

9.előadás: Adatbázisok-I.

dr. Hajas Csilla (ELTE IK)

<http://sila.hajas.elte.hu/>

Logika a relációs lekérdezésekhez: Datalog és Rekurzió (Eljut feladat)

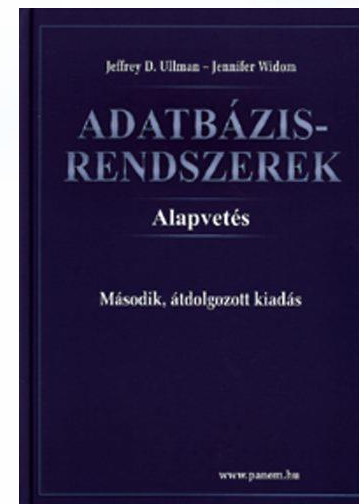
Tankönyv:

5.3.-5.4. Logika a relációkhoz: Datalog

10.2. WITH munkatáblák a SELECT-hez,
Rekurzív lekérd.: WITH RECURSIVE

[Az előadások Ullman-Widom: Adatbázisrendszerek
Alapvetés könyv; J.D.Ullman diái alapján készültek]

http://sila.hajas.elte.hu/AB1ea/REL5_adatb_hcs.pdf



Témakörök

Tankönyv

5.3. Logika a relációkhoz: Datalog

5.4. A relációs algebra és nem-rekurzív biztonságos Datalog kifejező erejének összehasonlítása

10.2. Az Eljut-feladat megoldása

(a.) Datalogban (monoton, lineáris rekurzió)

(b.) SQL-ben WITH RECURSION utasítással

Oracle kiegészítések - az Eljut feladat megoldása:

(c.) Eljut feladat PL/SQL-ben (illetve SQL/PSM-ben)

(d.) Oracle megoldások/új: WITH alkérdés faktorizáció

(e.) Oracle megoldások/régi: CONNECT BY PRIOR

Következik...

- Relációs algebra korlátai: bizonyos típusú lekérdezéseket nem tudunk relációs algebrával kifejezni...
- Nézzünk meg olyan logikai felépítést, amivel az ilyen rekurzív jellegű lekérdezések könnyen megoldhatók.
- Ez a nyelv: a Datalog

Milyen fontos rekurzív feladatok vannak?

I. Hierarchiák bejárása

➤ Leszármazottak-ősök

Szülője(szülő, gyerek)

- Keressük Marika őseit

➤ Vállalati hierarchia felettes-beosztott

Dolgozó(ID, fizetés)

Főnöke(főn_ID, dolg_ID)

Projekt(név, vez_ID)

- Keressük az összes fizetést az 'X'- projektnek

➤ Alkatrész struktúra (mely alkatrésznek mely alkatrész része, és annak mely alkatrész, stb)

Milyen fontos rekurzív feladatok vannak?

II. Gráf jellegű bejárások

➤ Repülőgép járatok, eljut-feladat

Járatok(járat, honnan, hová, költség)

➤ Keressük a legolcsóbb utat 'A'-ból 'B'-be

➤ Közösségi hálók

Ki-kinek az ismerőse, Twitterben ki-kit követ

Kiegészítés a gráf adatbázisokról

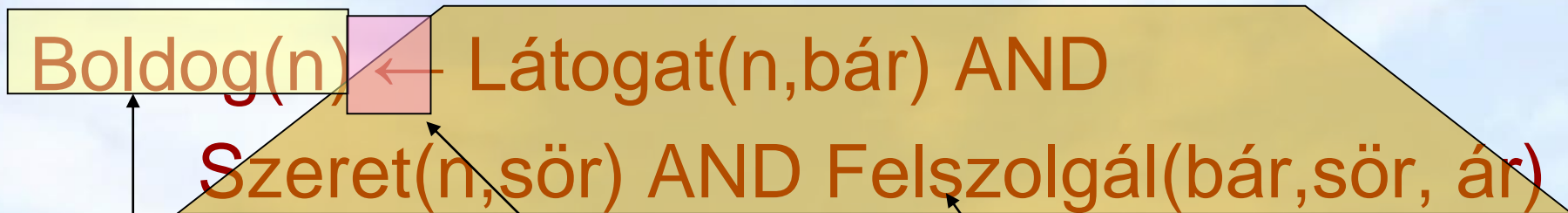
➤ Gráfok könnyen megadhatók relációs táblával, a gráf lekérdezések egyre gyakoribb feladatok, ezek relációs megoldása hatékonysági kérdés. Vannak kimondottan gráf-adatbázisok.

Logika, mint lekérdező nyelv

- Datalog: logikai, szabály-alapú nyelv szabály: IF feltétel THEN eredmény, ahol a feltétel relációkkal legyen megadható és az eredmény pedig az output tábla sorait eredményezze
- A példa a Sörivők adatbázison alapszik
 - Látogat(név, söröző),
 - Szeret(név, sör),
 - Felhasznál(söröző, sör, ár).
- Adjunk meg egy olyan szabályt, amely lekérdezi a „boldog” sörivőket, akik olyan sörözőt látogatnak, ahol felszolgálnak olyan sört, amelyet szeretnek.

Datalog szabályok felépítése

Datalogban a relációkat predikátumokkal reprezentáljuk.



Szabály feje = „akkor”
egy egyszerű rész cél

Szabály törzse = „ha” =
rész célok AND-el kapcsolva

„if-then” („ha-akkor”) szabályok

Rész célok: relációs atomi formulák (argumentumai változók vagy konstansok), akkor igaz, ha a behelyettesítési értékekből álló n -est tartalmazza a rel.atomi formulához tartozó tábla

Interpretáció: n akkor boldog, ha létezik olyan bár, sör, ár érték, amire $(n,bár)$ sora a Látogat táblának, stb, többire is.

Példa: Aritmetikai részcélok

- **Példa:** Egy sört akkor mondjuk, hogy “olcsó”, ha van legalább két söröző, ahol a sör \$2-nál kevesebbe kerül.

$$\text{Olcsó(sör)} \leftarrow \text{Felszolgál}(\text{bar1}, \text{sör}, \text{ár1}) \text{ AND}$$
$$\text{Felszolgál}(\text{bar2}, \text{sör}, \text{ár2}) \text{ AND } \text{ár1} < 2.00$$
$$\text{AND } \text{ár2} < 2.00 \text{ AND } \text{bar1} \neq \text{bar2}$$

- **Feladatok:** Írjuk át SQL SELECT utasításra ezt is és az előző oldalon levő szabályt is!
- Átírás ellenőrzése egy későbbi lapon lesz az átírásoknál: SQL -és- alap relációs algebra -és- (nem rekurzív) biztonságos Datalog program

Negált részcélok

- Negált részcélok, amelyek előtt NOT szerepel, NOT megváltoztatja a az eredményt a negáltjára
- **Példa:** Legyen adva az $Arc(a,b)$ reláció, amely egy gráf éleit tartalmazza.
- $S(x,y)$ keressük azokat a párokat x -ből y -ba, ahol el tudunk jutni 2 hosszú úton x -ből y -ba, de közvetlenül nem tudunk eljutni x -ből y -ba.

$$S(x,y) \leftarrow Arc(x,z) \text{ AND } Arc(z,y) \\ \text{AND NOT } Arc(x,y)$$

Biztonságossági elvárás

- Ha biztosítani szeretnénk, hogy egy szabály kiértékelésekor kapott reláció véges legyen, akkor bizonyos megszorításokat kell tennünk a változókra nézve.
- A szabályok kiértékelhetőek legyenek, ehhez:
- A szabályban szereplő minden változónak elő kell fordulnia a törzsben nem-negált relációs rész-célban is.

Biztonságos szabályok

- A rule is *safe* if:
 1. Each distinguished variable,
 2. Each variable in an arithmetic subgoal, and
 3. Each variable in a negated subgoal,
also appears in a nonnegated, relational subgoal,

Példa: Nem biztonságos szabályokra

- Each of the following is unsafe and not allowed:
 1. $S(x) \leftarrow R(y)$
 2. $S(x) \leftarrow R(y) \text{ AND } x < y$
 3. $S(x) \leftarrow R(y) \text{ AND NOT } R(x)$
- In each case, an infinity of x 's can satisfy the rule, even if R is a finite relation.

Datalog programok

- **Datalog program** = collection of rules.
- In a program, predicates can be either
 1. EDB relációk = **Extensional Database** = stored table (csak a törzsben szereplő relációk)
 2. IDB relációk = **Intensional Database** = relation defined by rules (szerepel fej-ben)
- Never both! No EDB in heads.

Datalog programok kiértékelése

- As long as there is no recursion, we can pick an order to evaluate the IDB predicates, so that all the predicates in the body of its rules have already been evaluated.
- If an IDB predicate has more than one rule, each rule contributes tuples to its relation.

Példa: Datalog program

- Using EDB **Sells(bar, beer, price)** and **Beers(name, manf)**, find the manufacturers of beers Joe doesn't sell.

JoeSells(b) ← Sells('Joe''s Bar', b, p)

Answer(m) ← Beers(b,m)

AND NOT JoeSells(b)

Példa: Kiértékelése

- **Step 1:** Examine all **Sells** tuples with first component 'Joe''s Bar'.
 - Add the second component to **JoeSells**.
- **Step 2:** Examine all **Beers** tuples (b,m).
 - If b is not in **JoeSells**, add m to Answer.

Datalog kifejező ereje

- Rekurzió nélkül a biztonságos Datalog program kifejező ereje megegyezik az alap relációs algebrával (azok és csak azok a lekérdezések fejezhetők ki az egyikben, mint a másokban és viszont)
 - Mégpedig ugyanaz, mint az SQL-ben select-from-where záradékokkal (összesítések és csoportosítás nélkül)
- Rekurzív biztonságos Datalog kifejező ereje nagyobb, mint az alap relációs algebra és SQL kifejező ereje (lásd Eljut feladat)

Relációs algebrai kifejezések átírása nem-rekurzív Datalog programra

- Mi a leggyakrabban előforduló típus, amiből építkeznek? $\prod_{\text{Lista}}(\sigma_{\text{Felt}}(\mathbf{R} \bowtie \mathbf{S} \bowtie \dots))$

Ezt a komponenst támogatja legerősebben az SQL is: **SELECT lista**

FROM táblák összekapcsolása

WHERE felt

Ez felel meg egy Datalog szabálynak.

- **Halmazműveletek:** kezdjük ezzel, hogyan lehet a metszetet és különbséget Datalog szabállyal, és az egyesítést Datalog programmal kifejezni.

Relációs algebra és Datalog ---1

Rel.algebrai műveletek hogyan néznek ki Datalogban?

Halmazműveletek: T.f.h $R(x_1, \dots, x_n)$, $S(x_1, \dots, x_n)$

predikátumokhoz tartozó reláció $R(A_1, \dots, A_n)$, $S(A_1, \dots, A_n)$

➤ $R \cap S$ metszetnek megfelelő szabály:

Válasz(x_1, \dots, x_n) $\leftarrow R(x_1, \dots, x_n)$ AND $S(x_1, \dots, x_n)$

➤ $R - S$ különbségnek megfelelő szabály:

Válasz(x_1, \dots, x_n) $\leftarrow R(x_1, \dots, x_n)$ AND NOT $S(x_1, \dots, x_n)$

➤ $R \cup S$ unió műveletet egyetlen szabállyal nem tudom felírni, mert a törzsben csak AND lehet, OR nem. Ehhez több szabályból álló Datalog program kell:

Válasz(x_1, \dots, x_n) $\leftarrow R(x_1, \dots, x_n)$

Válasz(x_1, \dots, x_n) $\leftarrow S(x_1, \dots, x_n)$

Relációs algebra és Datalog ---2

Kiválasztás:

- $\sigma_{x_i \theta x_j}(R)$ kifejezésnek megfelelő szabály :
Válasz(x_1, \dots, x_n) $\leftarrow R(x_1, \dots, x_n)$ AND $x_i \theta x_j$
- $\sigma_{x_i \theta c}(E1)$ kifejezésnek megfelelő szabály:
Válasz(x_1, \dots, x_n) $\leftarrow R(x_1, \dots, x_n)$ AND $x_i \theta c$

Vetítés:

- $\Pi_{A_{i_1}, \dots, A_{i_k}}(R)$ kifejezésnek megfelelő szabály:
Válasz(x_{i_1}, \dots, x_{i_k}) $\leftarrow R(x_1, \dots, x_n)$

Megjegyzés: név nélküli anonymus változók, amelyek csak egyszer szerepelnek és mindegy a nevük azt aláhúzás helyettesítheti. Például:

HosszúFilm(c,é) \leftarrow Film(c,é,h,_,_,_) AND $h \geq 100$

Relációs algebra és Datalog ---3

Természetes összekapcsolás: Tegyük fel, hogy
 $R(A_1, \dots, A_n, C_1, \dots, C_k)$ és $S(B_1, \dots, B_m, C_1, \dots, C_k)$

- $R \bowtie S$ kifejezésnek megfelelő szabály:

Válasz($x_1, \dots, x_n, y_1, \dots, y_m, z_1, \dots, z_k$) \leftarrow

$\leftarrow R(x_1, \dots, x_n, z_1, \dots, z_k) \text{ AND } S(y_1, \dots, y_m, z_1, \dots, z_k)$

- A felírt szabályok biztonságosak.
- Minden Q relációs algebrai kifejezéshez van nem rekurzív, biztonságos, negációt is tartalmazó Datalog program, amelyben egy kitüntetett IDB predikátumhoz tartozó kifejezés ekvivalens a Q lekérdezéssel.
- A nem rekurzív, biztonságos, negációt is tartalmazó Datalog kifejezőerő tekintetében EKVIVALENS a relációs algebrával.

Biztonságos Datalog program átírása SQL-be

- Először a **nem-rekurzív biztonságos Datalog** programok SQL-be való átírását nézzük meg (példákon keresztül is), majd ezután vizsgáljuk meg a rekurzív Datalog programok átírását egyrészt SQL WITH RECURSIVE szerint, majd SQL/PSM (PL/SQL) programban megvalósítva.
- Egy Datalog szabály megfelel az SQL-ben egy **[SELECT lista FROM táblák WHERE feltétel]**-nek
- Datalog program: **halmazműveletek** felismerése (ugyanazok a változók) és átírva SQL-be. **Példák:**

Példa: Datalog átírása SELECT-re

- Szeret(Név, Gyümölcs) sémájú tábla alapján
Kik szeretik az almát is és a körtét is logikai megoldását írjuk át SQL SELECT utasításra!

Válasz(N) ← Szeret(N, 'alma') AND Szeret(N, 'körte')

A törzs két predikátumában megegyeznek a szabad változók, ami a metszet halmazműveletnek felel meg:

```
SELECT Név FROM Szeret WHERE Gyümölcs='alma'
```

```
INTERSECT
```

```
SELECT Név FROM Szeret WHERE Gyümölcs='körte';
```

Példa: Datalog átírása SELECT-re

- Az előző szabály másik ekvivalens átírása

Válasz(N) ← Szeret(N,'alma') AND Szeret(N,'körte')

Ez a szabály ugyanaz, mintha kiírnánk a változókat:

Válasz(N1) ← Szeret(N1,Gy1) AND Szeret(N2,Gy2)
AND N1=N2 AND Gy1= 'alma' AND Gy2='körte'

Ennek a másik logikának megfelelő SELECT pedig:

```
SELECT s1.Név FROM Szeret s1, Szeret s2  
WHERE s1.Név= s2.Név AND s1.Gyümölcs='alma'  
AND s2.Gyümölcs='körte';
```


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://sila.hajas.elte.hu/AB1gy/create_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] R1 AS <R1 definíciója>  
      [RECURSIVE] R2 AS <R2 definíciója>  
      ...  
      [RECURSIVE] Rn AS <Rn definíciója>  
< R1,R2,...,Rn relációkat tartalmazó lekérdezés>
```

Másik példa: Rekurzív Datalog

- A testvérek (féltestvérek) gyerekei első unokatestvérek, az első unokatestvérek gyerekei másod-unokatestvérek, és így tovább. Hívjuk egyszerűen unokatestvéreknek, akik valamilyen szinten unokatestvérek. A rokonok azok, akik közös ősnek leszármazottjai.
- Milyen Datalog program írja ezt le?

testvér(x,y) ←gyerek(x,z),gyerek(y,z),x ≠ y

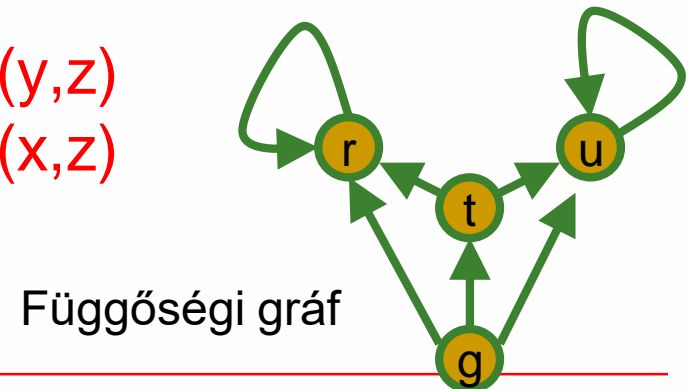
unokatestvér(x,y) ←gyerek(x,z),gyerek(y,v),testvér(z,v)

unokatestvér(x,y) ←gyerek(x,z),gyerek(y,v),unokatestvér(z,v)

rokon(x,y) ←testvér(x,y)

rokon(x,y) ←rokon(x,z),gyerek(y,z)

rokon(x,y) ←rokon(z,y),gyerek(x,z)



Példa folyt:Rekurzív Datalog átírása SQL-be

WITH

T(x,y) as (select G1.u x, G2.w y from G G1,G G2
where G1.w=G2.u and G1.u<>G2.u),

RECURSIVE U(x,y) as (select G1.u x, G2.u y
from G G1, G G2, T where T.x=G1.w and T.y=G2.u)

UNION (select G1.u x, G2.u y
from G G1, G G2, U where U.x=G1.w and U.y=G2.u),

RECURSIVE R(x,y) as (select * from T) UNION
(select R.x x,G.u y from R,G where R.y=G.w) UNION
(select G.u x, R.y y from R,G where R.x=G.w)

(select T.x, T.y, 'T' from T union
select U.x, U.y, 'U' from U union
select R.x, R.y, 'R' from R);

Rekurzív lekérdezések

- **Datalog rekurzió** segít megérteni az SQL-99 szabványban bevezetett **rekurzív lekérdezések WITH RECURSIVE** záradékát.
- A BSc-n **csak MONOTON rekurziót** vesszük, vagyis nem használjuk nem-monoton különbség műveletet, nincs csoportosítás-aggregálás (ugyanis az olyan lekérdezések, amelyek nem-monotonok, megengedik a negációt és aggregálást az olyan különös hatással van a rekurzióra, ezt csak MSc kurzusokon vesszük).
- A rekurzív Eljut-feladatnak az Oracle **CONNECT BY** záradékkal ill. az egyéb **megoldásait** is megnézzük!

Eljut feladat PL/SQL-ben ---1

- **Rek1.feladat:** Mely (x, y) várospárokra lehet egy vagy több átszállással eljutni x városból y városba?
- Ehhez hozzuk létre eljut(honnan,hova) táblát,
DROP TABLE eljut;
CREATE TABLE eljut(
 honnan VARCHAR2(10),
 hova VARCHAR2(10));
- Írjunk egy olyan PL/SQL programot, ami feltölti az ELJUT táblát a sorait a járatok tábla alapján (ehhez ciklust szervezni, az insert több sor felvitele 2.alakja alkérdéssel járatok és eljut táblák alapján)

Eljut feladat PL/SQL-ben ---2

- Az ELJUT feladat megoldása Oracle PL/SQL-ben
- A ciklus során ellenőrizni kell, hogy addig hajtsuk végre a ciklust, amíg növekszik az eredmény (Számláló)
- **DECLARE** RegiSzamlalo Integer;
UjSzamlalo Integer;
- Deklarációs rész után **BEGIN ... END;** között az utasítások, először az eljut táblának kezdeti értéket adunk (a megvalósításnál az INSERT-nél figyelni, hogy ne legyenek ismétlődő sorok: `select distinct delete from eljut;`
`insert into eljut (SELECT distinct honnan, hova FROM jaratok);`

Eljut feladat PL/SQL-ben ---3

- Szamlalo változóknak adunk kiindulási értéket:

```
RegiSzamlalo := 0;
```

```
select count(*) into UjSzamlalo from eljut;
```

- A ciklust addig kell végrehajtani, amíg növekszik az eredmény (Szamlalo) duplikátumokra figyelni!

LOOP

```
insert into eljut (lásd a köv.oldalon...)
```

```
select count(*) into UjSzamlalo from eljut;
```

```
EXIT WHEN UjSzamlalo = RegiSzamlalo;
```

```
RegiSzamlalo := UjSzamlalo;
```

```
END LOOP;
```

```
commit;
```

Eljut feladat PL/SQL-ben ---4

- Az eljut tábla növelése a ciklusban, figyelni kell a duplikátumokra, csak olyan várospárokat vegyünk az eredményhez, ami még nem volt!

insert into eljut

```
(select distinct eljut.honnan, jaratok.hova  
from eljut, jaratok --- *from (lineáris rekurzió)  
where eljut.hova = jaratok.honnan  
and (eljut.honnan,jaratok.hova)  
    NOT IN (select * from eljut));
```

- Megjegyzés: PSM-ben a **nem-lineáris rekurzió** is megengedett: **from eljut e1, eljut e2 ---*from-ban**

Oracle megoldások: with utasítással

- Az **Oracle SQL** a WITH RECURSIVE utasítást (UNION) nem támogatja, **ott másképpen** oldották meg 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_iseaf, connect_by_iscycle  
FROM jaratok  
START WITH honnan = 'SF'  
CONNECT BY NOCYCLE PRIOR hova = honnan;
```

Kérdés/Válasz

- **Köszönöm a figyelmet! Kérdés/Válasz?**
- **Vizsgára** ismerni/ismertetni kell az Eljut feladatot és az SQL-99 szabvány WITH RECURSION utasítását, továbbá feladatok lesznek Datalog program átírására SQL SELECT-re (WITH RECURSION záradékkal is).
- **Házi feladat: Gyakorlás** Oracle Példatár 3.fejezete tartalmaz hierarchikus lekérdezésekre feladatokat
- Próbáljuk ki az az Eljut-feladathoz az Oracle gépes-megoldásait, ehhez a Jaratok táblát létrehozó script: http://sila.hajas.elte.hu/AB1gy/create_jaratok_tabla.txt