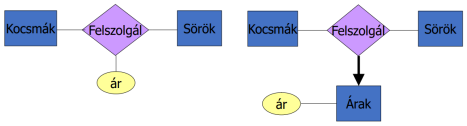
9.3.1. Kapcsolatok típusai

- Általában bináris kapcsolatról beszélünk most
- Sok-sok: mindkét adathalmaz egyedei 0/1/sok másik egyedhez kapcsolódhatnak
  - Jelölése: sima vonal
- Sok-egy: (jelölése: "egy" oldalon háromszög nyíl van)
  - Első egyedhalmaz minden eleme legfeljebb egyhez kapcsolódnak
  - Második egyedhalmaz elemei 0, 1 vagy több egyedhez kapcsolódhatnak
- Egy-egy: minden entitáns legfeljebb egy másikhoz kapcsolódik
  - Jelölése: mindkét végén háromszög nyíl
- "Legfeljebb" helyett "pontosan" egy: lekerekített nyíl ($-\triangleright$ helyett $\rightarrow$)
- Alternatív jelölés: él címkézése [a,b] intervallummal (sok=n)



9.3.2. Kapcsolatok attribútumai

- Az attribútum a kapcsolatban álló egyedhalmazok együttes tulajdonsága
- Jelölés: simán hozzákötünk egy attribútumot a kapcsolathoz
- Alternatív jelölés: egyedhalmaz bevezetése és vastag háromszöges nyíllal rámutatunk a kapcsolatból



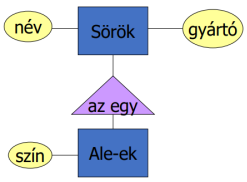
9.3.3. Kapcsolat szerepek

- Szerep: felcímkézett él
- Szükséges, ha egy egyedhalmaz többszörös tagja kapcsolatnak

9.4. Egyedhalmazokról bővebben

9.4.1. Alosztály

- Speciális eset, kevesebb egyede van, több tulajdonsága
- Azaz extra attribútumokat vezethetünk be hozzájuk
- Viszont az alosztály egyedei az őszosztály egyedei is egyben
- Nem engedjük a többszörös öröklődést: alosztályok rendszere egy fa
- Jelölés: ősosztályra mutató háromszög, beleírva "az egy"

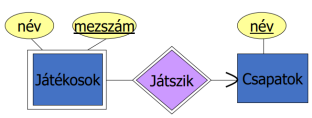


9.4.2. Kulcs

- Kulcs: attribútumok halmaza, amelyekre nincs két olyan egyed, hogy a kulcs minden attribútuma azonos legyen
- Minden egyedhalmaznak van kulcsa
- Jelölés: attribútum nevének aláhúzásával
- Alosztálynak nincs külön kulcsa: ősosztály kulcsa vonatkozik rá

9.4.3. Gyenge egyedhalmaz

- Egyedeit csak külső segítséggel lehet egyértelműen azonosítani: egy/több sok-egy kapcsolatot kell követni és a kapott egyedek kulcsértékei kellenek
- Gyenge egyedhalmaz jelölése: dupla szélű téglalap
- Támogató sok-egy kapcsolat jelölése: dupla szélű rombusz

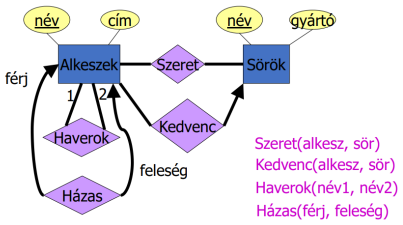


9.5. Tervezési technikák

- Redundancia elkerülése
  - Helypazarlás
  - Növeli az inkonzisztencia veszélyét
- Gyenge egyedhalmazok óvatos használata
  - Általában jó, ha minden egyedhalmaznak van saját kulcsa
  - Saját kulcs készítése nem mindig lehetséges: nincs, ami kiosztaná ezeket (decentralizált rendszerek)
- Attribútum használata egyedhalmaz helyett, ha lehetséges
  - Egyedhalmaz kell, ha ezek közül egy teljesül:
    - van legalább egy nem kulcs attribútum benne
    - egy "sok-egy" kapcsolatnak a "sok" végén szerepel

9.6. E/K diagrammból relációséma

- Egyedhalmazból reláció, attribútumból attribútum
- Kapcsolatból reláció, aminek attribútumai:
  - összekapcsolt egyedhalmazok kulcsai
  - kapcsolat saját attribútumai



- Relációk összevonása: egyedhalmaz relációja összevonható az egyedhalmazzal sok-egy kapcsolatban álló reláció, ha az eredeti egyedhalmaz a "sok", a másik pedig az "egy" oldalon van
  - Sok-sok kapcsolat összevonása helytelen, redundanciához vezet

- Gyenge egyedhalmazból reláció:
  - Relációnak a teljes kulcsot tartalmaznia kell.
  - Támogató kapcsolatokat nem írjuk át (redundanciához vezetne)
- Alosztályok átalakítása
  - NULL értékek használata
    * Alosztály nem hoz be újabb relációt, csak újabb oszlopo(ka)t
    * Helytakarékos, kivéve, ha sok a NULL érték
    * Összekapcsolásokat is megspórolja
  - Objektumorientált megközelítés
    * Ősosztálynak és alosztálynak külön reláció
    * Ősosztály minden oszlopa létezik az alosztályban is
    * Egy egyed csak az egyik relációban van benne
    * Hasznos, ha az ősosztály nem-kulcs az alosztály oszlopait érintő lekérdezésekhez
  - E/K style
    * Ősosztálynak és alosztálynak külön reláció
    * Ősosztálynak csak a kulcs oszlopai léteznek az alosztály relációban
    * Egy egyed mindkét relációban benne van
    * Összekapcsolás szükséges, hogy minden adat egyszerre meglegyen
    * Hasznos, ha az alosztályt csak ritkán használjuk

10. Objektum-relációs adatbázisok

10.1. Bevezetés

- Relációs és az objektumorientált modell egyesítése
  - OO modell komplexebb adattípusokat is támogat
  - Relációs modell magas szintű lekérdezéseket támogat
  - Objektum-relációs adatmodell ezt a kettőt akarja megtartani
- Adatbázis-kezelő rendszerek (DBMS) fejlődése
  - OO rendszerek sokáig nem voltak elég hatékonyak
  - Relációs DBMS OO-val való kiterjesztése így jó megoldás volt
- Főbb különbségek:
  - Attribútum típusa nem csak atomi típus lehet
  - Metódusok
  - Sorok egyéni azonosítót kapnak, lehet rájuk hivatkozást készíteni

10.2. Hivatkozások

- Cél: redundancia, módosítási anomáliák kiküszöbölése
  - Áttekinthetőbb alternatíva egy kapcsolási táblára
- Hivatkozás típus rekurzívan van definiálva
- A attribútum R sémájú sorra történő...
  - hivatkozás jelölés: A(*R)
  - hivatkozás halmaz (több sorra hivatkozás) jelölés: A({*R})
- Ha T egy UDT, akkor REF T egy hivatkozás (ami egy típus)
- OID (objektum azonosító): beépített azonosító, alapból rejtett
- Oracle: nincs különbség hivatkozás követés és normál mező elérés között
  - De ha `ff.sör` egy hivatkozás, akkor `SELECT DEEREF(ff.sör)` kell
- SQL-99: ha `ff` egy reláció alias, akkor `ff.sör()->név`
- Beszúrás: `SELECT típus(REF(<alias>), ... FROM ....` segítségével

10.3. Felhasználó által definiált adattípus: User Defined Type, UDT

- Hasonló OOP-hez: adatszerkezet és annak metódusai
- Oracle: `CREATE TYPE <típusnév> AS OBJECT (<név-típus pár lista>);`
  - SQL szabvány: AS után nincsen OBJECT
- 1. felhasználási mód: sortípus: reláció ilyen adatokat tárol
  - Egy oszlop van a táblában, azon belül több mező
  - `CREATE TABLE <név> OF <UDT>;`
  - Lekérdezés: `SELECT <alias.xy> FROM <reláció> <alias>`
    - Oracle esetén az alias mindig kötelező
    - SQL-99 esetén `xy` helyett `xy()` (egy generátor) kell
  - Beszúrás
    - Oracle: sima `INSERT`, konstruktor hívással
    - SQL-99: mutátorok (`R.A(42)`) kellenek
      - Először létre kell hozni egy változót: `SET v = MyType();`
      - Utána mutálni kell: `v.x(42); v.y("valami");`
      - Végül be kell szúrni: `INSERT INTO MyTable VALUES(v);`
- 2. felhasználási mód: egy reláció attribútumának a típusa

10.4. Metódusok UDT-ken belül

- Oracle:
  - CREATE TYPE-ban kell deklarálni, mint egy attribútumot:
    `MEMBER FUNCTION <név>(<paraméterek>) RETURN <típus>, <pragma>`
  - `PRAGMA RESTRICT_REFERENCES(<metódus>, WNDS)`
    - WNDS jelentése: nem módosíthat, csak read-only
    - RNDS jelentése: ki sem olvassa az adatokat
  - Definíció: `CREATE TYPE BODY <típus> AS MEMBER FUNCTION <metódus>(...) RETURN <típus> IS BEGIN ... END; END;`
    - Paramétereknél itt már nem kell `IN/OUT`
  - Saját objektum: SELF

10.4.1. Rendező metódusok

- Ezek segítségével működik az `ORDER BY, WHERE` lekérdezésben
- SQL-99: EQUAL, LESSTHAN (logikai visszatérési érték)
- Oracle: bármelyik lehet rendező metódus, nincs kötelező neve
  - ORDER prefix kell MEMBER FUNCTION elé deklarációnál
  - Kötelező pragma-k: WNDS, RNDS, WNPF, RNPS
  - Visszatérési érték: szám (<0, $=0$, >0)

10.5. Beágyazott táblák

- Lényeg: oszlop típusa egy táblázat
- Séma típusú attribútum jelölés: AttrNév(AttrTag1, AttrTag2)
  - Emlékeztető: ez nem egy tuple/struct, hanem egy táblázat
- Séma típus rekurzívan van definiálva (kiindulás: atomi típus)
- Példa: Oktatók(név, TAJ, címek(város, utca), tárgyak(tárgynév, kód))
- Csak Oracle szintaxist nézünk
- Tábla típus létrehozása: `CREATE TYPE S AS TABLE OF T`
- Beágyazott relációk tárolása Oracle-ben:
  - "Fő" tábla létrehozásakor megadni az oszlop-tábla hol legyen tárolva
  - `CREATE TABLE Gyártók (..., sörök SörTáblaTípus)`
    `NESTED TABLE sörök STORE AS SörTábla;`
- Beszúrás: tábla konstruktora, utána sorok felsorolása
- Beágyazott tábla lekérdezése:
  - Minta sima érték lenne, csak egy listát kapunk vissza, mint "skalárt"
  - Több beágyazott tábla esetén descartes szorzat a beágyazott táblákra
- Beágyazott táblán belüli lekérdezés
  - Hagyományos táblává alakítás: `TABLE(SELECT .. FROM ..) <alias>`
- Beágyazott táblává alakítása (pl. beszúráshoz):
  `CAST(MULTISET(SELECT ...) AS <type>)`

11. XML és XML/DTD sémák

11.1. Félig strukturált modellek (semi-structured data)

- Félig strukturált modellek: XML, JSON, ...
- Félig strukturált
  - Motiváció: adatok rugalmas megjelenítése
  - Motiváció: dokumentumok átadása rendszerek (/adatbázisok) között
  - Információt ad arról, hogy mi a séma (magán az adaton felül)
- Gráfként felfogható
  - Csúcsok: objektumok
  - Élek: kapcsolatok, címke a kapcsolat neve
  - Levél: atomi értékek

11.2. XML (Extensible Markup Language) bevezetés

- Kezdő deklarációs sor: `<?xml version="1.0" encoding="utf-8" ?>`
- Tagek (jelölők): `<tag> ... </tag>`
  - Tetszőlegesen egymásba ágyazhatók
  - Üres tag: `<tag/>`
  - Érzékenyek a kis- és nagybetű különbségekre
- Jól formált XML:
  - Önálló tagek bevezetése megengedett
  - Nem hiányzik a deklarációs sor
  - Minden nyitó tagnak van záró párja + jó sorrend, "helyes zárójelezés"
- Valid XML: egy előre adott sémának megfelel, pl. DTD séma, XML séma

11.3. XML felhasználása

- Információintegrációs probléma:
  - adatbázis modellek (relációs, NoSQL, stb) eltérnek
  - sémák eltérnek
  - más mértékegység, stb.
- Régi, örökölt adatbázisokra támaszkodnak régi alkalmazások, ezeket meg akarjuk tartani
  - Megoldás: interfészt vezetünk be régi adatbázisok fölé
  - Új felhasználó az interfészen keresztül lép kapcsolatba az adattal
  - Az interfész közös több örökölt adatbázison
- Integrációra két megközelítés: (mi legyen az interfész)
  - Adattárházba naponta/hetente szinkronizáció
    - Egy csomagoló adatbázisból adattárházba viszi az adatot
    - Egy irányú a megoldás: adattárházból nem lehet módosítani
      - Ez biztos?
  - Nézetek létrehozása, mintha integrált rendszer részei lennének
    - Mediátor csomagolókon keresztül beszél az adatbázisokkal
    - Két irányú a kapcsolat: lekérdezés és módosítás is lehet

11.4. DTD séma

- Speciális formátumot használ (nem XML)
- Felhasználás
  - <!DOCTYPE ...> a deklarációs sor és a dokumentum közé illeszthető
  - külön fájl: <!DOCTYPE kocsmák SYSTEM "bar.dtd">
- Elemek: `<!ELEMENT <név> (alelemek...)`
  - Egy elemben lehet adat (levél, `#PCDATA`) vagy elemek (vagy üres)
  - Üres elem: <!ELEMENT valami EMPTY> (tag: <valami <attr...>/>)
  - Alelemek megadása: típus és utána opcionálisan multiplicitás (*,+,?)
  - Alelemeknél van VAGY (|): `<!ELEMENT név ((x,y) | (a,b))>`
- Attribútumok: `<!ATTLLIST Elem A1 Type1 Mod1 A2 Type2 Mod2...>`
  - "Mod": #IMPLIED (opcionális), #REQUIRED (kötelező)
    - #DEFAULT (hogyan kell használni? nem hangzott el)
    - #FIXED (mindig egy adott értéket kell megadni... valahogy)
  - Adattípusok: CDATA, ID, IDREF, IDREFS
    - CDATA: szöveg
    - ID, IDREF: egymásra mutathatnak az adatok, emiatt gráfot alkothatnak (nem pedig fát)
    - ID: azonos típusú elemnek nem lehet ugyan ilyen értéke (unique)
    - IDREF: nem tudjuk megmondani, hogy milyen típusú elemre mutasson
    - IDREFS: egyszerre többre hivatkozás
- ```
  <!DOCTYPE kocsmák [
    <!ELEMENT kocsmák (kocsma*, sör*)>
    <!ELEMENT kocsma (felszolgál+)>
      <!ATTLIST kocsma név ID #REQUIRED>
    <!ELEMENT felszolgál (#PCDATA)>
      <!ATTLIST felszolgál melyikSör IDREF #REQUIRED>
    <!ELEMENT sör EMPTY>
      <!ATTLIST sör név ID #REQUIRED>
      <!ATTLIST sör kapható IDREFS #IMPLIED>
  ]>
  ```
  ```

11.5. XML séma

- XML formátum, így az első sor: `<?xml version=... ?>`
- Root tag: `<xs:schema xmlns:xs="URL"> ... </xs:schema>`
- Felhasználás: <valami xmlns:xsi="schema-URL" xsi:noNamespaceSchemaLocation="valami.xsd"> ... </valami>

11.5.1. Egyszerű element (xs:element)

- Kötelező attribútum: name (tag neve)
- Kötelező attribútum: type (érték típusa, pl. xs:string)
- Nincs alelem

11.5.2. Összetett típusok (nem element, csak típus)

- Nem kötelező nevesíteni, lehet egyből egy xs:element aleleme (pl. ha csak egyszer használjuk)
- Összetett (alelemet tartalmazó) típusok: xs:complexType
  - Kötelező attribútum: name (tag neve)
  - Típus: alelemek írják le
- xs:sequence: xs:element sorozata
  - minOccurs és maxOccurs attribútumok minden xs:element-en
  - occurs értékek: szám (0,1,stb.) vagy unbounded
- xs:attribute: name, type, use="required/optional" (alapból optional)

11.5.3. Megszorítás típusok (nem element, csak típus)

- Gyökér: xs:simpleType name=...
- Alelem: `<xs:restriction base=típus> ... </xs:restriction>`
  - Típus: milyen típusú elemek lehetnek, mit szorítunk meg
  - `{min/max}{Inclusive/Exclusive}`: alsó/felső korlát megadása
- Lehetséges elemek felsorolása: 1-1 `xs:enumeration value="..."` alelem

11.5.4. Kulcsok

- xs:element-hez tartozhat xs:key alelem: `<xs:key name="..."> ...`
  - <xs:selector xpath="kocsma" />
  - <xs:field xpath="@nev" /> (@ jelentése: attribútum)
  - Jelentése: szelektor által megadott elemek adott field-jének mind unique-nak kell lennie
- xs:element-hez tartozhat: `<xs:keyref name="..." refer="kulcs">`:
  - xs:key-hez hasonlóan selector és field beállítás kell
  - Jelentése: bizonyos értékeknek egy létező kulcsot kell leírniuk

```
<?xml version = "1.0" encoding = "utf-8" ?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
    <xs:complexType name = "sörTípus">
        <xs:attribute name = "név" type = "xs:string" use = "required" />
    </xs:complexType>
    <xs:complexType name = "kocsmaTípus">
        <xs:sequence>
            <xs:element name = "sör" type = "sörTípus"
                minOccurs = "1" maxOccurs = "unbounded" />
        </xs:sequence>
        <xs:attribute name = "név" type = "xs:string" use = "required" />
    </xs:complexType>
    <xs:complexType name = "kocsmákTípus" >
        <xs:sequence>
            <xs:element name = "kocsma" type = "kocsmaTípus"
                minOccurs = "0" maxOccurs = "unbounded" />
        </xs:sequence>
    </xs:complexType>
    <xs:element name = "kocsmák" type = "kocsmákTípus" />
</xs:schema>
```

12. XML lekérdezőnyelvek

- XPath/XQuery adatmodell
  - Tételek listájával (item sequence) dolgozunk: ez a bemenet, kimenet
  - Tétel lehet: egyszerű (levél, skalár érték) vagy csomópont
- Csomópontok:
  - dokumentum csomópont: teljes dokumentum (pl. doc(http://...))
  - elem csomópont: xml tag és az alelemei
  - attribútum

12.1. XPath

- Emlékeztető: eredmény mindig egy lista
- Tag-ek követése: /kocsmák/kocsma/ár (alemeket járunk be sorban)
- Attribútum hivatkozása: @ prefix
- Nem csak a gyökérből lehet indulni:
  - //X/... a dokumentumban bárhol (gyökérben vagy lejjebb) lévő X tag-től indul
  - Eredmény sorrendben lesz: feljebb (pl. gyökérből) induló esetek lesznek először
  - Nem csak gyökér szinten lehet alkalmazni, hanem lejjebb is
- Tetszőleges tag: * (pl. /*/*/ár)
- Feltételek: tag után [...]
  - Atomi értékű törzsre szűrés: [text()]
  - Szám összehasonlítás: [. $<$ 3.14] (. az aktuális elem)
  - Attribútum megszorítás: [@valami = "xyz"]

- Tengelyek (axes): következő lépésnél feldolgozandóak listája
  - Alapértelmezett: `child::` (pl. /child::kocsmák)
  - attribute:: (@ hosszú formája)
  - parent:: (pl. //könyv/cím[. = 'Valami']/parent::könyv)
  - descendant-or-self:: (hasonló a //-hez)
  - ancestor::, ancestor-or-self::
  - self:: (. hosszú formája)

13. XQuery

- Nem vettük, nem maradt rá idő

14. Relációs algebra gyakorlati jegyzet

- $\sigma_{x=NULL}$ az nem része az alap algrebrának, csak a kiterjesztett algebrának
- Relációs algebra mindig átírható SQL-be, de visszafelé nem
  - pl. alkérdések nincsenek relációs algebrában
- Tábla indexelve: átnevezés, csak így könnyebb (hivatalosan: $\rho_{Sz_1}(Sz)$)
- Min két gyümölcsöt szeret: $Sz_1 \times Sz_2$, 1 sorban 2 különböző gyümölcs
- Gyakorlaton $\bowtie\!\!\!\sqsubset$ helyett $\bowtie_R$ jelet használjuk (teljes: ${}_L\bowtie_R$)
- Komplementer halmaz képzés: $(\Pi_{name}(SZ) \times \Pi_{fruit}(SZ)) \setminus (SZ)$
- Tábla nevét $\bowtie$-nál mindig ki kell írni, kivéve, ha NATURAL JOIN van

14.1. Kiterjesztett algebra

- Multihalmazzal dolgozunk
- Létezik NULL érték
- Csoportképzés (group by féleség): γ (gamma)
  - GROUP BY X, COUNT(Y) AS db: $\gamma_{X,COUNT(Y)\to db}(R)$
  - Aggregációs függvények: MIN, MAX, SUM, AVG, COUNT
  - Ezután csak az alsó indexben lévőket lehet használni
  - Lehet 0/1/több oszlop alapján group-olni, lehet 0/1/többet aggregálni
  - Relációs algebrában nincs külön WHERE ÉS HAVING (csak σ)
  - Relációs algebrában is van COUNT(*)
  - COUNT(DISTINCT X) = COUNT(δ X)
- Sorba rendezés: τ (tau), ORDER BY X: $\tau_X(R)$

15. SQL függvények, műveletek

- Dátum, idő
  - TO_DATE('2000-01-01', 'YYYY-MM-DD') -> dátum
  - TO_CHAR(DATE, 'YYYY-MM-DD') -> szöveg
  - SYSDATE -> aktuális dátum (NEM FÜGGVÉNY)
  - Dátumok kivonása a napok számát adja meg
- Szöveg (string) kezelés
  - Oracle 1-től indexel
  - SUBSTR('valami', 2, 1) -> a
  - SUBSTR('valami', -2, 1) -> m
  - SUBSTR('valami', 2) -> 'alami'
  - LENGTH('alma') -> 4
  - INSTR('valami', 'a', 1, 2) -> 2. 'a' pozíciója az 1-es index után
  - RPAD('a', 4, '#') -> 'a###', LPAD, üres szöveget nem lehet
  - UPPER('alma') -> 'ALMA', LOWER
  - mezo (NOT) LIKE '_I%' -> 2. betű I, hossza >= 2
  - Konkatenáció: `'a' || 'b'`
- Matematika
  - ROUND(0.123, 2) -> 0.12
  - SQRT(x), FLOOR(x), CEIL(x)
  - MOD(x, 2)
  - LEAST(a, b), GREATEST(a, b)
- NVL(NULL, 'Ismeretlen', ...) -> első nem-NULL érték
- TO_NUMBER('42') és TO_CHAR(42)

16. PL/SQL (PL SQL, PLSQL) programozás

16.1. Függvény, procedura létrehozás

- Sql Developer buta és nem ismeri fel, hogy hol ér véget a fv
  - Tehát ki kell jelölni a fv-t és csak az fv-t és úgy futtatni
- `CREATE OR REPLACE PROCEDURE <név> (params...) IS`
  `<változó> <típus> [:= kezdőérték];`
  `BEGIN`
  `... --Minden sor végén ;`
  `END [név];`
  - Hívása: CALL <név>(params...);
- `CREATE OR REPLACE FUNCTION <név> (params...) RETURN <típus> IS ...`
  - Hívás: SELECT <név>(params1), <név>(params2) FROM dual;
- Paraméterek: pl. [IN] [OUT] nev VARCHAR2 (nincs méret!)
  - IN (bemeneti): alapból minden ez, nem lehet módosítani
  - IN OUT: konstans nem adható át ennek

16.2. Nyelvi konstrukciók

- For ciklus: FOR i IN 1..n LOOP ... END LOOP;
  - Másik irányban: ... IN REVERSE ...
- While ciklus: WHILE n > 0 LOOP ... END LOOP;
- IF <felt> THEN ... [ELSIF <felt> THEN ...] [ELSE ...] END IF;
- Loop-ból kilépés: `EXIT WHEN ...`
- Early "return": `IF ... THEN CONTINUE; END IF;`

16.3. Egyéb tudnivalók, trükkök

- `dbms_output.put_line('Hello' || 'nev'); --debug msg`
  - Be kell kapcsolni: `SET SERVEROUTPUT ON` (egy ideig hat)
- Név nélküli blokk, függvény hívás egyszerűen:
  `DECLARE n := 42; BEGIN test(n); END;`
- Szöveg típus: `VARCHAR2`, ami üres esetén `NULL`
  - Általában nem baj, ha a függvényünk ilyen bemenetre nem működik
  - Ha üreshez akarunk konkatenálni, lehet érdemes inkább `VARCHAR1`
- Tábla másolás, extra oszlop létrehozásával:
  `CREATE TABLE ... AS SELECT dolgozo.*, 1 sorszam FROM dolgozo;`

16.4. Adat bekérés, prompt, `ACCEPT`

- Egy makró-t töltünk fel (ami egy C-s `#define` féleség)
- Hasznos ZH-n név nélküli blokkok tesztelésére
- `ACCEPT <név> <típus> PROMPT 'Popup prompt szöveg';`
- Makró használat: `&<név>`
  - Ha makró szöveg volt, attól még idézőjelek nincsenek ebben benne
  - Ilyenkor használat: `'&<név>'`
- `ACCEPT` sor nélkül is jó, de ekkor minden futtatáskor bekéri az értékét

16.5. Kurzorok (CURSOR)

```
CREATE OR REPLACE PROCEDURE teszt IS
  CURSOR curs1 IS SELECT * FROM SZ;
  rec curs1%rowtype; --1. lehetőséghez kell
BEGIN
  --1. lehetőség: bonyolult, több kurzoros feladatra is jó
  OPEN curs1;
  LOOP
    FETCH curs1 INTO rec;
    EXIT WHEN curs1%NOTFOUND;
    -- valami(rec.adattag);
  END LOOP;
  CLOSE curs1;

  --2. lehetőség, nem kell a fenti 'rec' deklaráció
  FOR rec IN curs1 LOOP
    -- valami(rec.adattag);
  END LOOP;
END;
```

16.5.1. Kurzorok módosításra

- Deklarálás: `CURSOR curs1 IS ... FOR UPDATE;`
  - UPDATE: akár delete-elni is lehet, nincs köze az UPDATE kulcsszóhoz
- Lehet több olvasó, de egyszerre csak 1 író (és írás kizárja az olvasást)
  - `FOR UPDATE` nélküli eljárásban hívhatunk meg módosító eljárást
- Cache-elt ResultSet-en iterálunk: iterálás közbeni változtatás nem látszik
- Változtatás SQL-lel: `UPDATE ... SET ... WHERE CURRENT OF curs1;`
  - `rec` és tábla módosítása nem módosítja egymást
  - Manuálisan frissíthetjük a `rec`-et, majd azzal a táblát
- Kapcsolt táblán iterálás, egyiken módosítás:
  - `FOR UPDATE` helyett `FOR UPDATE OF <oszlopnév...>`
  - Ez teljes zárat ad a táblára, nem csak az oszlopra (törölni is lehet)
- Módosító `FUNCTION`-t `SELECT` helyett név nélküli blokkban futtatunk

17. Példák

17.1. Tábla létrehozás

```
DROP TABLE teszt_1;
CREATE TABLE teszt_1 (
  lebego REAL NOT NULL,
  egesz INT DEFAULT 0,
  nev VARCHAR2(100) UNIQUE,
  CONSTRAINT pk_teszt_1 PRIMARY KEY (egesz, nev)
);

DROP TABLE teszt_2;
CREATE TABLE teszt_2 (
  datum DATE PRIMARY KEY
);

DROP TABLE teszt_3;
CREATE TABLE teszt_3 (
  egesz INT,
  nev VARCHAR2(100),
  datum DATE REFERENCES teszt_2 (datum),
  CONSTRAINT fk_teszt_1 FOREIGN KEY (egesz, nev)
                        REFERENCES teszt_1 (egesz, nev)
);
```

17.2. PL/SQL függvény

```
CREATE OR REPLACE FUNCTION faktor(n INTEGER) RETURN INTEGER IS
  val INTEGER := 1;
BEGIN
  FOR i IN 2..n LOOP
    val := val * i;
  END LOOP;
  RETURN val;
END;
SELECT faktor(0), faktor(5) FROM dual;
```

17.3. PL/SQL eljárás

```
CREATE OR REPLACE PROCEDURE paratlan IS
  CURSOR curs1 IS SELECT * FROM dolgozo ORDER BY dnev;
  sorszam INT := 0;
BEGIN
  FOR rec IN curs1 LOOP
    sorszam := sorszam + 1;
    IF MOD(sorszam, 2) = 1 THEN
      dbms_output.put_line(TO_CHAR(sorszam) || '.: Név: '
          || rec.dnev || ', fizetés: ' || TO_CHAR(rec.fizetes));
    END IF;
  END LOOP;
END;
SET SERVEROUTPUT ON;
CALL paratlan();
```