7.előadás: Adatbázisok-I.

dr. Hajas Csilla (ELTE IK)
http://sila.hajas.elte.hu/

SQL gyakorlatban: SQL DDL, SQL DCL

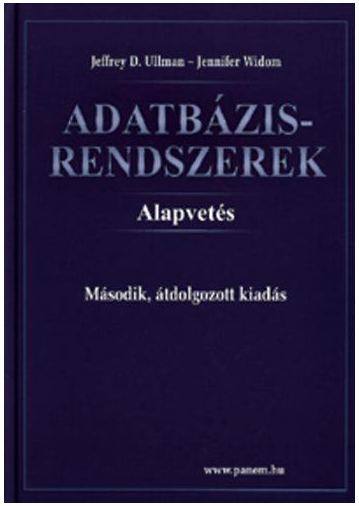
Tankönyv:

7.4.-7.5. Önálló megszorítások, Triggerek

8.1.-8.2., 8.5. Nézetek [8.3.-8.4. Indexek]

10.1. SQL DCL (biztonság, jogosultságok)

Folyt. 7.kieg.ea: SQL WITH RECURSIVE
http://sila.hajas.elte.hu/ABEA/07kieg_adatb_hcs.pdf



SQL áttekintés

Mi volt eddig?

6.1-6.4. Lekérdezések az SQL-ben: SELECT

6.5. SQL DML: INSERT, DELETE, UPDATE

6.6. ACID-tranzakciók, COMMIT, ROLLBACK

7.1.-7.3. Megszorítások CREATE TABLE, CONSTRAINTS

Hogyan folytatjuk? 7.ea.:

7.4. Önálló megszorítások
7.5. Triggerek (trigger)

8.1-8.2., 8.5. Nézetek (view)
-- 8.3.-8.4. Indexek (2.félévben)
-- 9.fej. SQL/PSM (később 8.ea)
10.1. SQL DCL (vezérlő ny.)

Folyt.köv. 7.kieg.ea (logika)

5.3-5.4 Datalog és rel.alg.
10.2. WITH RECURSIVE

Emlékeztető: Reláció sémák SQL-ben

- A legegyszerűbb formája:

```
CREATE TABLE relációnév (
        Attribútum deklarációk listája,
        Kiegészítő lehetőségek
);
```

- Az attribútum deklaráció legalapvetőbb elemei:

  Attribútumnév típus [kiegészítő lehetőségek]

-- itt: a **típus** olyan, amit az SQL konkrét megvalósítása támogat (gyakorlaton Oracle környezetben nézzük meg), Típusok, pl: INTEGER, REAL, CHAR, VARCHAR, DATE

-- A **kiegészítő lehetőségek** például [PRIMARY KEY], stb., [FOREIGN KEY, REFERENCES], [NOT NULL], [CHECK]

Megszorítások (áttekintés)

(1) Kulcsok és idegen kulcsok
- A hivatkozási épség fenntartása
- Megszorítások ellenőrzésének késleltetése

(2) Értékekre vonatkozó feltételek
- NOT NULL feltételek
- Attribútumra vonatkozó CHECK feltételek

(3) Sorokra vonatkozó megszorítások
- Sorra vonatkozó CHECK feltételek

(4) Megszorítások módosítása (constraints)

(5) Önálló megszorítások (create assertion)

(6) Triggerek (create trigger)

Példa: sor-alapú megszorítások

- Csak Joe bárja nevű söröző̋ben lehetnek drágábbak a sörök 5 dollárnál:

```
CREATE TABLE Felszolgál (
    söröző  CHAR(20),
    sör     CHAR(20),
    ár      REAL,
    CHECK (söröző= 'Joe bárja'
                OR ár <= 5.00)
);
```

- Itt fejeztük be! Jön: Írjuk át önálló megszorításra!

Önálló megszorítások: Assertions

- SQL aktív elemek közül a leghatékonyabbak nincs hozzárendelve sem sorokhoz, sem azok komponenseihez, hanem táblákhoz kötődnek.
- Ezek is az adatbázissémához tartoznak a relációsémákhoz és nézetekhez hasonlóan.
- CREATE ASSERTION <név>
  CHECK (<feltétel>);
- A feltétel tetszőleges táblára és oszlopra hivatkozhat az adatbázisséméból.

Példa: önálló megszorítások

- A Felszolgál(söröző, sör, ár) táblában nem lehet olyan söröző, ahol a sörök átlagára 5 dollárnál több

```
CREATE ASSERTION CsakOlcsó CHECK
 (
NOT EXISTS (
    SELECT söröző
    FROM Felszolgál
    GROUP BY söröző
    HAVING 5.00 < AVG(ár)
));
```

(SELECT .. olyan sörözők, ahol a sörök átlagosan drágábbak 5 dollárnál)

Példa: önálló megszorítások

- Az Sörvivó(név, cím, telefon) és Söröző(név, cím, engedély) táblákban nem lehet több bár, mint amennyi sörivó van.

```
CREATE ASSERTION KevésBár CHECK (
    (SELECT COUNT(*) FROM Söröző)
    <=
    (SELECT COUNT(*) FROM Sörivó)
);
```

Önálló megszorítások ellenőrzése

- Alapvetően az adatbázis bármely módosítása előtt ellenőrizni kell.
- Egy okos rendszer felismeri, hogy mely változtatások, mely megszorításokat érinthetnek.
  - Példa: a Sörök tábla változásai nincsenek hatással az iménti KevésBár megszorításra. Ugyanez igaz a Sörivók táblába történő beszúrásokra is.

Megszorítások v.s. triggerek

- Aktív elemek – olyan kifejezés vagy utasítás, amit egyszer eltároltunk az adatbázisban és azt várjuk tőle, hogy a megfelelő pillanatban lefusson (pl. adatok helyességének ellenőrzése)
- A megszorítás adatelemek közötti kapcsolat, amelyet az adatbázis-kezelő rendszernek fent kell tartania.
- Triggerek olyankor hajtódnak végre, amikor valamilyen megadott esemény történik, mint például sorok beszúrása egy táblába.

Miért hasznosak a triggerek?

- Az önálló megszorításokkal (assertions) sok mindent le tudunk írni, az ellenőrzésük azonban gondot jelenthet.
- Az attribútumokra és sorokra vonatkozó megszorítások ellenőrzése egyszerűbb (tudjuk mikor történik), ám ezekkel nem tudunk minden kifejezni.
- A triggerek esetén a felhasználó mondja meg, hogy egy megszorítás mikor kerüljön ellenőrzésre.

Esemény-Feltétel-Tevékenység szabályok

- A triggereket esetenként *ECA szabályoknak* (*event-condition-action*) *esemény-feltétel-tevékenység szabályoknak* is nevezik.
- Esemény: általában valamilyen módosítás a adatbázisban, INSERT, DELETE, UPDATE.
- Mikor?: BEFORE, AFTER, INSTEAD
- Mit?: OLD ROW, NEW ROW FOR EACH ROW
  OLD/NEW TABLE FOR EACH STATEMENT
- Feltétel : SQL igaz-hamis-ismeretlen feltétel.
- Tevékenység : SQL utasítás, BEGIN..END, SQL/PSM tárolt eljárás

Példa triggerre

- Ahelyett, hogy visszautasítanánk a Felszolgál(söröző, sör, ár) táblába történő beszúrást az ismeretlen sörök esetén, a Sörök(név, gyártó) táblába is beszúrjuk a megfelelő sort a gyártónak NULL értéket adva.

Példa: trigger definíció

```
CREATE TRIGGER SörTrig
  AFTER 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);
```

Esemény

Feltétel

Tevékenység

Triggerek --- 1

- A *triggerek,* amelyeket szokás *esemény-feltétel-tevékenység szabályoknak* is nevezni, az eddigi megszorításoktól három dologban térnek el:
- A triggereket a rendszer csak akkor ellenőrzi, ha bizonyos *események* bekövetkeznek.
  A megengedett események általában egy adott relációra vonatkozó beszúrás, törlés, módosítás, vagy a tranzakció befejeződése.

Triggerek --- 2

- A kiváltó esemény azonnali megakadályozása helyett a trigger először egy *feltételt* vizsgál meg
- Ha a trigger feltétele teljesül, akkor a rendszer végrehajtja a triggerhez tartozó *tevékenységet*. Ez a művelet ezután megakadályozhatja a kiváltó esemény megtörténtét, vagy meg nem történtté teheti azt.

Tankönyv példája (7.5. ábra)

```
-- Nem engedi csökkenteni a gyártásirányítók nettó bevételét:
CREATE TRIGGER NetBevétTrigger
AFTER UPDATE OF nettóBevétel ON GyártásIrányító
REFERENCING
    OLD ROW AS RégiSor,
    NEW ROW AS ÚjSor
FOR EACH ROW
WHEN (RégiSor.nettóBevétel > ÚjSor.nettóBevétel)
    UPDATE GyártásIrányító
    SET nettóBevétel = RégiSor.nettóBevétel
    WHERE azonosító = ÚjSor.azonosító;
```

Tankönyv példája (7.6. ábra)

```
-- Az átlagos nettó bevétel megszorítása:
CREATE TRIGGER ÁtlagNetBevétTrigger
AFTER UPDATE OF nettóBevétel ON GyártásIrányító
REFERENCING
    OLD TABLE AS RégiAdat,
    NEW TABLE AS ÚjAdat
FOR EACH STATEMENT
WHEN(500000 > (SELECT AVG(nettóBevétel)
                FROM GyártásIrányító)
BEGIN
    DELETE FROM GyártásIrányító
    WHERE (név, cím, azonosító) IN ÚjAdat;
    INSERT INTO gyártásIrányító
        (SELECT * FROM RégiAdat);
END;
```

Tankönyv példája (7.7. ábra)

-- A beszúrt sorok NULL értékeinek helyettesítésére, itt csak egyszerűen 1915-tel helyettesíti a trigger a NULL értéket, de ez akár egy bonyolult módon kiszámított érték is lehet:

(A BEFORE triggerek egy fontos alkalmazása, amikor egy beszúrandó sort a beszúrás előtt megfelelő formára hoznak)

```
CREATE TRIGGER ÉvJavítóTrigger
BEFORE INSERT ON Filmek
REFERENCING
    NEW ROW AS ÚjSor,
    NEW TABLE AS ÚjAdat
FOR EACH ROW
WHEN ÚjSor.év IS NULL
UPDATE ÚjAdat SET év=1915;
```

A mai témáink (folytatás) Nézetek

- Eddig: Tankönyv 7. Megszorítások, triggerek
- Ezután következik a Tankönyv 8.fejezete Nézetek és indexek: Adatbázisok-1 kurzuson csak 8.1. Nézetek; 8.2. Adatok módosítása nézeteken keresztül; 8.5. Tárolt nézettáblák; (Nézetek az AB1 gyakorlaton és vizsgán is!)
- (Tankönyv 8.3.-8.4. Indexek később az Adatbázisok-2 kurzuson lesz részletesen)
- Tankönyv 10.1. Jogosultságok, SQL DCL
- Tankönyv 10.2. Rekurzió (ez külön 7.ea.kieg)

Mik a nézettáblák?

- A nézettábla olyan reláció, amit tárolt táblák (vagyis alaptáblák) és más nézettáblák felhasználásával definiálunk.
- **`EMPLOYEES` table**

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALA
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	240
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	170
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	170
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	90
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	42
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	58
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	35
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	31
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	26
144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	25
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	105
174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	110
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	86
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	70
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	44
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	130
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	60
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	120
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	83

EMPLOYEE_ID	LAST_NAME	SALARY
149	Zlotkey	10500
174	Abel	11000
176	Taylor	8600

20 rows selected.

A nézettáblák előnyei

Korlátozható az adatok elérése

Bonyolult lekérdezések egyszerűbben kifejezhetők

Az adatfüggetlenség biztosítása

Ugyanazon adatokat különböző nézőpontokból szemlélhetjük

Virtuális vagy materializált?

- Kétféle nézettábla létezik:
  - Virtuális = nem tárolódik az adatbázisban, csak a relációt megadó lekérdezés.
  - Materializált = kiszámítódik, majd tárolásra kerül.

Nézettáblák létrehozása és törlése

- Létrehozása:

```
CREATE [OR REPLACE][FORCE|NOFORCE]
[MATERIALIZED] VIEW <név>
AS <lekérdezés>
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

- Alapesetben virtuális nézettábla jön létre.
- Nézettábla megszüntetése:

```
DROP VIEW <név>;
```

Példa: nézettábla létrehozása

- Mit_ihat(név, sör) nézettáblában a sörivók mellett azon söröket tároljuk, amelyeket legalább egy olyan sörözőben felszolgálnak, amelyet látogat:

```
CREATE VIEW Mit_ihat AS
    SELECT név, sör
    FROM Látogat L, Felszolgál F
    WHERE L.söröző = F.söröző;
```

Példa: nézettáblákhoz való hozzáférés

- A nézettáblák ugyanúgy kérdezhetők le, mint az alaptáblák.
  - A nézettáblákon keresztül az alaptáblák néhány esetben módosíthatóak is, ha a rendszer a módosításokat át tudja vezetni (lásd módosítások, SQL DML)
- Példa lekérdezés:

```
SELECT sör FROM Mit_ihat
WHERE név = 'Sally';
```

Módosítható nézettáblák

- Az SQL szabvány formálisan leírja, hogy mikor lehet egy nézettáblát módosítani és mikor nem, ezek a szabályok meglehetősen bonyolultak.
  - Ha a nézettábla definíciójában a SELECT után nem szerepel DISTINCT, további kikötések:
  - A WHERE záradékban R nem szerepelhez egy alkérdésben sem
  - A FROM záradékban csak R szerepelhet, az is csak egyszer és más reláció nem
  - A SELECT záradék listája olyan attribútumokat kell, hogy tartalmazzon, hogy az alaptáblát fel lehessen tölteni (vagyis kötelező a kulcsként vagy not null-nak deklarált oszlopok megadása)

Tankönyv példája: nézettáblára

Tk.8.1. Példa: Egy olyan nézettáblát szeretnénk, mely a Film(cím, év, hossz, színes, stúdióNév, producerAzon) reláció egy részét jelképezi, pontosabban a Paramount stúdió által gyártott filmek címét és gyártási évét

```
CREATE VIEW ParamountFilm AS
    SELECT cím, év
    FROM Filmek
    WHERE stúdióNév = 'Paramount';
```

Nézeteken instead-of-triggerek

Tk. 8.8. Példa: Az előző nézettábla módosítható, és hogy az alaptáblába való beszúráskor a stúdióNév attribútum helyes értéke , 'Paramount' legyen, ezt biztosítja ez az INSTEAD OF (helyette) típusú trigger:

```
CREATE TRIGGER ParamountBeszúrás
    INSTEAD OF INSERT ON ParamountFilm
    REFERENCING NEW ROW AS ÚjSor
    FOR EACH ROW
    INSERT INTO Filmek(cím, év, stúdióNév)
    VALUES(Újsor.cím, ÚjSor.év, 'Paramount');
```

Tárolt nézettáblák (Tk.8.5.)

- **CREATE [OR REPLACE]**
  MATERIALIZED VIEW <név>
  AS <lekérdezés>
- Adattárházaknál használják (MSc kurzusok)
- Probléma: minden alkalommal, amikor az alaptáblák valamelyike változik, a materializált nézettábla frissítése is szükségessé válhat.
  - Ez viszont néha túl költséges.
- Megoldás: Periodikus frissítése a materializált nézettábláknak, amelyek egyébként „nem aktuálisak”.

SQL gyakorlatra példa nézetekre

- Képezzük osztályonként az összfizetést, vegyük ezen számok átlagát, és adjuk meg, hogy mely osztályokon nagyobb ennél az átlagnál az összfizetés.

```
CREATE OR REPLACE VIEW osztaly_osszfiz AS
  SELECT onev, SUM(fizetes) ossz_fiz
  FROM dolgozo d, osztaly o
  WHERE d.oazon = o.oazon
  GROUP BY onev;

CREATE OR REPLACE VIEW atlag_koltseg AS
  SELECT SUM(ossz_fiz)/COUNT(*) atlag
  FROM osztaly_osszfiz;

SELECT * FROM osztaly_osszfiz
  WHERE ossz_fiz > (SELECT atlag FROM atlag_koltseg) ;
```

Példa nézetek helyett munkatáblák

- Ugyanez WITH átmeneti munkatáblával megadva, később a rekurziónál is fogunk még a WITH utasítással foglalkozni

```
WITH osztaly_osszfiz AS
                    (SELECT onev, SUM(fizetes) ossz_fiz
                     FROM dolgozo d, osztaly o
                     WHERE d.oazon = o.oazon
                     GROUP BY onev),
          atlag_koltseg AS
                    (SELECT SUM(ossz_fiz)/COUNT(*) atlag
                     FROM osztaly_osszfiz)
SELECT * FROM osztaly_osszfiz
  WHERE ossz_fiz > (SELECT atlag FROM atlag_koltseg);
```

Jogosultság-kezelés (Tk.10.1)

- Egy UNIX-szerű fájlrendszerhez hasonlítva az analógiák: Tipikusan írás, olvasás és végrehajtási jogosultságokról van szó.
- Az adatbázisok lényegesen bonyolultabbak a fájlrendszereknél, ezért az SQL szabványban definiált jogosultágok is összetettebbek.
  - Az SQL kilencféle jogosultságot definiál (SELECT, INSERT, DELETE, UPDATE, REFERENCES, USAGE, TRIGGER, EXECUTE, UNDER)
  - Bizonyos „résztvevőkhöz” sorolja a jogosultságokat, például rendszergazda, korlátozott jogosultságokkal rendelkező felhasználó. Spec. PUBLIC (mindenki)

SQL DCL: GRANT utasítás

- Jogosultságok megadásának szintaktikája:

  GRANT <jogosultságok listája>

  ON <reláció vagy másféle objektum>

  TO <jogosultsági azonosítók listája>;

- Ehhez hozzáadható:

  WITH GRANT OPTION

Példa: GRANT

```
GRANT SELECT, UPDATE(ár)
    ON Felszolgál
    TO Sally;
```

- Ez után Sally kérdéseket adhat meg a Felszolgál táblára vonatkozóan és módosíthatja az ár attribútumot.

Jogosultságok

- A relációkra vonatkozó jogosultságok:

  SELECT = a reláció lekérdezésének joga.

  INSERT = sorok beszúrásának joga.

  (egyetlen attribútumra is vonatkozhat)

  DELETE = sorok törlésének joga.

  UPDATE = sorok módosításának a joga.

  (szintén egy attribútumra is vonatkozhat)

Példa: jogosultságok

- Az alábbi utasítás esetében:

```
INSERT INTO felh.Sörök(név)
SELECT sör FROM felh.Felszolgál f
   WHERE NOT EXISTS
      (SELECT * FROM felh.Sörök
       WHERE név = f.sör);
```

azok a sörök, amelyek még nincsenek benne a sörök táblában. A beszúrás után a gyártó értéke NULL.

- Ehhez az INSERT utasítás végrehajtásához szükséges: SELECT jogosultság a felh (user) felszolgál és sörök tábláira és INSERT jog a Sörök tábla név attribútumára vonatkozóan.

Adatbázis objektumok

- Jogosultságokat nézetekre és materializált nézetekre vonatkozóan is megadhatunk.
- Egy másik fajta jogosultság lehet pl. adatbázis objektumok létrehozásának a joga: pl. táblák, nézetek, triggerek.
- A nézettáblák segítségével tovább finomíthatjuk az adatokhoz való hozzáférést.

Példa: nézettáblák és jogosultságok

- Tegyük fel, hogy nem szeretnénk SELECT jogosultságot adni az Dolgozók(név, cím, fizetés) táblában.
- Viszont a BiztDolg nézettáblán már igen:

```
CREATE VIEW BiztDolg AS
    SELECT név, cím FROM Dolgozók;
```

- A BiztDolg nézettáblára vonatkozó kérdésekhez nem kell SELECT jog a Dolgozók táblán, csak a BiztDog nézettáblán.

Jogosultságok megadása

- A magunk készítette objektumok esetében az összes jogosultsággal rendelkezünk.
- A felhasználókat egy jogosultsági azonosító (authorization ID) alapján azonosítjuk, általában ez a bejelentkezési név, ennek felhasználásával másoknak is megadhatunk jogosultságokat.
- vagy a PUBLIC jogosultsági azonosítót is használhatjuk, a PUBLIC jogosultság minden felhasználó számára biztosítja az adott jogot.
- A WITH GRANT OPTION utasításrész lehetővé teszi, hogy aki megkapta a jogosultságot, tovább is adhassa azt.

Példa: Grant Option

```
GRANT UPDATE ON Felszolgál TO Sally
  WITH GRANT OPTION;
```

- Ez után Sally módosíthatja a Felszolgál táblát és tovább is adhatja ezt a jogosultságot.
  - Az UPDATE jogosultságot korlátozottan is továbbadhatja: `UPDATE(ár)ON Felszolgál`.

Jogosultságok visszavonása

REVOKE <jogosultságok listája>
ON <reláció vagy más objektum>
FROM <jogosultsági azonosítók listája>;

- Az általunk kiadott jogosultságok ez által visszavonódnak.
  - De ha máshonnan is megkapták ugyanazt a jogosultságot, akkor az még megmarad.

Az Eljut-feladat (Tk. 10.2)

7.kieg.ea: SQL WITH RECURSIVE

http://sila.hajas.elte.hu/ABEA/07kieg_adatb_hcs.pdf

Tankönyv

10.2. Az Eljut-feladat (monoton, lineáris rekurzió) megoldása
(a.) az SQL-ben WITH RECURSION utasítással

Kiegészítések a rekurzióhoz, az Eljut feladat megoldása:

(b.) Oracle megoldások/2: CONNECT BY PRIOR
(c.) Oracle megoldások/3: WITH alkérdés faktorizáció

Az Eljut-feladat

Tankönyv 10.2. fejezet példája (az ELJUT feladat)

- Jaratok(legitarsasag, honnan, hova, koltseg, indulas, erkezes) táblában repülőjáratok adatait tároljuk.
- A járatok táblát létrehozó script:

  http://people.inf.elte.hu/sila/eduAB/jaratok_tabla.txt
- Mely (x,y) párokra lehet eljutni x városból y városba?
- Ezt egy relációs algebrai kifejezésként nem tudjuk megadni zárt alakban, klasszikus SQL SELECT utasítással sem tudjuk kifejezni, csak azt tudjuk, hogy átszállás nélkül, egy, két, stb… átszállással:

Az Eljut-feladatnak nincs algebrai megoldása

```
select distinct honnan, hova
    from jaratok
union
select j1.honnan, j2.hova
    from jaratok j1, jaratok j2
    where j1.hova=j2.honnan
union
select j1.honnan, j3.hova
    from jaratok j1, jaratok j2, jaratok j3
    where j1.hova=j2.honnan
    and j2.hova=j3.honnan
--- union stb... Ezt így nem lehet felírni…
```

Az Eljut-feladat Datalogban

Tankönyv 10.2. fejezet példája (az ELJUT feladat)

- Jaratok(legitarsasag, honnan, hova, koltseg, indulas, erkezes) EDB-táblában repülőjáratok adatait tároljuk.

  Mely (x,y) párokra lehet eljutni x városból y városba?

- Datalogban felírva (lineáris rekurzió)

  Eljut(x, y) <- Jaratok(l, x, y, k, i, e)
  Eljut(x, y) <- Eljut(x, z) AND Jaratok(l, z, y, k, i, e)

- Vagy másképp felírva Datalogban (mi a különbség?)

  Eljut(x, y) <- Jaratok(_, x, y, _, _, _) --- anonimus változók
  Eljut(x, y) <- Eljut(x, z) AND Eljut(z, y) --- nem lineáris rek.

Az Eljut feladat SQL-99 szabványban

- Datalog LINEÁRIS, MONOTON rekurzió átírható:
  Eljut(x, y) <- Jaratok(l, x, y, k, i, e)
  Eljut(x, y) <- Eljut(x, z) AND Jaratok(l, z, y, k, i, e)

- Hova, mely városokba tudunk eljutni Budapestről?

```
WITH RECURSIVE Eljut(honnan, hova) AS
        (SELECT honnan, hova FROM Jaratok
      UNION
          SELECT Eljut.honnan, Jaratok.hova
          FROM Eljut, Jaratok
          WHERE Eljut.hova = Jaratok.honnan)
SELECT hova FROM Eljut WHERE honnan='Bp';
```

SQL-99 szabvány: Rekurzív lekérdezés

- A WITH utasítás több ideiglenes relációra vonatkozó definíciója:

WITH [RECURSIVE] R_1 AS <R_1 definíciója>
[RECURSIVE] R_2 AS <R_2 definíciója>
...
[RECURSIVE] R_n AS <R_n definíciója>
< R_1,R_2,...,R_n relációkat tartalmazó lekérdezés>

Oracle megoldások: with utasítással

- Az Oracle SQL a WITH RECURSIVE utasítást azt úgy támogatja, hogy WITH szerepel RECURSIVE nélkül, és UNION helyett UNION ALL-t lehet csak beírni
- WITH utasítással (Oracle 11gR2 verziótól használható)

```
WITH eljut (honnan, hova) as
   (select honnan, hova from jaratok
        UNION ALL
    select jaratok.honnan, eljut.hova
    from jaratok, eljut
    where jaratok.hova=eljut.honnan)
SEARCH DEPTH FIRST BY honnan SET SORTING
CYCLE honnan SET is_cycle TO 1 DEFAULT 0
select distinct honnan, hova from eljut order by honnan;
```

Oracle megoldások: connect by

- SELECT DISTINCT hova FROM jaratok
  WHERE HOVA <> 'DAL'
  START WITH honnan = 'DAL'
  CONNECT BY NOCYCLE PRIOR hova = honnan;

- SELECT LPAD(' ', 4*level) || honnan, hova,
  level-1 Atszallasok,
  sys_connect_by_path(honnan||'->'||hova, '/'),
  connect_by_isleaf, connect_by_iscycle
  FROM jaratok
  START WITH honnan = 'SF'
  CONNECT BY NOCYCLE PRIOR hova = honnan;

JÖN: 7.kieg.ea Eljut feladat

- A 7.ea második felét bővebben lásd a 7.kieg.ea fájlban:
- Tk.5.3-5.4 Datalog: Logika az SQL lekérdezésekhez
- Tk.10.2. **Eljut feladat** megoldása az SQL-ben (WITH)

http://sila.hajas.elte.hu/ABEA/07kieg_adatb_hcs.pdf

- Következő héten 8.ea-on folytatjuk, hogyan lehet az Eljut feladatot SQL/PSM programmal megvalósítani?
- **Rek1.feladat:** Mely (x,y) párokra lehet egy vagy több átszállással eljutni x városból y-ba?
- **Rek2.feladat:** Mely (x,y,n,k) értékekre lehet eljutni x városból y-ba, egy vagy több n darab átszállással és k összes költséggel?
- **Rek3.feladat:** átszálláskor az érkező járatnak legalább egy órával a rákövetkező indulás előtt meg kell érkeznie.

Kérdés/Válasz

- Köszönöm a figyelmet! Kérdés/Válasz?

Gyakorlás: Oracle példatár

- SELECT, Oracle hierarchikus lekérdezések (Példatár 3.fej.)
- DML-utasítások, tranzakciók
  - DML-utasítások: insert, update, delete (Példatár 5.fej.)
  - Adatbázis-tranzakciók: commit, rollback, savepoint
- DDL-utasítások, create table, create view
  - DDL-utasítások: adattáblák létrehozása, módosítása, integritási megszorítások (Példatár 5.fejezet folyt.) és
  - Nézettábla létrehozása és törlése, táblák tartalmának módosítása nézettáblákon keresztül (Példatár 6.fej.)