1-2. óra

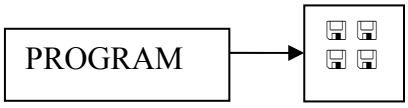
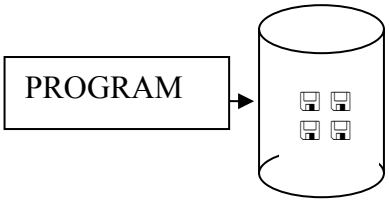
Adatbázis-kezelőkről

A felhasználó adatbázis-kezelőn keresztül éri el az adatokat.
Egy adatbázis-kezelő rendszerrel szemben a következő elvárásaink vannak:

1. Tegye lehetővé a felhasználók számára, hogy új adatbázisokat hozhassanak létre és azok sémáját, vagyis az adatok logikai struktúráját egy speciális nyelven adhassák meg. Ezt a speciális nyelvet *adatdefiníciós nyelv*nek nevezzük.
2. Engedje meg a felhasználóknak, hogy az adatokat egy megfelelő nyelv segítségével lekérdezhessék és módosíthassák. Ezt a nyelvet szokás *lekérdezőnyelv*nek vagy *adatmanipulációs nyelv*nek nevezni.
3. Támogassa nagyon *nagy mennyiségű adat* (gigabájtok vagy még több adat) hosszú időn keresztül való *tárolásá*t, garantálja az *adatok biztonságá*t a meghibásodásokkal és az illetéktelen felhasználókkal szemben, és tegye lehetővé a *hatékony adathozzáférés*t a lekérdezések és az adatbázis-módosítások számára.
4. Felügyelje a több felhasználó által egy időben történő adathozzáféréseket úgy, hogy az egyes felhasználók műveletei ne legyenek hatással a többi felhasználóra és az egyidejű adathozzáférések ne vezethessenek az adatok hibássá vagy következetlenné válásához. (Tranzakció kezelés)

(Ullman-Windom: Adatbázisrendszerek Alapvetés: 1. fejezet)

Fájl-kezelő és adatbázis-kezelő rendszerek összehasonlítása

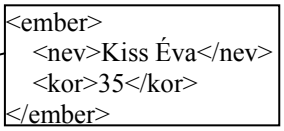
Fájl-kezelő rendszer	Adatbázis-kezelő rendszer
dBase, Clipper, FoxBase, FoxPro, Access (nem tesz eleget pl. a 3.pontnak)	Oracle, MS SQL Server, IBM DB2, Informix, Sybase, MySQL, PostgreSQL
nem felelnek meg minden követelménynek	**megfelelnek minden követelménynek**
foglalkozni kell a fizikai tárolással 	fizikai szinttel nem kell foglakozni (programunk egy másik programot szólít meg) 
egyidejűleg egy felhasználó használhatja	egyszerre több felhasználó is használhatja párhuzamosan
nincs adatvédelem, nincsenek jogosultságok	van adatvédelem, különböző jogosultságok vannak

fájl kezelő rendszer megkerülésével is hozzá lehet férni a fájlokhoz	módosításhoz adatbázis kezelő rendszer kell
nincs kapcsolat a fájlok között	táblák között van kapcsolat
rekordok sorrendje fontos	sorok sorrendje nem számít
ha megsérült egy fájl, nem tudom helyreállítani	bármilyen hiba keletkezik biztosítja a helyreállítást
szabadon törölhető egy fájl	táblák nem törölhetők egyszerűen

Adatmodellezésről általában

Az adatmodellezés célja, hogy a valós információk tárolására kitaláljunk valami olyasféle struktúrát, amiben az adatok információvesztés nélkül tárolhatók, az adatok közti kapcsolatok leírhatók és a struktúra a számítógépes feldolgozás szempontjából hatékony.

A modelljeinket többféle alapmodellre építhetjük. Egy modellt a jelölés rendszerével és műveleteivel határozhatunk meg. Néhány példa:



- hálós (történelmi jelentőségű)
  - jelölésrendszer: gráf
  - műveletek: adat definiáló nyelv
- hierarchikus (XML (dokumentum leíró nyelv) miatt újra aktuális)
  - jelölésrendszer: faszerkezet (csúcsok – adatok, élek - kapcsolatok)
  - műveletek: speciális nyelv
- relációs
  - jelölésrendszer: mátrix (=tábla =reláció)
  - műveletek: SQL nyelv

A relációs adatmodell

A relációs modellben az adatokat táblákban tároljuk. A „tábla” szó a megfelelő, azt, hogy „táblázat” nem használjuk (az a táblázatkezelés pl. Excel). Minden táblát egyedi neve alapján azonosítunk. Felépítésüket tekintve a táblák oszlopokból (vagy más néven attribútumokból vagy mezőkből) és sorokból (vagy más néven rekordokból) állnak. Minden oszlopnak táblán belül egyedi neve és meghatározott típusa van.

Minden cellában (azaz sor-oszlop metszetben) egy elemi érték szerepelhet. (Speciális érték a NULL érték, amelyet akkor alkalmazunk, ha egy cellát nem áll szándékunkban kitölteni. (NULL ≠ 0 és NULL ≠ ’’))

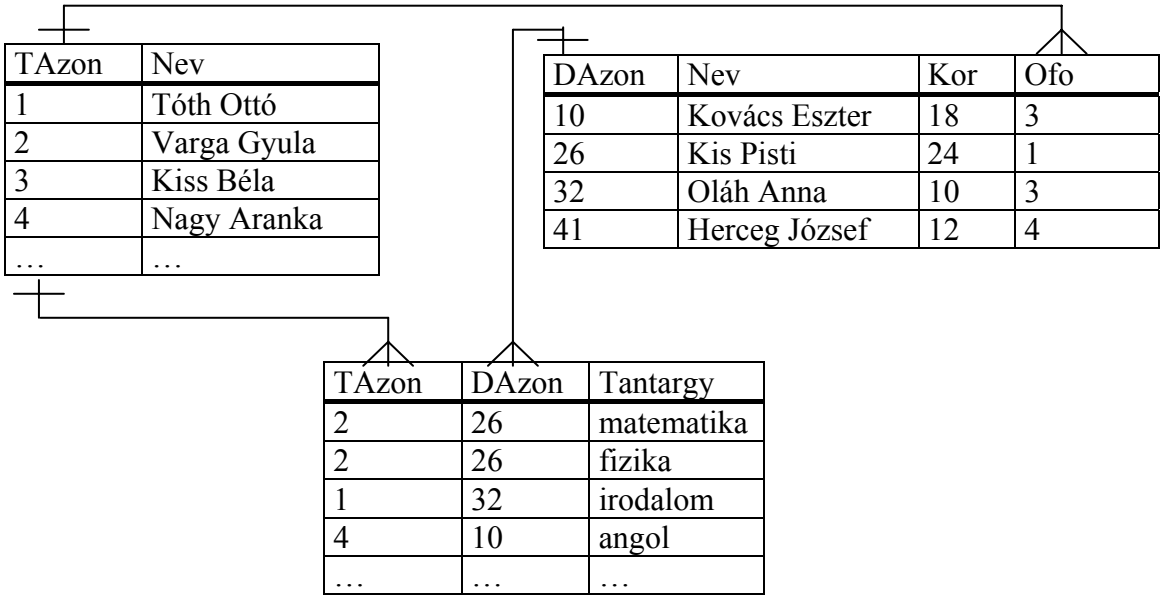
Általában létezik az oszlopoknak egy olyan kombinációja amely egyértelműen meghatározza bármelyik sort, ezt a kombinációt nevezzük kulcsnak. A „meghatározza” kifejezés alatt itt nem azt értjük, hogy a kulcsmező értékeiből kiszámítható a többi mező értéke! Ez valójában csak annyit jelent, hogy adott kulcsérték csak egyszer fordulhat elő.

A táblák pontosabban az egyedek között kapcsolatokat is megfogalmazhatunk (amiknek akár plusz tulajdonságaik is lehetnek). A kapcsolatok típusai:

- kapcsolat (pl.: ország - főváros) ⇨ attribútum
- több - egy kapcsolat (pl.: anya - gyerek) ⇨ idegen kulcs
- több - több kapcsolat (pl.: tanárok - diákok)
  ⇨ 1 kapcsoló tábla, 2db több - egy kapcsolat

A félév során többször használt példa adatbázisunkban, diákok és tanárok adatait tároljuk. A tTanar táblában a tanárok azonosítóját és nevét tároljuk, a tDiak táblában pedig a diákok azonosítóját, nevét és életkorát.

tTanar tábla (TAzon – egész, Nev – szöveg)
tDiak tábla (DAzon - egész, Nev - szöveg, Kor - egész, TAzon - egész)
tTanit tábla (TAzon – egész, DAzon – egész, Tantargy - szöveg)



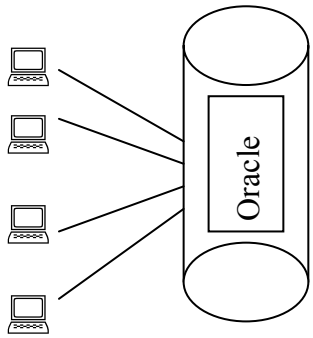
TAzon	Nev
1	Tóth Ottó
2	Varga Gyula
3	Kiss Béla
4	Nagy Aranka
…	…

DAzon	Nev	Kor	Ofo
10	Kovács Eszter	18	3
26	Kis Pisti	24	1
32	Oláh Anna	10	3
41	Herceg József	12	4

TAzon	DAzon	Tantargy
2	26	matematika
2	26	fizika
1	32	irodalom
4	10	angol
…	…	…

Érdemes valamilyen terminológiát bevezetni az objektumok elnevezését illetően. Mi a következőben egyezünk meg a félév erejéig: tTablanev (kis t és utána a név nagykezdőbetűvel ékezetek nélkül), azonosítók XAzon (a tábla nagybetűs kezdőbetűje és utána Azon)

Technológia

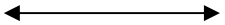


Tipikus a Kliens – Server kapcsolat. A kliensen dolgozik a felhasználó vagy közvetlen adatbázis-eléréssel (pl. SQL parancsokat ad ki), vagy egy olyan programmal, ahol a program tárolja az adatait adatbázisban (pl. a program SQL utasításokat tartalmaz). A szerver gépen található az adatbázis szerver szoftver.

A hálózati kapcsolat általában TCP/IP-re épül, de lehet más is (pl. MS SQL Server-nél tipikusan NamedPipes). A kliens-re telepíteni kell olyan szoftverkörnyezetet, ami ismeri az adott adatbázis-kezelő saját kommunikációs protokollját. Sajnos ez nem szabványos, így ahány fajta adatbázis-kezelőt használunk, annyi fajta klienst fel kell telepítenünk a gépünkre. A kommunikációs protokollok egymásra épülése pl. Oracle és TCP/IP használata esetén a következő: (OCI – Oracle Client Interface)

Felhasználó (SQL)	Program
OCI	
TCP	
IP (Internet Protocol)	
Fizikai hálózat	

Oracle adatbázis szerver
Oracle szerver hálózati protokoll
TCP
IP (Internet Protocol)
Fizikai hálózat

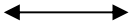


(A fenti ábra nem teljes, mert csak öt, számunkra fontos szintjét tűntettük fel a szabványos 7 szintű ISO/OSI protokoll felépítésnek.) Az IP a kis adatcsomagok továbbításáról gondoskodik, a TCP pedig a csomagok összeállítását végzi.

Javítja a helyzetet, hogy léteznek olyan szabványos hálózati protokollok, amelyek elfedik a felhasználói program elől, hogy valójában milyen adatbázis-kezelőt és protokollt használunk. (Ezzel elveszítjük az egyes adatbázis kezelők plusz funkcióit.) Microsoft-os környezetben ilyen az ODBC, vagy JAVA-ban a JDBC.
A fenti ábra ilyenkor kiegészül egy újabb szinttel:

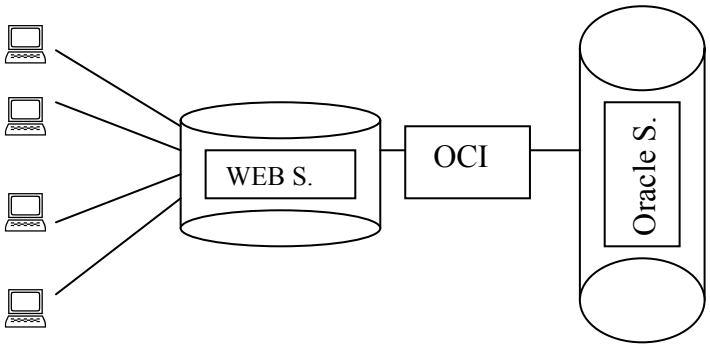
Felhasználó (SQL)	Program
ODBC	
OCI	
TCP	
IP	
Fizikai hálózat	

Oracle adatbázis szerver
ODBC
Oracle szerver hálózati protokoll
TCP
IP
Fizikai hálózat



Egy adatbázis szerveren több logikai adatbázis példány is lehet, mindegyiknek van egy-egy egyedi neve. Az adatbázis példányok sémákat tartalmaznak (szintén névvel azonosítva), amik elkülönült környezetet biztosítanak különböző alkalmazások számára, hogy az adatbázis objektumaikat (pl. tábláikat) tárolják.

A gyakorlatokon egy WEB-es architektúrában dolgozunk, ahol nem közvetlenül a kliens számítógépek csatlakoznak az adatbázis szerverhez, hanem WEB-es formokat töltünk ki, így csak a háttérben lévő WEB szerver csatlakozik az Oracle-höz.



Néhány szó az SQL-ről

Az SQL a „Structured Query Language” – „Strukturált Lekérdező Nyelv” rövidítése. Azokat a lekérdező nyelveket, amiknek a kifejező ereje ekvivalens a relációs algebrával, teljes nyelvnek nevezzük. Azokat, amikben ezen felül még olyan kérdések is feltehetők, amik a relációs algebrában nem, több mint teljes nyelvnek hívjuk. Az SQL a lekérdezéseket tekintve több mint teljes nyelv. Ezen felül nem csak lekérdezésekre, hanem adatmódosításokra, sémadefiníciókra, jogosultságokra vonatkozó parancsokat is megfogalmazhatunk SQL-ben.

SQL nyelv részei:

- DDL (Data Definition Language – Adat Definiáló Nyelv) ⇨ relációs séma
  séma kezelő utasítások: adatbázisok, táblák létrehozása, módosítása és törlése
  CREATE, ALTER, DROP
- DML (Data Manipulation Language – Adat Manipuláló nyelv)
  adat kezelő utasítások: adatok rögzítése, módosítása, törlése és lekérdezése
  INSERT, UPDATE, DELETE, SELECT
- DCL (Data Control Language)
  GRANT, REVOKE, ABORT, COMMIT, …, LOAD,…

Az SQL általában nem különbözteti meg a kis- és nagybetűket (szerver beállítástól függ). (Oracle-ben csak az idézőjelek közé tett kifejezések kivételek ez alól.)

A jegyzetben szereplő jelölésekről

- < > jelek közé írt értékeket mindig az aktuális értékekkel kell helyettesíteni
- [ ] jelek közé írt részek opcionálisak
- {..|...} a parancsba a felsoroltak közül az egyik lehetőséget kell írni

Táblák létrehozása

CREATE TABLE <tábla név>
(<oszlop név$_1$> <oszlop típus$_1$>,
<oszlop név$_2$> <oszlop típus$_2$>,
… ,
<oszlop név$_n$> <oszlop típus$_n$>);

Típusok:

- integer – egész
- number - szám
- char(hossz) – fix hosszú szöveg
- varchar(maxhossz) – maximum maxhossz hosszú szöveg
- date – dátum (TO_DATE('1980.06.25','YYYY.MM.DD'))

Példa: Hozzuk létre a fenti példában szereplő tTanar táblát!

```
CREATE TABLE tTanar
(TAzon INTEGER,
 Nev VARCHAR(40));
```

Megszorítások csoportosítása

Tábla szintű megszorítások:

- elsődleges kulcs
- idegen kulcs
- egyediség
- logikai feltétel

Mezőkre vonatkozó megszorítások:

- alapértelmezett érték
- kötelezőség (nem null érték)

Megszorítások elhelyezkedése a CREATE parancsban

CREATE TABLE <tábla név>
(<mező név$_1$> <típus név$_1$> <mező szintű megszorítás$_{11}$> … <mező szintű megszorítás$_{1k}$>,
…,
<mező név$_n$> <típus név$_n$> <mező szintű megszorítás$_{n1}$> …<mező szintű megszorítás$_{nm}$>,
<tábla színtű megszorítás$_1$>,
…,
<tábla színtű megszorítás$_h$>)

A tábla szintű megszorítások általános szintaxisa:

CONSTRAINT <megszorítás neve> <megszorítás definíciója>

Megszorítások

1. elsődleges kulcs

- egyedi azonosítója a sornak (egész rekordot meghatározza)
- esetek 95%-ban van elsődleges kulcs
- példánkban a tTanar táblában a TAzon, a tDiak táblában pedig a DAzon az elsődleges kulcs
- szokásos elnevezés: pk_tablanev
- CONSTRAINT <megszorítás név>
  PRIMARY KEY (<mezőnév$_1$>,…, <mezőnév$_k$>)

2. idegen kulcs

- olyan mező (vagy mezők) a táblában (hivatkozó tábla), amely egy másik táblában (hivatkozott tábla) elsődleges kulcsként előforduló értékeket vehet fel (!típus kompatibilitás)
  Emiatt ezt a megszorítástípust szokás tartalmazási függőségnek is nevezni, pl.:
  tDiak(Ofo) $\subseteq$ tTanar(tAzon)
  (Ebben a példában a tDiak(Ofo) jelölés a tDiak tábla Ofo oszlopában előforduló összes érték halmazát jelenti.)
- hivatkozott táblából nem törölhető olyan rekord, amely elsődleges kulcsára hivatkozik a hivatkozó tábla
- jelölés: nyíl (hivatkozó táblától a hivatkozott táblához) vagy „tyúk láb” (tyúk láb a hivatkozó táblánál)
- példánkban a tDiak tábla Ofo mezője hivatkozik a tTanar tábla TAzon mezőjére
- Az idegen kulcsot két perspektívából is nézhetjük. Eddig a megszorítás megközelítésre koncentráltunk, ami azt sugallja, hogy valamit nem lehet csinálni: pl. olyan Ofo

értéket a tDiák táblába írni, amilyen azonosítójú tanár nem létezik a tTanar táblában. A másik megközelítés a kapcsolat: a tanárok és a diákok között egy „osztályfőnök” elnevezésű logikai kapcsolatot hozunk létre. Az idegen kulcs mindig egy-a-sokhoz kapcsolatot valósít meg (egy diáknak egy osztályfőnöke van, de egy tanár sok diáknak lehet osztályfőnöke).

- szokásos elnevezés: fk_hivatkozotablanev_hivatkozotttablanev (fk_tDiak_tTanar)
- CONSTRAINT <megszorítás név>
  FOREIGN KEY (<mező név$_1$>,…, <mező név$_k$>)
  REFERENCES <hivatkozott tábla név> (<új mező név$_1$>,…, <új mező név$_k$>)

3. egyediség

- akkor használjuk, ha több kulcs is van egy táblában. Mivel olyan nincs, hogy „másodlagos” vagy „harmadlagos” kulcs, minden, az elsődleges kulcs utáni kulcsot az egyediség megszorítással adunk meg.
- logikailag ugyanazt jelenti, mint az elsődleges kulcs: a táblában az adott mezőben minden értéknek különbözőnek kell lennie
- annyi eltérés van az elsődleges kulcshoz képest, hogy tartalmazhat NULL-os mezőt is. Ilyenkor előfordulhat a NULL érték is, de csak egyszer.
- amennyiben a megszorítás több mezőre vonatkozik, akkor a mezőkben szereplő értékek együttesének kell különbözőnek lenni
- szokásos elnevezés: uq_tablanev
- CONSTRAINT <megszorítás név> UNIQUE (<mező név$_1$>,…, <mező név$_k$>)

4. logikai feltétel

- feltételeket adhatunk meg a tábla mezőire vonatkozóan
- szokásos elnevezés: ck_tablanev
- CONSTRAINT <megszorítás név> CHECK (<logikai feltétel>)
- különböző logikai operátorokat használhatunk, pl. IN, OR, AND, összehasonlító relációk, egyéb műveletek
- példa logikai feltételre:
  - <mező név$_1$> IN (<halmaz$_1$>) AND … AND <mező név$_k$> IN (<halmaz$_k$>)
  - <mező név$_m$> >= <mező név$_k$> AND <mező név$_t$>* <mező név$_s$> <50

5. alapértelmezett érték

- ha egy rekordnál az adott mező értékét nem adjuk meg, akkor az alapértelmezett értéket veszi fel értékként
- DEFAULT <érték>

6. kötelezőség (nem null érték)

- nem engedi meg olyan sorok előfordulását, amelyben az adott attribútum érték nincs megadva
- elsődleges kulcs mezői nem NULL értékűek
- NOT NULL
- ha van alapértelmezett érték és kötelezőség is, akkor a definícióban a helyes sorrend: DEFAULT <érték> NOT NULL

Feladat:

Hozzuk létre a példánkban szereplő adatbázist, a következő megszorításokkal:

- elsődleges kulcsok: tTanar: TAzon, tDiak: DAzon
- idegen kulcs: tDiak: Ofo (tTanar: TAzon)

- egyediség: tDiak: Nev
- logikai feltétel: tDiak: Kor csak egy és 100 közötti érték lehet
- alapértelmezett érték: tDiak: Ofo alapértelmezett értéke 1
- kötelezőség: elsődleges kulcsok és tDiak: Ofo

```
CREATE TABLE tTanar
(TAzon INTEGER NOT NULL,
 Nev VARCHAR(50),
 CONSTRAINT pk_tTanar PRIMARY KEY (TAzon))

CREATE TABLE tDiak
(DAzon INTEGER NOT NULL,
 Nev VARCHAR(50),
 Kor INTEGER,
 Ofo INTEGER DEFAULT 1 NOT NULL,
 CONSTRAINT pk_tDiak PRIMARY KEY (DAzon),
 CONSTRAINT fk_tDiak_tTanar FOREIGN KEY (Ofo)
            REFERENCES tTanar (TAzon),
 CONSTRAINT ck_tDiak CHECK (Kor BETWEEN 1 AND 100),
 CONSTRAINT uq_tDiak UNIQUE (Nev))
```

Adatok rögzítése

Most ismertetjük annak a három DML műveletnek a szintaxisát, amelyekkel létező táblákba:

- új sorokat tudunk beszúrni (INSERT INTO)
- létező sorok adatait tudjuk módosítani (UPDATE)
- sorokat tudunk törölni (DELETE)

Itt nagyon fontos ismerni az adott adatbázis-kezelő rendszer tranzakciókezelését. Az Oracle-ben ahogy kiadunk egy INSERT INTO vagy UPDATE vagy DELETE parancsot, automatikusan elindul egy tranzakció. Ez addig tart, amíg a COMMIT paranccsal jóvá nem hagytuk, vagy a ROLLBACK paranccsal vissza nem vontuk. Jó tudni, hogy vannak olyan kliens oldali környezetek, amik egy-egy DML parancs után automatikusan kommitálnak. Ehhez viszont nem jó hozzászokni, mert egyrészt a parancs kiadása után már nem lesz lehetőségünk visszavonni azt, másrészt pedig nem építhetünk több parancsból álló tranzakciókat. Ennek megfelelően abban a WEB-es környezetben, amit mi használunk, nincs automatikus tranzakció kezelés, tehát az alábbi parancsokat **MUSZÁJ KOMMITÁLNI VAGY VISSZAVONNI!!!**

Fontos tudnivaló továbbá, hogy egy tranzakció csak a fent felsorolt három fajta parancsból állhat. Egy CREATE TABLE pl. evidens módon nem tranzakcionális parancs, azaz visszavonhatatlanul azonnal végrehajtódik, amikor futtatjuk, és sem a COMMIT sem a ROLLBACK nincs rá hatással.

INSERT INTO <tábla név>
[(<mező név$_1$>, …, <mező név$_n$>)] VALUES (<érték$_1$>, …,<érték$_n$>);

Feladat: Töltsük fel adatokkal a tTanar táblát!
Megoldás (3 parancsból álló tranzakciót készítünk, és egyszerre kommitáljuk):

```
INSERT INTO tTanar VALUES (1,'Tarcsi Ádám');
INSERT INTO tTanar VALUES (2,'Papp Szabolcs');
INSERT INTO tTanar VALUES (3,'Varga Zsuzsanna');
COMMIT;
```

Feladat: Rögzítsük 1-es azonosítóval Nagy Júliát, aki 14 éves és az 2-es azonosítójú tanár tanítja!

```
INSERT INTO tDiak VALUES (1,'Nagy Júlia', 14, 2);
COMMIT;
```

Feladat: Rögzítsük 2-es azonosítóval Kiss Évát!

```
INSERT INTO tDiak (DAzon, Nev) VALUES (2,'Kiss Éva');
COMMIT;
```

Feladat: Töltsük fel további adatokkal a tDiak táblát!
Megoldás:

```
INSERT INTO tDiak VALUES (3,'Tóth Ottó', 14, 2);
INSERT INTO tDiak VALUES (4,'Farkas Péter', 13, 3);
INSERT INTO tDiak VALUES (5,'Kovács Elemér', 17, 1);
INSERT INTO tDiak VALUES (6,'Kocsis Janka', 15, 1);
COMMIT;
```

Adatok módosítása

UPDATE <tábal név> SET <mező név$_1$>=<érték$_1$>, …, <mező név$_k$>=<érték$_k$>

[WHERE <feltétel>];
Ha nincs WHERE, akkor a tábla összes rekordjára vonatkozik a módosítás!

Feladat: Módosítsuk az 1-es azonosítójú diák nevét 'Nagy Júlia Anna'-ra!

```
UPDATE tDiak SET Nev='Nagy Júlia Anna' WHERE DAzon=1;
COMMIT;
```

Feladat: Nagy Júlia Anna egy évvel idősebb lett, módosítsuk az adatait ennek megfelelően!

```
UPDATE tDiak SET Kor=Kor+1 WHERE Nev='Nagy Júlia Anna';
COMMIT;
```

Feladat: Minden diáknak a 2-es aztonosítójú tanár legyen az osztályfőnöke!

```
UPDATE tDiak SET Ofo=2;
COMMIT;
```

Adatok törlése

DELETE FROM <tábla név> [WHERE <feltétel>];
Ha nincs WHERE, akkor a tábla összes rekordját töröljük!

Példa: Töröljük a 2-es azonosítójú diákot!

```
DELETE FROM tDiak WHERE DAzon=2;
COMMIT;
```

Példa: Töröljük azokat a diákokat, akiknek a 3-as azonosítójú tanár az osztályfőnökük!

```
DELETE FROM tDiak WHERE Ofo=3;
COMMIT;
```

Táblák módosítása

ALTER TABLE <tábla név> …;

- új oszlop hozzáadása a táblához (a tábla „végére” kerülnek az új oszlopok)
  ALTER TABLE <tábla név> ADD <oszlop név> <oszlop típus>;
  Példa: Adjunk egy Kor nevű egész típusú oszlopot a tTanar táblához!
  ```
  ALTER TABLE tTanar ADD Kor INTEGER;
  ```

- egy oszlop típusának módosítása
  ALTER TABLE <tábla név> MODIFY <oszlop név> <új oszlop típus>;
  (Ha már vannak az adatbázisban adatok, akkor probléma merülhet fel a művelet végrehajtása közben!)
  Feladat: Legyen a tDiak tábla Nev mezőjének típusát 50 hosszú szöveg!
  ```
  ALTER TABLE tDiak MODIFY Nev VARCHAR(50);
  ```

- egy oszlop törlése

  ALTER TABLE <tábla név> DROP COLUMN <oszlop név>;
  (Ha az oszlop például elsődleges kulcs, akkor a művelet hibához vezet.)
  Feladat: Töröljük a tTanar táblából a Kor oszlopot!
  ```
  ALTER TABLE tTanar DROP COLUMN Kor
  ```

Tábla szintű megszorítások utólagos kezelése

- ALTER TABLE <tábla név> ADD CONSTRAINT <megszorítás név> …;
  Feladat: Adjunk egy olyan megszorítást a tTanar táblához, aminek következtében nem tárolhatunk két ugyanolyan nevű tanárt!
  ```
  ALTER TABLE tTanar ADD CONSTRAINT uq_tTanar UNIQUE (Nev);
  ```
  Ellenőrzés: `INSERT INTO tTanar VALUES (1,'Tarcsi Ádám');`

- ALTER TABLE <tábla név> DROP CONSTRAINT <megszorítás név>
  Feladat: Dobjuk el az előbbi megszorítást!
  ```
  ALTER TABLE tTanar DROP CONSTRAINT uq_tTanar;
  ```
  Ellenőrzés: `INSERT INTO tTanar VALUES (1,'Tarcsi Ádám');`

Mező szintű megszorítások kezelése

MODIFY segítségével (! NOT NULL megszüntetése: NULL)
Példa: Legyen a tDiak tábla Kor mezőjének 18 az alapértelmezett értéke!
```
ALTER TABLE tDiak MODIFY Kor INTEGER DEFAULT 18;
```
Ellenőrzés: `INSERT INTO tDiak (DAzon, Nev) VALUES (7,'Pap Éva');`

Táblák eldobása

DROP TABLE <táblanév> [CASCADE];
Feladat: Dobjuk el a tDiak táblát!
```
DROP TABLE tDiak;
```

Feladat: Dobjuk el a tTanar táblát!
Megoldás: `DROP TABLE tDiak;`

3. óra

Adatok lekérdezése

- input: egy vagy több tábla
  output: a megfogalmazott feltételeknek eleget tevő rekordok megfelelő attribútumait tartalmazó tábla

- utasítás általános formája

  SELECT [ALL/DISTINCT] {*/<mező név$_1$>, …, <mező név$_k$>}
  FROM <tábla név$_1$> [<hivatkozási név$_1$>], …, <tábla név$_h$> [<hivatkozási név$_h$>]
  [WHERE <feltétel$_1$>
  GROUP BY <mező név$_1$>, …, <mező név$_m$>
  HAVING <feltétel$_2$>
  ORDER BY <mező név$_1$> [ASC/DESC], …, <mező név$_j$> [ASC/DESC]]

- műveletek sorrendje

  1. FROM: melyik táblákból? (direkt szorzat képzés)
  2. WHERE: melyik rekordok? (nem kívánt sorok elhagyása)
  3. GROUP BY: mi szerint csoportosítva? (csoportosítás)
  4. HAVING: melyik csoportok? (nem kívánt csoportok elhagyása)
  5. ORDER BY: mi szerint rendezve? (rendezés)
  6. SELECT: melyik oszlopok? (nem kívánt oszlopok elhagyása)

- SELECT * FROM <táblanév>;
  Eredménye: ugyanolyan tábla, mint a bemeneti

- SELECT * FROM <táblanév$_1$>, <táblanév$_2$>
  Eredménye: a két tábla direktszorzata (oszlopainak száma a két bemeneti tábla oszlopszámainak összege)

  Példa: `SELECT * FROM t1,t2`

t_1:	a
	1
	2

t_2:	b
	x
	y
	z

  Eredmény (direktszorzat):

a	b
1	x
1	y
1	z
2	x
2	y
2	z

- Ha nincs minden mezőre szükség az eredmény táblában, akkor a * helyett fel kell sorolni a szükséges mezőket.
- Az oszlopnevek mellett kifejezéseket is írhatunk a SELECT rész után, pl:
  `SELECT a*b/3, log(c)-1 FROM t;`
- Ha több táblából kérdezünk le, akkor azt is meg kell adni, hogy a kívánt mező mely táblából való (minősítés): <táblanév>.<oszlopnév>

Példa: `SELECT t1.a FROM t1,t2`

Eredmény:

a
1
1
1
2
2
2

- A táblanév rövidíthető, ha neve után megadunk egy hivatkozási (alias) nevet.
  Feladat: `SELECT * FROM tDiak d, tTanar t …`
- **Ha két olyan táblát direktszorzunk össze, amik között idegen kulcs van, a rendszer alapból nem veszi figyelembe ezt a kapcsolatot (mint pl. Access-ben)! Ilyenkor un. join feltétel megadása szükséges.**
  Ha pl. azt szeretnénk, hogy minden diák mellett csak az osztályfőnöke adatai látsszanak, akkor a következő lekérdezést kell kiadnunk:
  ```
  SELECT * FROM tDiak d, tTanar t
  WHERE d.Ofo=t.Tazon;
  ```
  Ez biztosítja, hogy nem minden diák minden tanárral kerül párban kiíratásra, hanem csak az osztályfőnökével.

- Ha az eredmény táblában más nevet szeretnénk adni az oszlopoknak, akkor ezt úgy tehetjük meg, hogy az oszlop név után egy „AS” szócskát írunk és a kívánt oszlop nevet.
  Példa: SELECT tDiak.Nev AS Diaknev, … FROM tDiak …

- A SELECT után megadhatunk konstans értékeket is, ekkor az eredmény táblában lesz egy olyan oszlop, amelynek neve az adott konstans és minden rekord adott mezője az adott konstanst veszi fel értékként.

Példa: `SELECT a,1,'hello' FROM t1`

Eredmény:

a	1	'hello'
1	1	hello
2	1	hello

- Ha egy oszlopból csak a különböző értékeket szeretnénk lekérdezni, akkor a DISTINCT kulcsszót kell használni.
  SELECT DISTINCT <oszlop név> FROM tDiak

  Feladat: Határozzuk meg, hogy hány éves diákok vannak az adatbázisban!
  Megoldás: `SELECT DISTINCT Kor FROM tDiak`

- Konjunktív feltételek:
  - egyenlőség (egyenlőtlenség) mezők és/vagy konstansok között
  - ÉS operátor
  - JOIN (összekapcsolás: hivatkozó tábla idegen kulcsa = hivatkozott tábla elsődleges kulcsa)

  Feladat: Adjuk meg azon diákok nevét, akiknek az 1-es azonosítójú tanár az ofőjük!
  `SELECT Nev FROM tDiak WHERE Ofo=1`

  Feladat: Adjuk meg azon diákok nevét, akiknek nincs megadva az ofőjük!

```
SELECT Nev FROM tDiak WHERE Ofo IS NULL
```

Feladat: Készítsünk egy olyan táblázatot, melynek első oszlopában a diák neve szerepel, második oszlopában pedig az adott diák osztályfőnökének neve!

```
SELECT d.Nev, t.Nev
FROM tDiak d, tTanar t
WHERE d.Ofo=t.Tazon

SELECT d.Nev || '-t ' || t.Nev || ' tanítja'
FROM tDiak d, tTanar t
WHERE d.Ofo=t.Tazon
```

Feladat: Adjuk meg azon tanulók listáját, akiknek Ottó az osztályfőnökük és elmúltak 18 évesek! Ehhez tudnunk kell, hogy a LIKE operátort használhatjuk sztringek joker karakterrel történő összehasonlítására. Pl. a NEV LIKE '%A%' feltétel azokra a nevekre lesz igaz, amikben szerepel legalább egy A betű. Fontos, hogy a legtöbb adatbázis-kezelő rendszer csak a % jelet fogadja el jokerként, más helyettesítő karakter nincs (? vagy * nem jó!).

```
SELECT d.Nev, d.Kor
FROM tDiak d, tTanar t
WHERE
  d.Kor >18 and t.Nev LIKE '% Otto' AND d.Ofo=t.Tazon
```

- Aggregátum függvények:
  - Számosság: SELECT COUNT(*) FROM <tábla név>
    Ha nincs GROUP BY, akkor az eredmény egy egyoszlopos, egyetlen sort tartalmazó tábla, amiben a sorok számát kapjuk meg.
    Példa: Adjuk meg, hogy hány diák adatát tároljuk az adatbázisban.

    ```
    SELECT COUNT(*) FROM tDiak
    SELECT COUNT(1) FROM tDiak
    ```

    Eredmény:

COUNT(*)
<diákok száma>

  - Összeg: SELECT SUM(<oszlop név>) FROM <tábla név>
    Példa: Adjuk meg, hogy mennyi a diákok átlagéletkora!

    ```
    SELECT SUM(Kor)/COUNT(Kor) FROM tDiak
    ```

  - Átlag: SELECT AVG(<oszlop név>) FROM <tábla név>
    Példa: Adjuk meg, hogy mennyi a diákok átlagéletkora!

    ```
    SELECT AVG(Kor) FROM tDiak
    ```

  - Minimum, maximum: SELECT {MIN/MAX} (<oszlop név>) FROM <tábla név>
    Példa: Adjuk meg, hogy hány éves a legidősebb diák!

    ```
    SELECT MAX(Kor) FROM tDiak
    ```

  - Megjegyzések
    - Több aggregátum függvényt is meg adhatunk a SELECT után, vesszővel elválasztva!

- Ha nincs GROUP BY, akkor aggregátum függvény után oszlopnév nem megengedett!

- Az ORDER BY utasítás segítségével meghatározhatjuk, hogy az eredmény táblában milyen sorrendben jelenjenek meg a rekordok. (Ha nem használjuk ezt az utasítást, akkor a program tetszőleges sorrendben adja meg a rekordokat.)

  Feladat: Rendezzük a tDiak tábla rekordjait név szerint növekvő és azon belül kor szerint csökkenő sorrendbe!
  Megoldás:

```
SELECT *
FROM tDiak
ORDER BY Nev [ASC], Kor DESC
```

4. óra

GROUP BY

A GROUP BY utasítással a táblák sorait egy-vagy több oszlop szerint csoportosíthatjuk, megszámolhatjuk, hogy hány sor van egy csoportban, vagy kiválaszthatjuk a csoport egy kívánt tagját.

Feladat: Adjuk meg, hogy az egyes tanárok hány diáknak osztályfőnökei! (Azonosítóval)
Megoldás:

```
SELECT Ofo, COUNT(DAzon) AS Db
FROM tDiak
GROUP BY Ofo;
```

Eredmény:

Ofo	Db
1	3
3	4
4	1

Ezzel az utasítással nem kerülnek be az eredmény táblába azok a tanárok, akik egy diáknak sem osztályfőnökei!

Az előző alkalommal azt tanultuk, hogy ha aggregátum függvényt használunk a SELECT után, akkor egyszerű mezőnevet már nem írhatunk a SELECT utáni listába. Most azért írhattuk az Ofo mezőt mégis oda, mert eszerint csoportosítottunk.

Feladat: Adjuk meg, hogy az egyes tanárok hány diáknak osztályfőnökei! (Névvel)
Megoldás:

```
SELECT t.Nev, COUNT (d.DAzon) AS Db
FROM tDiak d, tTanar t
WHERE d.Ofo=t.TAzon
GROUP BY t.Nev;
```

Rossz eredményt kapunk abban az esetben, ha van két azonos nevű tanár!

Helyes megoldás:

```
SELECT t.Nev, COUNT (d.DAzon) AS Db
FROM tDiak d, tTanar t
WHERE d.Ofo=t.TAzon
GROUP BY t.Nev, t.TAzon;
```

Feladat: Osztályfőnökönként adjuk meg az egyes tanárok diákjainak átlagéletkorát!
Megoldás:

```
SELECT t.Nev, AVG(d.Kor) AS Atlag
FROM tDiak d, tTanar t
WHERE d.Ofo=t.TAzon
GROUP BY t.Nev,t.TAzon;
```

Megjegyzés: Csak akkor írhatunk egy oszlopnevet a SELECT utáni felsorolásba, ha szerepel a GROUP BY utáni is, az adott oszlop neve!

HAVING

Csoportokra vonatkozó feltételt fogalmazhatunk meg a segítségével, ezért csak a GROUP BY utasítással együtt használjuk.

Feladat: Adjuk meg azon tanárok nevét, akik legalább 3 diáknak osztályfőnökei!
Megoldás:

```
SELECT t.Nev
FROM tDiak d, tTanar t
WHERE d.Ofo=t.TAzon
GROUP BY t.Nev, t.TAzon
HAVING COUNT(d.DAzon)>=3
```

Feladat: Adjuk meg azon osztályfőnökök nevét és tanítványainak átlagéletkorát, akik legalább 3 diáknak osztályfőnökei!
Megoldás:

```
SELECT t.Nev, AVG(d.Kor) AS Atlag
FROM tDiak d, tTanar t
WHERE d.Ofo=t.TAzon
GROUP BY t.Nev,t.TAzon
HAVING COUNT(d.DAzon)>=3
```

Al SELECT

Az eddig tanult lekérdezések korlátja, hogy csak olyan kérdéseket tehetünk fel, amikre a válasz megadható a kiindulási tábláink direktszorzatából képzett sorok szűrésével és/vagy csoportosításával. Nagyon egyszerűen fel tudunk tenni olyan kérdést, ami ezzel az eszközrendszerrel nem megválaszolható.

Ilyen például: „Melyik tanár nem osztályfőnök?”

Mivel a két tábla tDiak.Ofo = tTanar.TAzon feltétel szerinti JOIN-jában eleve nem szerepelnek azok a tanárok, akik nem osztályfőnökök, így esélytelen, hogy konjunktív lekérdezéssel ezt a kérdést megválaszoljuk.

Ezzel szemben azt a kérdést, hogy „Melyik tanár osztályfőnök?”, nagyon egyszerűen meg tudjuk válaszolni:

```
SELECT DISTINCT t.Nev
FROM tDiak d, tTanar t
WHERE d.Ofo=t.TAzon;
```

A különbség csak a „nem” szócska. A negálás egy központi probléma a relációs adatbázis elméletben. A gondot az okozza, hogy véges sorokat tartalmazó adattáblák esetén az olyan jellegű kérdések, hogy mi van benne egy táblában, mindig véges eredményt adnak, viszont az olyanok eredménye, hogy mi nincs benne egy táblában, végtelen is lehet (tekintve hogy a típus-értékhalmazok elméletileg végtelenek).

A megoldás a fenti problémára SQL-ben az un. al-SELECT. Itt az történik, hogy egy SELECT által visszaadott eredményhalmazt (sorokat) valamilyen halmazoperátorral felhasználjuk egy másik SQL utasításon belül:

Példa: Melyik tanár nem osztályfőnök?
Megoldás:

```
SELECT *
FROM tTanar
WHERE TAzon NOT IN (SELECT Ofo FROM tDiak)
```

Az alábbiakban szerepel két példa arra, hogy konjunktív lekérdezésekkel megoldható feladatokra is alkalmazhatjuk az al-SELECT-eket. Megjegyezzük, hogy ez a megoldás kerülendő, mert bizonyos adatbázis-kezelő rendszerek a direkt szorzatos megoldást hatékonyabban kezelik.

Feladat: Melyik tanárok osztályfőnökök?
Megoldás:

```
SELECT *
FROM tTanar
WHERE TAzon IN (SELECT Ofo FROM tDiak)
```

Feladat: Adjuk meg azon osztályfőnököket, akiknek van náluk idősebb diákjuk!
Megoldás:

```
SELECT Nev
FROM tTanar t
WHERE TAzon IN
     (SELECT Ofo FROM tDiak d WHERE d.Kor>t.Kor)
```

INSERT INTO <táblanév> (<mezőnév$_1$>, …, <mezőnév$_n$>) (<al-select>)

Az INSERT INTO parancs ezen verziójával egyszerre több sort is beszúrhatunk. Olyankor alkalmazzuk, amikor nem egy konstans sort akarunk hozzáfűzni egy táblához, hanem a beszúrandó adatok egy lekérdezés eredményeképpen kaphatók meg.
Nyilván alapvető elvárás, hogy az al-select annyi oszlopot szelektáljon, ahány mezőnevet a zárójelben előtte felsoroltunk.

Példa: A tOsztaly tábla rekordjait szúrjuk be a tDiak táblába, úgy hogy az Ofo értéke minden rekord esetén 1 legyen!

tOsztály:	Dazon	Nev	Kor
	11	Kelemen János	19
	23	Huszár Béla	21
	34	Nagy István	12

```
INSERT INTO tDiak (DAzon, Nev, Kor, Ofo)
 (SELECT DAzon, Nev, Kor, 1 FROM tOsztaly)
```

UPDATE feltételeiben al-select-ekkel

Akkor használunk al-selectet módosítás esetén, ha a művelet függ egy másik táblától (azaz olyan WHERE feltételt adunk meg, ami nem csak arra a táblára vonatkozik, amit update-elünk).

UPDATE <tábla név> SET <mező név$_1$> = <érték$_1$>, …, <mező név$_k$> = <érték$_k$>
WHERE <feltétel al-select-ekkel>

Példa: Változtassuk a Ofo mező értékét 3-ra azon diákoknál, akiknek Zsakó László az osztályfőnökük!

```
UPDATE tDiak SET Ofo=3
WHERE Ofo IN
     (SELECT TAzon FROM tTanar WHERE Nev='Zsakó László')
```

DELETE FROM feltételeiben al-select-ekkel

Akkor használunk al-selectet törlés esetén, ha a művelet függ egy másik táblától (azaz olyan WHERE feltételt adunk meg, ami nem csak arra a táblára vonatkozik, amiből törlünk).

DELETE FROM <tábla név> WHERE <feltétel al-select-ekkel>

Példa: Töröljük azokat a tanárokat, akik egy diáknak sem osztályfőnökei!

```
DELETE FROM tTanar
WHERE TAzon NOT IN (SELECT Ofo FROM tDiak)
```

5. óra

GRANT, REVOKE

engedélyező utasítása

GRANT <jog> ON <táblanév> TO <felhasználó azonosítója>

lehetséges jogok: SELECT, INSERT, DELETE, UPDATE, ALL

Feladat: Adjunk SELECT jogot az Geza azonosítójú felhasználónak a tTanar táblához!
Megoldás: `GRANT SELECT ON tTanar TO Geza`
Próbáljuk ki:

- SELECT * FROM <felhasználó azonosítója>.tTanar
- beszúrás
- SELECT

visszavonó utasítás

REVOKE <jog> ON <táblanév> FROM <felhasználó azonosítója > [CASCADE]
(CASCADE: ha egy engedélyezési képességgel kiegészített jogosultságot vonunk vissza, akkor az illető jogosultsággal és engedélyezési képességgel más felhasználóknak átruházott jogosultságokat is mind megszüntetjük)

Feladat: Vonjuk vissza az előbb adott SELECT jogot!
Megoldás: `REVOKE SELECT ON tTanar FROM Geza`

Halmazműveletek

A lekérdezések eredmény sorainak halmazát tekintve több lekérdezés között értelmezhetjük a szokásos halmaz műveleteket. Természetesen ez csak akkor lehetséges, ha a lekérdezések kompatibilisek egymással, azaz ugyanannyi, ugyanolyan típusú oszlopot adnak vissza.

Szintaxis:
SELECT … <művelet> SELECT …;

A következő műveleteket ismerjük:

- UNION (unió halmazképzéssel, azaz az azonos sorokból csak egyet szerepeltet)
- UNION ALL (unió halmazképzés nélkül, az azonos sorok bent maradnak)
- INTERSECT (metszet)
- MINUS (halmaz különbség)

Példa: Írjuk ki az összes tanár nevét, és hogy hány diáknak osztályfőnöke! Azok a tanárok is jelenjenek meg, akik nem osztályfőnökök!

```
SELECT t.nev, COUNT(1)
  FROM tTanar t, tDiak d
  WHERE d.Ofo = t.TAzon
  GROUP BY t.nev, t.TAzon
UNION
SELECT nev, 0
  FROM tTanar
  WHERE TAzon NOT IN (SELECT Ofo FROM tDiak);
```

Sorszámozás

A táblákban gyakran használunk folytonosan növekvő egész számokat arra, hogy azonosítsuk a sorokat, és tipikusan ezt jelöljük meg elsődleges kulcsként. Minden adatbázis-kezelő rendszer ad lehetőséget az automatikus sorszámgenerálásra. Erre tipikus megoldás, hogy a tábla definíciójában a mező típusánál megmondjuk, hogy ez egy sorszámozandó mező, és a rendszer az új sorok beszúrásakor mindig automatikusan generál egy, az előzőnél eggyel nagyobb számot (pl. Microsoft SQL Server, MySQL).
Az Oracle ezt –nagyon szimpatikus módon– máshogy oldja meg. A tábláktól függetlenül létrehozhatunk speciális objektumokat (sequence-eket), amik generálják a számokat:
CREATE SEQUENCE <nev> [START WITH a] [INCREMENT BY b];
Ekkor a sorszámozás a-val kezdődik, és b-esével növekszik. Alapértelmezésben a=1 és b=1;

Pl. CREATE SEQUENCE seqProba;
A sequence-ből az Oracle dummy táblájára (u.n. DUAL) vonatkozó lekérdezéssel tudunk egy új értéket kérni: SELECT seqProba.nextval FROM DUAL;

Feladat: Készítsünk egy sequence-et a tDiak táblára, és szúrjunk be néhány diákot ennek segítségével!

Megoldás:

```
CREATE SEQUENCE seqDiak START WITH 50;

INSERT INTO tDiak (DAzon, Nev, Kor, Ofo)
(SELECT seqDiak.nextval, 'Kis Pista', 18, 2 FROM DUAL);

INSERT INTO tDiak (DAzon, Nev, Kor, Ofo)
(SELECT seqDiak.nextval, 'Nagy Jóska', 16, 1 FROM DUAL);

COMMIT;
```

Join-ok

Join-okról beszélünk, ha egy lekérdezésen belül több tábla adatait kombináljuk össze. Join feltételnek nevezzük a lekérdezések WHERE feltételében szereplő minden olyan részfeltételt, amiben legalább két tábla egy-egy oszlopa között fogalmazunk meg egy logikai kifejezést. A leggyakrabban használt join fajta az u.n. equijoin, ahol két tábla egy-egy oszlopa között az egyenlőség operátort használjuk. A „self join" kifejezést használjuk azokra a lekérdezésekre, melyekben egy táblát többször is szerepeltetünk a FROM részben.
Ahogy tanultuk, a join alapértelmezésben direktszorzat képzést jelent, mely eredményét tipikusan join feltételekkel szűkítjük. Erre a leggyakoribb példa az, amikor két tábla között egy-a-sokhoz (idegen kulcs) kapcsolat van, és a join feltétel az idegen kulcsban szereplő két oszlop között fogalmaz meg egyenlőséget.

Mivel a join feltételek a WHERE részben keverednek a többi feltétellel, speciális szintaxissal lehetőség van a join feltételt a FROM részben szerepeltetni. Így ez jól elkülönül a többi feltételtől.
Általános szintaxis:
SELECT … FROM t1 <join_típus> JOIN t2 ON t1.a = t2.b WHERE …;

Inner join

Az inner join (gyakran „egyszerű join”-nak, „beslő join”-nak vagy szimplán „join”-nak hívjuk) azt jelenti, hogy csakis azok a sorok jelennek meg a lekérdezés eredményében, melyek pontosan kielégítik a join feltételt.

Outer join

Az „külső join”-ok által visszaadott sorok halmaza bővebb, mint a belső join-ok eredménye. Visszaadják az összes olyan sort, melyek kielégítik a join feltételt, plusz még az egyik vagy mindkét táblából azokat is, melyekhez a másik táblában egyetlen olyan rekord sem található, amellyel a join feltétel kielégíthető lenne.
Pl. a tTanar és TDiak táblát a tDiak.Ofo = tTanar.TAzon feltétellel összekötve egy outer join azokat a tanárokat is tartalmazhatja, akik egyátalán nem is osztályfőnökök. Nyilván ezek a tanárok az inner join típusú összekötésben nem szerepelnének.
Ezeknél a pluszban megjelenő soroknál a másik tábla oszlopaiban csupa NULL értékeket fogunk látni. A fenti példában tehát azok a tanárok is megjelennek PONTOSAN EGYSZER, akik nem osztályfőnökök, és mivel nincs hozzájuk diák, a join eredményében a tDiak-ból származó összes oszlop értéke NULL lesz.
Az outer join-oknak három típusa van. Tegyük fel, hogy a t1 és a t2 táblából szelektálunk (ebben a sorrendben):

- left outer join: inner join által visszaadott sorok + egy-egy sor minden olyan t1-beli rekordhoz, melyhez nincs t2-ben egyetlen, a join feltétel által párosítható sor sem.
- right outer join: inner join által visszaadott sorok + egy-egy sor minden olyan t2-beli rekordhoz, melyhez nincs t1-ben egyetlen, a join feltétel által párosítható sor sem.
- full outer join: inner join által visszaadott sorok + egy-egy sor minden olyan t1-beli rekordhoz, melyhez nincs t2-ben egyetlen, a join feltétel által párosítható sor sem + egy-egy sor minden olyan t2-beli rekordhoz, melyhez nincs t1-ben egyetlen, a join feltétel által párosítható sor sem.

Fontos látni, hogy a LEFT és a RIGHT típusoknak semmi köze sincs ahhoz, hogy az idegen kulcs melyik irányba mutat. A t1 LEFT OUTER JOIN t2 logikailag pontosan ugyanaz, mint a t2 RIGHT OUTER JOIN t1, azzal a különbséggel, hogy első esetben a t1 oszlopai lesznek elöl, második esetben pedig a t2 oszlopai.

Tegyük fel, hogy a tTanar és TDiak táblákban a következő sorok vannak. A példa kedvéért most megengedjük, hogy az Ofo mező NULL is lehessen (ezt hívjuk nem kötelező kapcsolatnak a két tábla között):

tTanar

TAzon	Nev
1	Kis Benedek
2	Horváth Mihály

tDiak

DAzon	Nev	Ofo
1	Kovács Béla	1
2	Tóth Eszter	1
3	Nagy Mária	

Az alábbiakban megadjuk a két tábla mind a négy fajta join-olásának szintaxisát és prezentáljuk az eredményt is. Megjegyzendő, hogy az outer join-oknál bemutatott (+) jel nem szabványos, és csak az Oracle-ben működik.

Inner join

```
SELECT *
  FROM tDiak d, tTanar t
  WHERE d.Ofo = t.TAzon;
```

vagy

```
SELECT *
  FROM tDiak d INNER JOIN tTanar t ON d.Ofo = t.TAzon;
```

DAzon	Nev	Ofo	TAzon	Nev_1
1	Kovács Béla	1	1	Kis Benedek
2	Tóth Eszter	1	1	Kis Benedek

Left outer join

```
SELECT *
  FROM tDiak d, tTanar t
  WHERE d.Ofo = t.TAzon(+);
```

vagy

```
SELECT *
  FROM tDiak d LEFT OUTER JOIN tTanar t ON d.Ofo = t.TAzon;
```

DAzon	Nev	Ofo	TAzon	Nev_1
1	Kovács Béla	1	1	Kis Benedek
2	Tóth Eszter	1	1	Kis Benedek
3	Nagy Mária			

Right outer join

```
SELECT *
  FROM tDiak d, tTanar t
  WHERE d.Ofo(+) = t.TAzon;
```

vagy

```
SELECT *
  FROM tDiak d RIGHT OUTER JOIN tTanar t ON d.Ofo = t.TAzon;
```

DAzon	Nev	Ofo	TAzon	Nev_1
1	Kovács Béla	1	1	Kis Benedek
2	Tóth Eszter	1	1	Kis Benedek
			2	Horváth Mihály

Full outer join

```
SELECT *
  FROM tDiak d FULL OUTER JOIN tTanar t ON d.Ofo = t.TAzon;
```

DAzon	Nev	Ofo	TAzon	Nev_1
1	Kovács Béla	1	1	Kis Benedek
2	Tóth Eszter	1	1	Kis Benedek
3	Nagy Mária			
			2	Horváth Mihály

Hierarchikus (rekurzív) lekérdezések

Relációs táblákban nagyon egyszerűen tárolhatunk hierarchikus adatokat úgy, hogy készítünk egy oszlopot, mely idegen kulccsal ugyanennek a táblának az azonosítójára mutat. Pl:
tDolgozo

DAzon	Nev	Fonokazon
1	Nagy Főnök	
2	Osztály Vezető1	1
3	Osztály Vezető2	1
4	Kisfőnök	2
5	Dolgozó1	4
6	Dolgozó2	4
7	Dolgozó3	3

Itt a Fonokazon ugyanebben a táblában a DAzon mezőre mutat, és a dolgozó közvetlen főnökét reprezentálja.
Feltehetjük az alábbi rekurzív megoldást igénylő kérdést: Kik az Osztály Vezető1 alá rendelt emberek? Erre a következő lekérdezéssel tudunk válaszolni (Oracle-ben):

```
SELECT *
  FROM tDolgozo
  START WITH nev = 'Osztály Vezetõ1'
  CONNECT BY FonokAzon = PRIOR DAzon;
```

Itt a START WITH kulcsszó utáni feltétel definiálja a gyökér elemet, a CONNECT BY feltétel pedig a hierarchikus fa éleit úgy, hogy a PRIOR operátorral mondhatjuk meg, hogy ez a fában „feljebb” lévő attribútum. Mindkét feltétel lehet összetett logikai kifejezés is. Addicionálisan használhatjuk a level értéket, ami megmondja, hogy az adott elem milyen mélyen van a fában:

```
SELECT DAzon, Nev, Fonokazon, level
  FROM TDOLGOZO
  START WITH nev = 'Osztály Vezetõ1'
  CONNECT BY FonokAzon = PRIOR DAzon;
```

Indexelés

Az adatbázisok megtervezésénél a logikai adatmodell után elkészítik a fizikai adatbázis struktúrát (pl. a táblák CREATE script-jei a megfelelő megszorításokkal együtt). Ennél a pontnál viszont bizonyos, a logikai modellből nem következő tényezőkre is figyelemmel kell lenni. Ezek tipikusan olyanok, amelyek már érintik az adatbázis tárolási módját, fizikai szerkezetét. Ezek közül most egyet vizsgálunk meg, az indexeket.
Az indexelés a keresések hatékonyságát növeli. Ha nem használjuk, akkor a rendszer a tábla / táblák összes sorára szekvenciálisan végrehajtja a WHERE feltételt . Indexek használata esetén az adatbázis-kezelő adott oszlop / oszlopok szerint egy vagy több index struktúrát (pl. B-fa) is fenntart, melyet minden módosításkor aktualizál. Ha olyan oszlop szerint keresünk, ami indexelve van, akkor az index struktúra alapján fog keresni. Pl. ha egy 10 millió embert nyilvántartó táblában kiadjuk a következő lekérdezést:
SELECT * FROM Emberek WHERE Nev = ’Tóth István’;
akkor alap helyzetben legrosszabb esetben 10 millió lépésben, 10 millió szöveges összehasonlítással fogja a rendszer megtalálni Tóth István adatait. Míg ha a Nev mező szerint a táblát előzőleg indexeltük, akkor a keresés log(10 millió) = 24 lépésben lefut.

Az indexeket a következő paranccsal hozhatjuk létre:
CREATE [UNIQUE] INDEX <index_neve> ON <táblanév> (oszlop1, …, oszlopN);

Több oszlopból álló (u.n. összetett) indexet akkor érdemes létrehozni, ha sok olyan lekérdezésünk van, aminek a WHERE feltételében egyszerre szűrünk ezekre az oszlopokra. A UNIQUE kulcsszó használata esetén automatikusan az UNIQUE constraint is születik ezekre az oszlopokra.
Az adatbázis-kezelők az elsődleges kulcsokra mindig automatikusan definiálnak indexet (tehát elsődleges kulcsra mindig hatékonyan tudunk keresni).

Íme néhány iránymutatás, amit érdemes megfontolni akkor, amikor azt tervezzük, hogy milyen indexeket hozzunk létre az adatbázisunkhoz:

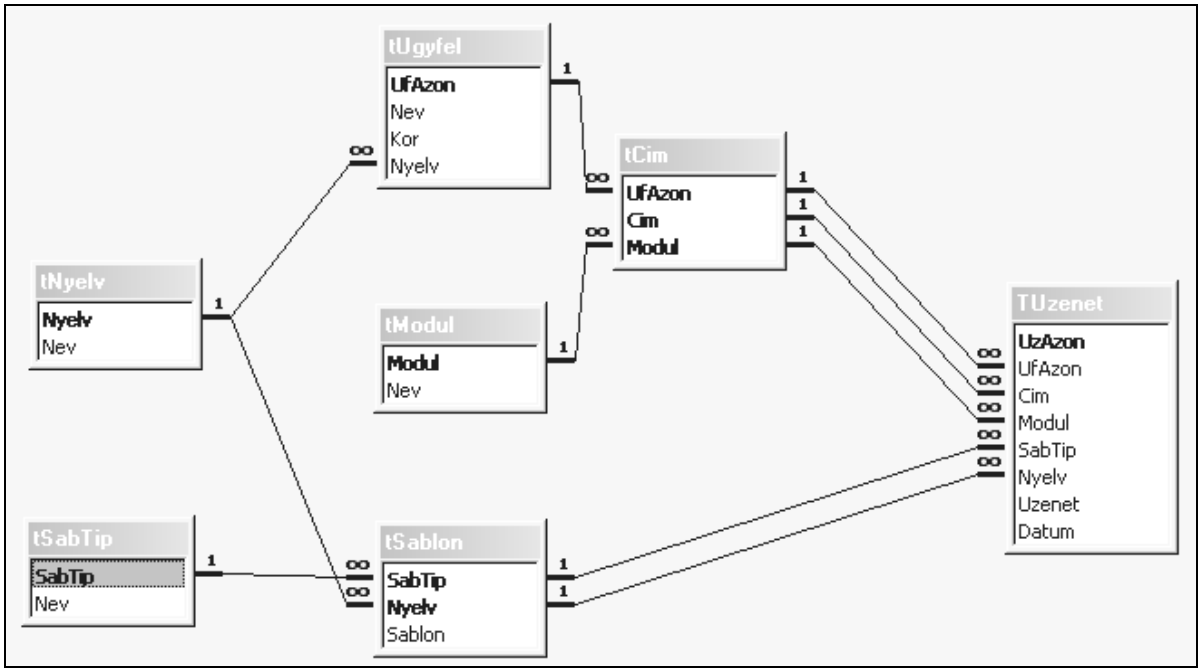
- Csak azokat a táblákat indexeljük, amikben nagy mennyiségű adatot tárolunk.
- Csak azokat a mezőket indexeljük, amelyekre gyakran hivatkozunk a WHERE feltételeinkből.
- Idegen kulcsok esetén a hivatkozó tábla hivatkozó oszlopát szinte mindig érdemes indexelni, ha a hivatkozott táblában aránylag sok adat van.
- Olyan mezőket ne indexeljünk, amik csak néhányféle adatot tartalmaznak (pl. emberek neme: ’F’/’N’, erre akkor sem szabad indexet tenni, ha nagyon sok emberünk van, és nagyon gyakran kérdezzük le pl. a férfiakat!).
- Gondoljuk meg, hogy indexeljük-e azokat a mezőket, amiket gyakran módosítunk. Az UPDATE, INSERT, DELETE parancsoknál ugyanis mindig újra kell építeni az index struktúrát, ami időt vesz igénybe. Mindig azt kell mérlegelni, hogy azt szeretnénk-e, hogy a lekérdezés legyen gyors (ekkor lehetőleg indexelünk), vagy a módosítások (ekkor lehetőleg nem indexelünk).
- Ha egy mezőre gyakran hivatkozunk, ám mindig speciális kifejezéssel, akkor az adatbázis-kezelő nem fogja tudni használni az indexet. Ilyenkor válasszuk inkább az úgynevezett funkció alapú indexelést. Pl. hiába van egy tábla a, b, c oszlopok szerint rendezve, a SELECT * FROM t WHERE a + b * (c – 1) < 100; lekérdezést csak index használat nélkül, szekvenciálisan végignézve az egész táblát fogja tudni végrehajtani az adatbázis-kezelő. Lehetőség van viszont a következő funkció alapú index létrehozására: CREATE INDEX iFunkcio ON t (a + b * (c – 1)); Ez után a fenti lekérdezés már hatékony lesz.
- Ne hozzunk létre feleslegesen túl sok indexet egy táblára, mert bonyolult lekérdezéseknél az adatbázis-kezelő nem feltétlenül azt/azokat az indexeket választja a végrehajtáshoz, amik a leghatékonyabb eredményt adnák. (Megjegyezzük, hogy lehetőség van egy lekérdezés megírásakor az adatbázis-kezelőt kényszeríteni arra, hogy bizonyos index-eket használjon, másokat pedig vegyen figyelmen kívül, amikor a lekérdezést futtatja.)
- Összetett indexeknél az un. „felvezető részekre” a rendszerek hatékonyan tudnak keresni, tehát a definícióban fontos a mezők sorrendje. Pl. ha (a, b, c) oszlop szerint összetett indexet készítünk, akkor hatékonyan tudunk keresni abban az esetben, ha a WHERE részben a következő mező kombinációkra adunk feltételt: (a), (a, b), (a, b, c), de a rendszer nem fogja használni az indexet a következő mezőkombinációkra vonatkozó feltételek esetén: (b), (c), (b,c). Ennek megfelelően pl. egy két oszlopot tartalmazó (x,y) indexet csak akkor hozzunk létre, ha (x,y)-ra együttesen nagyon gyakran keresünk (ilyenkor az összetett index hatékonyabb, mint külön-külön egy-egy index x-re és y-ra), de y-ra önmagában csak ritkán. Egyéb esetben inkább készítsünk egy indexet külön x-re és egyet külön y-ra. Ha minden eshetőségre be akarjuk

biztosítani magunkat, akkor készítsünk egy (x,y) és egy (y) indexet is. Nyilván egy (x) indexet pluszban létrehozni ekkor már értelmetlen.

Indexelt táblák használata esetén tisztában kell lennünk azzal, hogy mely függvények őrzik meg az indexelést, és melyek nem. Szerepeljen itt erre egy tipikus mintapélda (a datum mező DATE típusú, és erre van egy index):
SELECT * FROM t WHERE TO_CHAR(datum, 'YYYYMMDD') = '20040301';
SELECT * FROM t WHERE TRUNC(datum) = TO_DATE('20040301', 'YYYYMMDD');
Ez a két lekérdezés logikailag ugyanazt az eredményt adja. Az elsőnél azonban az indexet a rendszer nem fogja használni, mert a TO_CHAR típuskonverziós függvény nem őrzi meg az indexelést. Az adatbázis-kezelő itt tehát végigmegy majd minden soron. A második verzió viszont tökéletes, mert a TRUNC függvény (ami egyszerűen levágja az idő részt, tehát csak a dátum rész marad, és DATE típussal tér vissza) megőrzi az indexelést, így tehát az index struktúrát használva könnyedén megtalálja a keresett sorokat.

Gyakorló feladatsor

- Egy üzenet küldő program adatbázisát kell létrehoznunk!
- Tárolnunk kell a programot használó emberek különböző adatait (Név, Kor …) és különböző címeket. Ne feledkezzünk meg arról, hogy egy embernek több címe is lehet és egy cím több emberhez is tartozhat (például egy cég e-mail címe az összes dolgozóhoz tartozik)!
- A címek több félék lehetnek: e-mail cím, lakás cím, telefonszám…
- Az üzenetek különböző típusúak és ennek megfelelően különböző formájúak lehetnek. Például: Üdvözlés – Szia <Név>!, Meghívó – Tisztelt <Név>! Szeretettel meghívjuk….
- A programot különböző nyelven beszélő felhasználók használhatják, ezért minden üzenet típust el kell tárolni, minden lehetséges nyelven és természetesen azt is el kell tárolni, melyik felhasználó milyen nyelven beszél. (Érdemes eltárolni az összes lehetséges nyelvet egy külön táblában.)
- Ne feledkezzünk meg arról, hogy tárolnunk kell a program által küldött összes üzenetet is!



tSabTip:

SabTip	**Nev** .
UDV	Üdvözlés
MEGHIV	Meghívó

tNyelv:

Nyelv	**Nev** .
HUN	magyar
ENG	angol
GER	német

tSablon:

SabTip	**Nyelv**	**Sablon** .
UDV	HUN	Szia <Nev>!
UDV	ENG	Hi <Nev>!
UDV	GER	…
MEGHIV	HUN	Tisztelt <Nev>! Szeretettel meghívjuk Önt …

tModul:

Modul	**Nev** .
EMAIL	e-mail cím
POSTA	lakcím

0. Hozzuk létre az adatbázis tábláit!

```
CREATE TABLE tNyelv
(Nyelv VARCHAR(3) NOT NULL,
 Nev VARCHAR(20) NOT NULL,
 CONSTRAINT pk_tNyelv PRIMARY KEY(Nyelv))

CREATE TABLE tSabTip
(SabTip VARCHAR(10) NOT NULL,
 Nev VARCHAR(20) NOT NULL,
 CONSTRAINT pk_tSabTip PRIMARY KEY(SabTip))

CREATE TABLE tSablon
(SabTip VARCHAR(10) NOT NULL,
 Nyelv VARCHAR(3) NOT NULL,
 Sablon VARCHAR(1000) NOT NULL,
 CONSTRAINT pk_tSablon PRIMARY KEY (SabTip,Nyelv),
 CONSTRAINT fk_tSablon_tNyelv FOREIGN KEY (Nyelv)
     REFERENCES tNyelv(Nyelv),
 CONSTRAINT fk_tSablon_tSabTip FOREIGN KEY (SabTip)
     REFERENCES tSabTip(SabTip))

CREATE TABLE tUgyfel
(UfAzon INTEGER NOT NULL,
 Nev VARCHAR(50) NOT NULL,
 Kor INTEGER NOT NULL,
 Nyelv VARCHAR(3) NOT NULL,
 CONSTRAINT pk_tUgyfel PRIMARY KEY (UfAzon),
 CONSTRAINT fk_tUgyfel_tNyelv FOREIGN KEY (Nyelv)
     REFERENCES tNyelv(Nyelv))

CREATE TABLE tModul
(Modul VARCHAR(5) NOT NULL,
 Nev VARCHAR(20) NOT NULL,
 CONSTRAINT pk_tModul PRIMARY KEY(Modul))



CREATE TABLE tCim
(UfAzon INTEGER NOT NULL,
 Cim VARCHAR(100) NOT NULL,
 Modul VARCHAR(5) NOT NULL,
 CONSTRAINT pk_tCim PRIMARY KEY (UfAzon,Cim,Modul),
 CONSTRAINT fk_tCim_tUgyfel FOREIGN KEY (UfAzon)
     REFERENCES tUgyfel(UfAzon),
 CONSTRAINT fk_tCim_tModul FOREIGN KEY (Modul)
     REFERENCES tModul(Modul))

CREATE TABLE tUzenet
(UzAzon INTEGER NOT NULL,
 UfAzon INTEGER NOT NULL,
 Cim VARCHAR(100) NOT NULL,
 Modul VARCHAR(5) NOT NULL,
 SabTip VARCHAR(10) NOT NULL,
```

```
    Nyelv VARCHAR(3) NOT NULL,
    Uzenet VARCHAR(1500) NOT NULL,
    Datum DATE NOT NULL,
    CONSTRAINT pk_tUzenet PRIMARY KEY (UzAzon),
    CONSTRAINT fk_tUzenet_tCim
        FOREIGN KEY (UfAzon, Cim, Modul)
        REFERENCES tCim(UfAzon,Cim, Modul),
    CONSTRAINT fk_tUzenet_tSablon FOREIGN KEY (SabTip,Nyelv)
        REFERENCES tSablon(SabTip,Nyelv))
```

1. Adjuk meg az összes HUN nyelvű sablont!

```
SELECT Sablon FROM tSablon
WHERE Nyelv='HUN'
```

2. Adjuk meg az összes magyar nyelvű sablont!

```
SELECT Sablon
FROM tSablon s, tNyelv ny
WHERE ny.nyelv=s.nyelv AND ny.nev='magyar'
```

3. Adjuk meg úgy a tUzenet tábla összes rekordját, hogy az azonosítók helyett nevek szerepeljenek a megfelelő mezőkben!

```
SELECT
     uz.UzAzon as Üzenet_azonosító,
     uf.Nev as Ügyfél, uz.Cim as Cím, m.Nev as Modul,
     st.Nev as Sablon_típus, ny.Nev as Nyelv,
     uz.Uzenet as Üzenet, uz.Datum as Dátum
FROM
     tUzenet uz, tUgyfel uf, tModul m,
     tSabTip st, tNyelv ny
WHERE
     uf.UfAzon = uz.UfAzon AND m.Modul = uz.Modul AND
     st.SabTip = uz.SabTip AND ny.Nyelv = uz.Nyelv
```

4. Adjuk meg azoknak az ügyfeleknek a nevét, akik még nem kaptak üzenetet!

```
SELECT Nev FROM tUgyfel
WHERE UfAzon NOT IN (SELECT DISTINCT UfAzon FROM tUzenet)
```

5. Adjuk meg, hogy adott címre (hm@ize.hu) milyen nyelveken küldtünk üzenetet!

```
SELECT DISTINCT ny.Nev
FROM tUzenet uz, tNyelv ny
WHERE ny.Nyelv = uz.Nyelv AND uz.Cim ='hm@ize.hu'
```

6. Adjuk meg Sorok Sári összes címét modul szerint rendezve!

```
SELECT tCim.Cim, tCim.Modul
FROM tCim, tUgyfel
WHERE
     tCim.UfAzon = tUgyfel.UfAzon AND
     tUgyfel.Nev='Sorok Sári'
ORDER BY tCim.Modul
```

7. Határozzuk meg, hogy hány cím tartozik egy-egy modulhoz!

```
SELECT Modul, COUNT(1)
FROM tCim
```

```
GROUP BY Modul
```

Megjegyzés: Amelyik modulhoz egyetlen cím sem tartozik, az nem jelenik meg az eredmény táblában.

8. Adjuk meg (db-számra rendezve), hogy mely ügyfelek kaptak 3-nál több üzenetet és hányat?

```
SELECT uf.Nev, COUNT(*) AS Db
FROM tUzenet uz, tUgyfel uf
WHERE uz.UfAzon=uf.UfAzon
GROUP BY uf.Nev, uf.UfAzon
HAVING COUNT(*)>3
ORDER BY Db
```

9. Határozzuk meg milyen sablonokat kapott Gipsz Jakab! (Minden sablont csak egyszer adjunk meg!)

```
SELECT DISTINCT st.SabTip, st.Nev
FROM tUzenet uz, tUgyfel uf, tSabTip st
WHERE uz.UfAzon=uf.UfAzon AND uz.SabTip=st.SabTip AND
     uf.Nev='Gipsz Jakab'
```

10.*Mundjuk meg ki kapta a legtöbb üzenetet?

```
SELECT uf.Nev
FROM tUgyfel uf, tUzenet uz
WHERE uf.UfAzon=uz.UfAzon
GROUP BY uf.UfAzon, uf.Nev
HAVING count(*) in
(SELECT max(count(*))
 FROM tUzenet
 GROUP BY UfAzon)
```

11.** Kik azok az ügyfelek, akiknek több címük van?

```
SELECT u.nev
FROM tCim c, tUgyfel u
WHERE c.UfAzon=u.UfAzon
GROUP BY u.UfAzon,u.Nev
HAVING count(*)>=2
```

12. A 2-es ügyfelet 'cim' címmel vegyük fel a tCim táblába minden modulhoz!

```
INSERT INTO tCim (UfAzon, Cim, Modul)
(SELECT 2, 'cim', modul FROM tModul)
```

13. Töröljük az összes címet, ami német nyelvű ügyfélhez tartozik!

```
DELETE FROM tCim
WHERE UfAzon IN
     (SELECT u.UfAzon FROM tUgyfel u, tNyelv n
      WHERE u.Nyelv=n.Nyelv AND n.Nev='német')
```

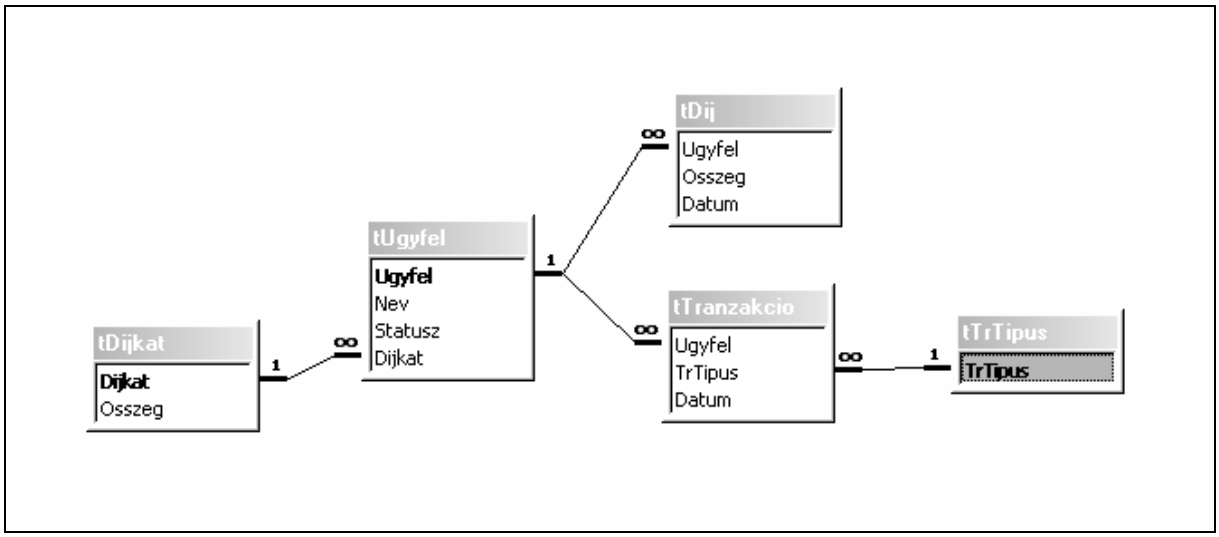
14.* Módosítsunk minden olyan üzenetet 'Helló'-ra, amely nyelve vagy magyar, vagy az ügyfél az utolsó a névsorban!

```
UPDATE tUzenet SET Uzenet='Hello' WHERE UzAzon IN
(SELECT uz.UzAzon
 FROM tUzenet uz, tUgyfel uf
 WHERE uz.UfAzon=uf.UfAzon AND
```

```
          uf.Nev IN (SELECT MAX(NEV) FROM TUGYFEL)
     UNION
     SELECT uz.UzAzon
     FROM tUzenet uz, tNyelv ny
     WHERE uz.Nyelv=ny.Nyelv AND
          ny.Nev='magyar')
```

6. óra

Gyakorló feladatsor



Példánkban egy bank elképzelt adatbázisából látunk részletet. Tároljuk az ügyfelek adatait és tranzakcióikat. A tranzakciók lehetséges típusait is külön táblában tároljuk. Az ügyfélnek minden tranzakció pénzbe kerül, hogy mennyibe –függetlenül a tranzakció típusától–, azt a tDijkat tábla Összeg mezője definiálja. Az ügyfeleknek a tranzakcióik után kirótt díjakat a tDij táblában tároljuk, amely kezdetben üres. A tTranzakció táblába on-line keletkeznek a tranzakciók, de az ezek után kiszabott díjakat csak havonta generáljuk le a tDij táblába olymódon, hogy díjgenerálás után minden egyes tranzakciónak megfelel majd egy rekord a tDij táblában.

A táblák attribútumainak típusai a következők: **tUgyfel**: *Ugyfel szám, Nev szöveg(50), Statusz szöveg(1), Dijkat szöveg(10); **tTranzakcio**: Ugyfel szám, TrTipus szöveg(10), Datum dátum/idő; **tTrTipus**: *TrTipus szöveg(10); **tDijkat**: *Dijkat szöveg(10), Osszeg szám; **tDij**: Ugyfel szám, Összeg szám, Datum dátum/idő.

Az elsődleges kulcsokat * jelzi, míg az idegen kulcsok az ábráról olvashatók le. Minden mező kötelező (nem lehet NULL érték).

A következő feladatokat SQL nyelvű parancsokkal kell megoldani. A szabványos SQL-en kívül bármely ismert adatbáziskezelő rendszer SQL bővítményeit használhatja (pl. MS SQL SERVER, ORACLE, DB2, stb.), de ennek nevét tüntesse fel a dolgozaton!

1. Készítse el az adattáblákat, valamint az elsődleges és idegen kulcsokat! Adja meg, hogy a tUgyfel tábla Statusz mezőjébe csak 'A' és 'I' értékeket lehessen írni (Aktív vagy Inaktív státuszú az ügyfél)!

```
CREATE TABLE tUgyfel
(Ugyfel NUMBER NOT NULL,
 Nev VARCHAR(50)  NOT NULL,
 Statusz VARCHAR(1) NOT NULL,
 Dijkat VARCHAR(10)  NOT NULL)

CREATE TABLE tTranzakcio
(Ugyfel NUMBER NOT NULL,
TrTipus VARCHAR(10) NOT NULL,
```

```
    Datum DATE NOT NULL)

    CREATE TABLE tTrTipus
    (TrTipus VARCHAR(10))

    CREATE TABLE tDijkat
    (Dijkat VARCHAR(10) NOT NULL,
     Osszeg NUMBER NOT NULL)

    CREATE TABLE tDij
    (Ugyfel NUMBER NOT NULL,
     Osszeg NUMBER NOT NULL,
     Datum DATE NOT NULL)

    ALTER TABLE tUgyfel ADD CONSTRAINT pk_tUgyfel
        PRIMARY KEY (Ugyfel)

    ALTER TABLE tTrTipus ADD CONSTRAINT pk_tTrTipus
        PRIMARY KEY (TrTipus)

    ALTER TABLE tDijkat ADD CONSTRAINT pk_tDijkat
        PRIMARY KEY (Dijkat)

    ALTER TABLE tUgyfel ADD CONSTRAINT fk_UgyfelDijkat
        FOREIGN KEY (Dijkat) REFERENCES tDijkat (Dijkat)

    ALTER TABLE tDij ADD CONSTRAINT fk_DijUgyfel
        FOREIGN KEY (Ugyfel) REFERENCES tUgyfel (Ugyfel)

    ALTER TABLE tTranzakcio
        ADD CONSTRAINT fk_TranzakcioUgyfel
        FOREIGN KEY (Ugyfel) REFERENCES tUgyfel (Ugyfel)

    ALTER TABLE tTranzakcio
        ADD CONSTRAINT fk_TranzakcioTrTipus
        FOREIGN KEY (TrTipus) REFERENCES tTrTipus (TrTipus)

    ALTER TABLE tUgyfel ADD CONSTRAINT ck_UgyfelStatusz
        CHECK (Statusz IN ('A','I'))
```

2. Készítsen lekérdezést, amely megadja az összes aktív státuszú ügyfél nevét és díjkategóriájának nevét!

```
    SELECT Nev, Dijkat FROM tUgyfel
    WHERE Statusz = 'A';
```

3. Készítsen lekérdezést, amely megadja, hogy a tranzakcióval rendelkezõ ügyfeleknek (egyenként) hány tranzakciójuk van!

```
    SELECT tUgyfel.Nev, COUNT(tTranzakcio.*)
    FROM tUgyfel, tTranzakcio
    WHERE tUgyfel.Ugyfel = tTranzakcio.Ugyfel
    GROUP BY tTranzakcio.Ugyfel, tUgyfel.Nev;
```

4. Készítsen lekérdezést, amely megadja, hogy mely olyan tranzakciótípusok szerepelnek a tTranzakcio táblában (egy típus csak egyszer jelenjen meg), amelyekhez tartozó

tranzakciót generáló ügyfél díjkategóriájában elõírt összeg nulla, vagy nagyobb, mint 15 Forint.

```
SELECT DISTINCT tTranzakcio.TrTipus
FROM tTranzakcio, tUgyfel, tDijkat
WHERE
    tTranzakcio.Ugyfel = tUgyfel.Ugyfel AND
    tUgyfel.Dijkat = tDijkat.Dijkat AND
    (tDijkat.Osszeg = 0) OR (tDijkat.Osszeg > 15);
```

5. Készítsen lekérdezést, amely megadja, hogy mely ügyfeleknek nincs tranzakciójuk!

```
SELECT Nev FROM tUgyfel
WHERE Ugyfel NOT IN (SELECT DISTINCT Ugyfel from
tTranzakcio);
```

6. Minden ’KAMAT’ típusú tranzakciót törölje a tTranzakció táblából!

```
DELETE FROM tTranzakcio WHERE TrTipus = 'KAMAT';
```

7. Tegyük fel, hogy összesen két díjkategória létezik. A ’STANDARD’ nevûnél a tranzakciók díja 25 Forint, míg a ’KEDVEZMENY’ nevûnél 10 Forint. Illessze be ezt a két rekordot a tDijkat táblába!

```
INSERT INTO tDijkat VALUES ('STANDARD', 25);
INSERT INTO tDijkat VALUES ('KEDVEZMENY', 10);
```

8. Módosítsa az összes ’Tóth’-tal kezdõdõ nevû ügyfél státuszát ’A’-ra!

```
UPDATE tUgyfel SET Statusz = 'A'
WHERE Nev LIKE 'Toth%';
```

9. Készítse el a hóvégi díjgeneráló parancsot! Ennek feladata, hogy a 2001. május hónapban keletkezett minden tranzakcióról (de a régebbiekrõl és a késõbbiekrõl ne!) készítsen egy-egy rekordot a tDij táblába úgy, hogy az Ugyfel és a Datum mezõk értékei legyenek ugyanazok, mint a tranzakciónál, az Összeg mezõ értéke pedig legyen az ügyfél díjkategóriájának megfelelõ érték!

```
INSERT INTO tDij
(SELECT
    tTranzakcio.Ugyfel, tDijkat.Osszeg, tTranzakcio.Datum
 FROM tTranzakcio, tDijkat, tUgyfel
 WHERE
    tTranzakcio.Ugyfel = tUgyfel.Ugyfel AND
    tDijKat.Dijkat = tUgyfel.Dijkat AND
    tTranzakcio.Datum > to_date('YYYYMMDD', '20010501')
    AND
    tTranzakcio.Datum < to_date('YYYYMMDD', '20010601'));
```

10. Adjon jogot a ’DIJGEN’ nevû felhasználónak, hogy a tDij táblába rekordokat szúrhasson be.

```
GRANT INSERT ON tDij TO abgyak;
```

11. Törölje le az adattáblákat!

```
DROP TABLE tDij
DROP TABLE tTranzakcio
DROP TABLE tTrTipus
DROP TABLE tUgyfel
DROP TABLE tDijkat
```