Baze de date

Curs 9 – Limbajul SQL: prelucrarea datelor (2/2).

Sorina Preduț
sorina.predut@fmi.unibuc.ro
Universitatea din București

Regăsirea datelor din 2 sau mai multe tabele

- O **joncţiune** (operaţia de compunere) este o interogare care regăseşte înregistrări din 2 sau mai multe tabele. Capacitatea de a realiza o joncţiune între 2 sau mai multe tabele reprezintă una dintre cele mai puternice facilităţi ale unui sistem relaţional.
- Legătura dintre înregistrările tabelelor se realizează prin **existenţa unor câmpuri comune** caracterizate prin domenii de definiţie compatibile (**chei primare sau străine**). Realizarea unei joncţiuni se poate face precizând în clauza FROM numele tabelelor utilizate, iar în clauza WHERE criteriul de compunere.

Produsul a 2 sau mai multe tabele

- În cazul în care în interogare se specifică mai multe tabele şi nu este inclusă o clauză WHERE, interogarea va genera **produsul cartezian al tabelelor**.
  Acesta va conţine toate combinaţiile posibile de înregistrări din tabelele componente.
  Astfel, produsul cartezian a 2 tabele care conţin 100, respectiv 50 de înregistrări va avea dimensiunea de 5.000 de înregistrări.

Produsul a 2 sau mai multe tabele - cont.

- De exemplu, să considerăm tabela CATEDRA cu următoarele 4 înregistrări:

COD_CATEDRA	NUME	PROFIL
10	INFORMATICA	TEHNIC
20	ELECTRONICA	TENHIC
30	AUTOMATICA	TENHIC
40	FINANTE	ECONOMIC

Produsul a 2 sau mai multe tabele - cont.

- și tabela PROFESOR cu următoarele 7 înregistrări:

COD	NUME	PRENUME	DATA_NAST	GRAD	SEF	SALARIU	PRIMA	COD_CATEDRA
100	GHEORGHIU	STEFAN	11-AUG-46	PROF	3000	3500		10
101	MARIN	VLAD	19-APR-45	PROF	100	2500		20
102	GEORGESCU	CRISTIANA	30-OCT-51	CONF	100	2800	200	30
103	IONESCU	VERONICA		ASIST	102	1500		10
104	ALBU	GHEORGHE		LECT	100	2200	2500	20
105	VOINEA	MIRCEA	15-NOV-65	ASIST	100	1200	150	10
106	STANESCU	MARIA	05-DEC-69	ASIST	103	1200	600	20

Produsul a 2 sau mai multe tabele - cont.

- Atunci următoarea interogare va genera produsul cartezian al tabelelor, adică va avea ca rezultat 7 x 4 = 28 de rânduri ce vor conţine toate combinaţiile posibile de înregistrări din cele 2 tabele:

```
SELECT * FROM profesor, catedra;
```

Produsul a 2 sau mai multe tabele - cont.

- Dacă în lista de SELECT sunt specificate coloanele selectate, atunci numele acestora trebuie să fie unic în cadrul tuturor tabelelor.
- Dacă ∃ un nume de coloană care apare în mai mult de un tabel, atunci, pentru evitarea ambiguităţii, trebuie specificat şi tabelul din care face parte coloana în cauză.
- De exemplu, în următoarea interogare pentru coloanele cod_catedra şi nume trebuie specificate tabelele din care fac parte:

```
SELECT profesor.nume, prenume, catedra.cod_catedra, catedra.nume
FROM profesor, catedra;
```

NUME	PRENUME	COD_CATEDRA	NUME
GHEORGHIU	STEFAN	10	INFORMATICA
MARIN	VLAD	10	INFORMATICA
GEORGESCU	CRISTIANA	10	INFORMATICA
IONESCU	VERONICA	10	INFORMATICA
ALBU	GHEORGHE	10	INFORMATICA
VOINEA	MIRCEA	10	INFORMATICA
STANESCU	MARIA	10	INFORMATICA
GHEORGHIU	STEFAN	20	ELECTRONICA
MARIN	VLAD	20	ELECTRONICA
GEORGESCU	CRISTIANA	20	ELECTRONICA
IONESCU	VERONICA	20	ELECTRONICA
ALBU	GHEORGHE	20	ELECTRONICA
VOINEA	MIRCEA	20	ELECTRONICA
STANESCU	MARIA	20	ELECTRONICA
GHEORGHIU	STEFAN	30	AUTOMATICA
MARIN	VLAD	30	AUTOMATICA

GEORGESCU	CRISTIANA	30	AUTOMATICA
IONESCU	VERONICA	30	AUTOMATICA
ALBU	GHEORGHE	30	AUTOMATICA
VOINEA	MIRCEA	30	AUTOMATICA
STANESCU	MARIA	30	AUTOMATICA
GHEORGHIU	STEFAN	40	FINANTE
MARIN	VLAD	40	FINANTE
GEORGESCU	CRISTIANA	40	FINANTE
IONESCU	VERONICA	40	FINANTE
ALBU	GHEORGHE	40	FINANTE
VOINEA	MIRCEA	40	FINANTE
STANESCU	MARIA	40	FINANTE

Produsul a 2 sau mai multe tabele - cont.

- În general, pentru a scurta textul comenzii, în astfel de cazuri se folosesc de obicei **aliasuri** pentru numele tabelelor, care pot fi folosite în interogare.
- Astfel interogarea de mai înainte se mai poate scrie:

```
SELECT p.nume, prenume, c.cod_catedra, c.nume
FROM profesor p, catedra c;
```

- În general, produsul cartezian este rar folosit, având o utilitate practică redusă.

Joncţiuni

- Pentru a realiza o joncţiune între 2 sau mai multe tabele se utilizează clauza WHERE a interogărilor pe aceste tabele.
- Pentru a realiza o compunere între n tabele, va fi nevoie de cel puţin n – 1 condiţii de join.
- În funcţie de criteriul de compunere, se disting mai multe tipuri de joncţiuni:
  - **joncţiuni echivalente (EQUI - JOIN) sau joncţiuni interne (INNER JOIN),**
  - **joncţiuni neechivalente (NON EQUI - JOIN),**
  - **joncţiuni externe (OUTER JOIN),**
  - **auto-joncţiuni (SELF JOIN).**

Joncţiuni echivalente

- O **echi-joncţiune** conţine operatorul egalitate (=) în clauza WHERE, combinând înregistrările din tabele care au valori egale pentru coloanele specificate.
- De exemplu, pentru a afişa cadrele didactice şi numele catedrei din care fac parte se combină înregistrările din cele 2 tabele pentru care codul catedrei este acelaşi.

```
SELECT p.nume, p.prenume, c.nume
FROM profesor p, catedra c
WHERE p.cod_catedra = c.cod_catedra;
```

NUME	PRENUME	NUME
GHEORGHIU	STEFAN	INFORMATICA
IONESCU	VERONICA	INFORMATICA
VOINEA	MIRCEA	INFORMATICA
MARIN	VLAD	ELECTRONICA
STANESCU	MARIA	ELECTRONICA
ALBU	GHEORGHE	ELECTRONICA
GEORGESCU	CRISTIANA	AUTOMATICA

Joncţiuni neechivalente

- **Joncţiunile neechivalente** sunt acelea care nu folosesc în clauza WHERE operatorul egal. Operatorii cei mai utilizaţi în cazul joncţiunilor neechivalente sunt: <, >, <=, >= ,<>, BETWEEN ... AND ... .
- Pentru a exemplifica un astfel de tip de joncţiune considerăm tabela GRADSAL ce conţine pragul minim şi pragul maxim al salariului dintr-un anumit grad de salarizare:

GRAD_SALARIZARE	PRAG_MIN	PRAG_MAX
1	500	1500
2	1501	2000
3	2001	2500
4	2501	3500
5	3501	10000

Joncţiuni neechivalente - cont.

- Evident, între tabelele PROFESOR şi GRADSAL nu are sens definirea unei joncţiuni echivalente deoarece nu există o coloană din tabela PROFESOR căreia să-i corespundă o coloană din tabela GRADSAL.
- Exemplul următor ilustrează definirea unei joncţiuni neechivalente care evaluează gradul de salarizare a cadrelor didactice, prin încadrarea salariului acestora într-un interval stabilit de pragul minim şi pragul maxim:

Joncţiuni neechivalente - cont.

```
SELECT p.nume, p.grad, p.salariu, g.grad_salarizare
FROM profesor p, gradsal g
WHERE p.salariu BETWEEN g.prag_min AND g.prag_max;
```

NUME	GRAD	SALARIU	GRAD_SALARIZARE
IONESCU	ASIST	1500	1
VOINEA	ASIST	1200	1
STANESCU	ASIST	1200	1
MARIN	PROF	2500	3
ALBU	LECT	2200	3
GHEORGHIU	PROF	3000	4
GEORGESCU	CONF	2800	4

Joncţiuni externe

- Dacă într-o joncţiune de tipul celor prezentate până acum una sau mai multe înregistrări nu satisfac condiţia de compunere specificată în clauza WHERE, atunci ele nu vor apărea în rezultatul interogării.
  Aceste înregistrări pot apărea însă dacă se foloseşte joncţiunea externă.
- **Joncţiunea externă** returnează toate înregistrările care satisfac condiţia de joncţiune plus acele înregistrări dintr-un tabel ale căror valori din coloanele după care se face legătura nu se regăsesc în coloanele corespunzătoare ale nici unei înregistrări din celălalt tabel.

Joncţiuni externe - cont.

- Pentru a realiza o joncţiune externă între tabelele A şi B ce returnează **toate înregistrările din tabela A se utilizează semnul (+) în dreapta tabelului B.**
  Pentru fiecare înregistrare din tabela A care nu satisface condiţia de compunere pentru nici o înregistrare din tabela B se va crea în tabela B o înregistrare nulă care va fi compusă cu înregistrarea din tabela A.
- Invers, pentru a realiza o joncţiune externă între tabelele A şi B ce returnează **toate înregistrările din tabela B, se utilizează semnul (+) în dreapta tabelului A.**

Joncţiuni externe - cont.

- În interogarea utilizată pentru a exemplifica joncţiunea echivalentă, se observă că au fost selectate numai catedrele în care există cadre didactice.
  Pentru a afişa toate catedrele, indiferent dacă ele cuprind sau nu cadre didactice, se foloseşte următoarea interogare:

```
SELECT p.nume, p.prenume, c.nume
FROM profesor p, catedra c
WHERE p.cod_catedra(+) = c.cod_catedra;
```

- Se observă că ultima înregistrare (ce corespunde catedrei de finanţe care nu are în componenţă nici un cadru didactic) va avea coloanele corespunzătoare primului tabel completate cu Null.

Joncţiuni externe - cont.

NUME	PRENUME	NUME
GHEORGHIU	STEFAN	INFORMATICA
IONESCU	VERONICA	INFORMATICA
VOINEA	MIRCEA	INFORMATICA
MARIN	VLAD	ELECTRONICA
STANESCU	MARIA	ELECTRONICA
ALBU	GHEORGHE	ELECTRONICA
GEORGESCU	CRISTIANA	AUTOMATICA
		FINANTE

Joncţiuni externe - cont.

- Pentru a afişa toate cadrele didactice, inclusiv cadrele didactice a căror catedră este necunoscută, se foloseşte următoarea interogare:

```
SELECT p.nume, p.prenume, c.nume
FROM profesor p, catedra c
WHERE p.cod_catedra = c.cod_catedra(+);
```

Joncţiuni externe - cont.

- Să se afişeze cadrele didactice şi numele catedrei din care fac parte.
  Rezultatul va conţine şi catedrele care nu cuprind cadre didactice, precum şi cadrele didactice a căror catedră este necunoscută (se folosește operatorul UNION menționat în Curs1, dar și începând cu slide-ul 44 al cursului curent).

```
SELECT p.nume, p.prenume, c.nume
FROM profesor p, catedra c
WHERE p.cod_catedra = c.cod_catedra(+)
UNION
SELECT p.nume, p.prenume, c.nume
FROM profesor p, catedra c
WHERE p.cod_catedra(+) = c.cod_catedra;
```

Joncţiuni externe - cont.

- Folosirea operatorului de joncţiune externă are următoarele restricţii:
  - Operatorul (+) poate fi plasat în oricare parte a condiţiei din clauza WHERE, însă nu în ambele părţi.
    Tabelul de partea căruia este amplasat acest operator va crea înregistrări nule care vor fi compuse cu înregistrările din celălalt tabel care nu satisfac condiţia de compunere.
  - Daca tabelele A şi B au condiţii multiple de joncţiune, atunci operatorul (+) trebuie utilizat în toate aceste condiţii;
  - Într-o singură interogare nu se poate realiza o joncţiune externă a unui tabel cu mai multe tabele.

Joncţiuni externe - cont.

- O condiţie care conţine operatorul (+) nu poate fi combinată cu o altă condiţie ce utilizează operatorul IN.
- O condiţie care conţine operatorul (+) nu poate fi combinată cu o altă condiţie prin operatorul OR.

Auto-joncţiuni

- **Auto-joncţiunea** reprezintă joncţiunea unui tabel cu el însuşi.
  Pentru ca rândurile dintr-un tabel să poată fi compuse cu rânduri din acelaşi tabel, în clauza FROM a interogării numele tabelului va apărea de mai multe ori, urmat de fiecare dată de un alias.

Auto-joncţiuni - cont.

- De exemplu, pentru a selecta toate cadrele didactice care au un şef direct şi numele acestui şef se foloseşte următoarea auto-joncţiune:

```
SELECT p.nume, p.prenume, s.nume, s.prenume
FROM profesor p, profesor s
WHERE p.sef = s.cod;
```

NUME	PRENUME	NUME	PRENUME
MARIN	VLAD	GHEORGHIU	STEFAN
GEORGESCU	CRISTIANA	GHEORGHIU	STEFAN
ALBU	GHEORGHE	GHEORGHIU	STEFAN
VOINEA	MIRCEA	GHEORGHIU	STEFAN
IONESCU	VERONICA	GEORGESCU	CRISTIANA
STANESCU	MARIA	IONESCU	VERONICA

Auto-joncţiuni - cont.

- Auto-joncţiunea poate fi folosită şi pentru verificarea corectitudinii interne a datelor.
  De exemplu, este puţin probabil să existe 2 cadre didactice care au cod diferit dar în schimb au acelaşi nume, prenume şi dată de naştere.
  Pentru a verifica dacă există astfel de înregistrări se foloseşte interogarea:

```
SELECT a.nume, a.prenume
FROM profesor a, profesor b
WHERE    a.nume = b.nume AND
         a.prenume = b.prenume AND
         a.data_nast = b.data_nast AND
         a.cod <> b.cod;
```

Joncţiuni - standard SQL3

- În sistemul Oracle11g, joncțiunea se poate implementa şi prin intermediul unei sintaxe specifice acesteia, introdusă de **standardul SQL3**.
  Faţă de compunerea realizată prin specificarea condiţiei în clauza WHERE, această sintaxă nu aduce beneficii în privinţa performanţei.
  Conform standardului SQL3, tipurile de compunere descrise anterior se pot rescrie, echivalent, prin cuvintele cheie:
  - **NATURAL JOIN**,
  - **FULL OUTER JOIN**,
  - **JOIN cu clauzele USING şi ON**.

Joncţiuni - cont.

- **Sintaxa corespunzătoare standardului SQL3** este următoarea:

```
SELECT tabel_1.nume_coloană, tabel_2.nume_coloană
FROM tabel_1
[NATURAL JOIN tabel_2]
| [JOIN tabel_2 USING (nume_coloană) ]
| [JOIN tabel_2 ON (tabel_1.nume_coloană = tabel_2.nume_coloană) ] |
[LEFT | RIGHT | FULL OUTER JOIN tabel_2
ON (tabel_1.nume_coloană = tabel_2.nume_coloană) ];
```

- **Utilizând aceste modalităţi de specificare a joncțiunii, clauza FROM înlocuieşte clauzele FROM şi WHERE din exemplele anterioare**.
- În lista SELECT sunt enumerate coloanele, calificate cu numele tabelelor din care provin, ale căror date sunt regăsite.

NATURAL JOIN

- Joncțiunea poate fi implementată cu ajutorul clauzei NATURAL JOIN în cazul în care **coloanele pe baza cărora se realizează compunerea au acelaşi nume în ambele tabele și același tip de date**.
- Clauza determină selectarea liniilor din cele 2 tabele, care au valori egale în toate coloanele ale căror nume coincid.
- Dacă tipurile de date ale coloanelor cu nume identice sunt diferite, va fi returnată o eroare.
- **Coloanele având acelaşi nume în cele 2 tabele trebuie să nu conţină calificatori** (să nu fie precedate de numele sau alias-ul tabelului corespunzător).

JOIN ... USING

- Pentru a putea preciza în mod explicit coloanele pe baza cărora are loc compunerea, se poate utiliza clauza **JOIN tabel_2 USING nume_coloană**.
- Aceasta efectuează un EQUI-JOIN pe baza coloanei cu numele specificat în sintaxă.
- Clauza este utilă atunci când ∃ coloane având acelaşi nume, dar care nu trebuie să intre în condiţia de join.
- **Coloanele referite în clauza USING nu pot conţine calificatori în nicio apariţie a lor în instrucţiunea SQL**.
- Clauzele NATURAL JOIN şi USING nu pot exista simultan în aceeaşi instrucţiune SQL.

JOIN ... ON

- Pentru a delimita condiţiile de join şi cele de căutare sau filtrare, este utilă clauza **JOIN tabel_2 ON tabel_1.nume_coloană = tabel_2.nume_coloană**, care efectuează un EQUI-JOIN pe baza condiţiei exprimate în opţiunea ON.
- **Condiţiile de căutare sau filtrare vor apărea distinct în clauza WHERE**.

Exemple

- Să se afişeze cadrele didactice şi numele catedrei din care fac parte pentru profesorii având codurile 105 și 106.
  Se vor funiza 3 variante de rezolvare, utilizând sintaxa specifică standardului SQL3.

```
SELECT profesor.nume, prenume, catedra.nume
FROM profesor
NATURAL JOIN catedra
WHERE cod IN (105, 106);
```

Exemple - cont.

```
SELECT profesor.nume, prenume, catedra.nume
FROM profesor
JOIN catedra USING (cod_catedra)
WHERE cod IN (105, 106);
```

```
SELECT p.nume, prenume, c.nume
FROM profesor p
JOIN catedra c ON (p.cod_catedra = c.cod_catedra)
WHERE cod IN (105, 106);
```

Exemple - cont.

- Condiţiile suplimentare ale unei interogări pot fi specificate atât în clauza WHERE, cât şi în clauza ON.
- Exemplu. Să se afişeze cadrele didactice şi numele catedrei din care fac parte pentru profesorii care au fost angajații în cursul lunii trecute (a se vedea Curs3, slide-ul 81 pt. schema completă).

```
SELECT p.nume, prenume, c.nume
FROM profesor p
JOIN catedra c ON (p.cod_catedra = c.cod_catedra)
     AND TO_CHAR(p.data_angajarii, 'MM/YYYY') =
     TO_CHAR(ADD_MONTHS(SYSDATE, -1), 'MM/YYYY');
```

- Instrucţiunea anterioară este corectă şi este echivalentă cu comanda obţinută înlocuind operatorul AND cu clauza WHERE.

OUTER JOIN ... ON

- Prin intermediul clauzelor **LEFT, RIGHT şi FULL OUTER JOIN tabel_2 ON (tabel_1.nume_coloană = tabel_2.nume_coloană)** se efectuează compunerea externă la stânga, dreapta, respectiv compunerea externă completă pe baza condiţiei exprimate în clauza ON.
- Joncțiunea (compunerea) externă completă (FULL OUTER JOIN) returnează rezultatele unei joncțiuni interne (INNER JOIN), dar şi cele ale joncțiunilor externe la stânga şi la dreapta.

Exemple

- Exemplul din slide-ul 19 al cursului curent se poate rezolva în standardul SQL3 astfel: (Pentru a afişa toate cadrele didactice, indiferent dacă aparțin unei catedre.)

```
SELECT p.nume, p.prenume, c.nume
FROM profesor p
LEFT OUTER JOIN catedra c ON (p.cod_catedra = c.cod_catedra);
```

Exemple - cont.

- Exemplul din slide-ul 17 al cursului curent se poate rezolva în standardul SQL3 astfel: (Pentru a afişa toate catedrele, indiferent dacă ele cuprind sau nu cadre didactice)

```
SELECT p.nume, p.prenume, c.nume
FROM profesor p
RIGHT OUTER JOIN catedra c ON (p.cod_catedra = c.cod_catedra);
```

Exemple - cont.

- Exemplul din slide-ul 20 al cursului curent se poate rezolva în standardul SQL3 astfel:
  (Să se afişeze cadrele didactice şi numele catedrei din care fac parte.
  Rezultatul va conţine şi catedrele care nu cuprind cadre didactice, precum şi cadrele didactice a căror catedră este necunoscută)

```
SELECT p.nume, p.prenume, c.nume
FROM profesor p
FULL OUTER JOIN catedra c ON (p.cod_catedra = c.cod_catedra);
```

Produsul cartezian

- Joncțiunea este o submulţime a unei combinaţii mai generale de 2 tabele, şi anume produsul cartezian.
- De obicei, produsul cartezian generează un nr. mare de linii şi constituie o operaţie puţin utilă în practică.
- Un caz de utilizare a acestei operaţii apare atunci când este necesară obţinerea unui volum mare de date utile pentru anumite teste de simulare.

Produsul cartezian

- Similar aplicării sale în cazul mulţimilor, produsul cartezian a 2 tabele este un alt tabel care conţine toate perechile posibile de linii din cele două tabele.
  Coloanele tabelului produs sunt toate coloanele din primul tabel, urmate de toate coloanele din cel de-al doilea.
- Dacă se specifică o interogare asupra a 2 tabele, fără o clauză WHERE, limbajul SQL va produce drept rezultat al interogării produsul cartezian al acestora.

Produsul cartezian

- Pentru produsul cartezian, standardul SQL3 prezintă un format special al instrucţiunii SELECT:

```
SELECT tabel_1.nume_coloană, tabel_2.nume_coloană
FROM tabel_1 CROSS JOIN tabel_2;
```

- Exemplul din slide-ul 6 al cursului curent se poate rezolva în standardul SQL3 astfel:

```
SELECT *
FROM profesor CROSS JOIN catedra;
```

SELECT - generare rezultate joncțiune

- Procedura de generare a rezultatelor unei instrucţiuni SELECT referitoare la o joncțiune este următoarea:
  1. formarea produsului cartezian al tabelelor enumerate în clauza FROM;
  2. dacă ∃ o clauză WHERE, condiţia de căutare se aplică fiecărui rând al tabelului produs, fiind reţinute cele care satisfac condiţia;
  3. pentru fiecare linie rămasă, se determină valoarea fiecărui element din lista SELECT, pentru a produce o singură linie în tabelul rezultant;
  4. dacă a fost specificat cuvântul cheie DISTINCT, se elimină liniile duplicat din tabelul rezultant;
  5. dacă există o clauză ORDER BY, se sortează tabelul rezultant conform criteriilor specificate.

SELECT - generare rezultate joncțiune - cont.

- În termeni de algebră relaţională, pasul 2 descris anterior creează o restricţie a produsului cartezian, iar etapele 3 şi 4 sunt echivalente unei proiecţii a restricţiei asupra coloanelor menţionate în lista SELECT.

Operatori pentru mulţimi

- Operatorii de mulţimi combină 2 sau mai multe interogări, efectuând operaţii specifice mulţimilor: reuniune, intersecţie, diferenţă.
- Aceşti operatori se mai numesc şi operatori verticali deoarece combinarea celor 2 interogări se face coloană cu coloană.
  Din acest motiv, nr. total de coloane şi tipurile de date ale coloanelor corespondente din cele 2 interogări trebuie să coincidă.

Operatori pentru mulţimi - cont.

- Există următorii operatori pentru mulţimi:
  1. **UNION** – Returnează rezultatele a 2 sau mai multe interogări eliminând toate înregistrările duplicat;
  2. **UNION ALL** – Returnează rezultatele a 2 sau mai multe interogări incluzând înregistrările duplicat;
  3. **INTERSECT** – Returnează toate înregistrările distincte găsite în ambele interogări;
  4. **MINUS** – Returnează toate înregistrările distincte care se găsesc în prima interogare dar nu şi în a doua interogare.

Operatori pentru mulţimi - cont.

- Să considerăm, de exemplu, următoarele interogări:

```
SELECT grad, salariu
FROM profesor
WHERE cod_catedra = 10;
```

GRAD	SALARIU
PROF	3000
ASIST	1500
ASIST	1200

```
SELECT grad, salariu
FROM profesor
WHERE cod_catedra = 10;
```

GRAD	SALARIU
PROF	2500
LECT	2200
ASIST	1200

Operatori pentru mulţimi - cont.

- În continuare exemplificăm fiecare dintre operatorii pentru mulţimi aplicaţi acestor interogări:

- ```
  SELECT grad, salariu
  FROM profesor
  WHERE cod_catedra = 10
  UNION
  SELECT grad, salariu
  FROM profesor
  WHERE cod_catedra = 20;
  ```

GRAD	SALARIU
ASIST	1200
ASIST	1500
LECT	2200
PROF	2500
PROF	3000
  ```

Operatori pentru mulţimi - cont.

```
SELECT grad, salariu
FROM profesor
WHERE cod_catedra = 10
UNION ALL
SELECT grad, salariu
FROM profesor
WHERE cod_catedra = 20;
```

GRAD	SALARIU
PROF	3000
ASIST	1500
ASIST	1200
PROF	2500
LECT	2200
ASIST	1200

Operatori pentru mulţimi - cont.

➢
```
SELECT grad, salariu
FROM profesor
WHERE cod_catedra = 10
INTERSECT
SELECT grad, salariu
FROM profesor
WHERE cod_catedra = 20;
```

GRAD	SALARIU
ASIST	1200

Operatori pentru mulţimi - cont.

```
SELECT grad, salariu
FROM profesor
WHERE cod_catedra = 10
MINUS
SELECT grad, salariu
FROM profesor
WHERE cod_catedra = 20;
```

GRAD	SALARIU
ASIST	1500
PROF	3000

Operatori pentru mulţimi - cont.

- Există următoarele reguli de folosire a operatorilor pentru mulţimi:
  - interogările trebuie să conţină acelaşi număr de coloane;
  - coloanele corespondente trebuie să aibă acelaşi tip de dată;
  - în rezultat vor apărea numele coloanelor din prima interogare, nu cele din a doua interogare chiar dacă aceasta foloseşte alias-uri; de exemplu:

```
SELECT cod FROM profesor
MINUS
SELECT sef FROM profesor;
```

```
COD
---------
      101
      104
      105
      106
```

Operatori pentru mulţimi - cont.

- clauza ORDER BY poate fi folosită o singură dată într-o interogare care foloseşte operatori de mulţimi; atunci când se foloseşte, ea trebuie poziţionată la sfârşitul comenzii; de exemplu:

```
SELECT grad, salariu
FROM profesor
WHERE cod_catedra = 10
UNION
SELECT grad, salariu
FROM profesor
WHERE cod_catedra = 20
ORDER BY 2;
```

GRAD	SALARIU
ASIST	1200
ASIST	1500
LECT	2200
PROF	2500
PROF	3000

Operatori pentru mulţimi - cont.

- operatorii pentru mulţimi pot fi utilizaţi în subinterogări;
- pentru a modifica ordinea de execuţie este posibilă utilizarea parantezelor; de exemplu:

```
SELECT grad FROM profesor WHERE cod_catedra = 10
INTERSECT
SELECT grad FROM profesor WHERE cod_catedra = 20
UNION
SELECT grad FROM profesor WHERE cod_catedra = 30;
```

GRAD

ASIST
CONF
PROF

Operatori pentru mulţimi - cont.

```
SELECT grad FROM profesor WHERE cod_catedra = 10
INTERSECT
(SELECT grad FROM profesor WHERE cod_catedra = 20
UNION
SELECT grad FROM profesor WHERE cod_catedra = 30 );
```

GRAD

ASIST
PROF

Subinterogări şi operatorii ANY, ALL, EXISTS

- O **subinterogare** este o comandă SELECT inclusă în altă comandă SELECT.
- Rezultatele subinterogării sunt transmise celeilalte interogări şi pot apărea în cadrul clauzelor WHERE, HAVING sau FROM.
- Subinterogările sunt utile pentru a scrie interogări bazate pe o condiţie în care valoarea de comparaţie este necunoscută.
  Această valoare poate fi aflată folosind o subinterogare.
  De exemplu:

```
SELECT coloane
FROM tabel
WHERE coloana = (SELECT coloane
                 FROM tabel WHERE condiţie)
```

Subinterogări - cont.

- Subinterogarea, denumită şi **interogare interioară (inner query)**, generează valorile pentru condiţia de căutare a instrucţiunii SELECT care o conţine, denumită **interogare exterioară (outer query)**.
- Instrucţiunea SELECT exterioară depinde de valorile generate de către interogarea interioară.
- În general, interogarea interioară se execută prima şi rezultatul acesteia este utilizat în interogarea exterioară.
- Rezultatul interogării exterioare depinde de numărul valorilor returnate de către interogarea interioară. În acest sens, putem distinge:
  - **Subinterogări care returnează un singur rând;**
  - **Subinterogări care returnează mai multe rânduri.**

Subinterogări - cont.

➢ Dpdv al ordinii de evaluare a interogărilor putem clasifica subinterogările în:

1. **Subinterogări simple/nesincronizate** – în care interogarea interioară este evaluată prima, independent de interogarea exterioară (interogarea interioară se execută o singură dată);
2. **Subinterogări corelate/sincronizate** – în care valorile returnate de interogarea interioară depind de valorile returnate de interogarea exterioară (interogarea interioară este evaluată pentru fiecare înregistrare a interogării exterioare).

Subinterogări - cont.

- Subinterogările sunt îndeosebi utilizate atunci când se doreşte ca o interogare să regăsească înregistrări dintr-o tabelă care îndeplinesc o condiţie ce depinde la rândul ei de valori din aceeaşi tabelă.
- **Observaţie**: Clauza ORDER BY nu poate fi utilizată într-o subinterogare.
  Regula este că poate exista doar o singură clauză ORDER BY pentru o declaraţie SELECT şi, dacă este specificată, trebuie să fie ultima clauză din comanda SELECT.
  Prin urmare, clauza ORDER BY nu poate fi specificată decât în interogarea cea mai din exterior.

Subinterogări care returnează un singur rând

- În acest caz condiţia, din clauza WHERE sau HAVING a interogării exterioare utilizează operatorii: =, <, <=, >, >=, <> care operează asupra unei subinterogări ce returnează o singură valoare.
- Interogarea interioară poate conţine condiţii complexe formate prin utilizarea condiţiilor multiple de interogare cu ajutorul operatorilor AND şi OR sau prin utilizarea funcţiilor agregat.

Subinterogări care returnează un singur rând

- Următoarea interogare selectează cadrele didactice care au salariul minim.
  Salariul minim este determinat de o subinterogare ce returnează o singură valoare.

```
SELECT nume, prenume, salariu
FROM profesor
WHERE salariu = (SELECT MIN(salariu)
                 FROM profesor);
```

NUME	PRENUME	SALARIU
VOINEA	MIRCEA	1200
STANESCU	MARIA	1200

Subinterogări care returnează un singur rând

- Procesul de evaluare al acestei interogări se desfăşoară astfel:
  - se evaluează în primul rând interogarea interioară:
    Valoarea obţinută este `MIN(salariu) = 1200`
  - rezultatul evaluării interogării interioare devine condiţie de căutare pentru interogarea exterioară şi anume:

    ```
    SELECT nume, prenume, salariu FROM profesor WHERE salariu
    =1200;
    ```
- În cazul în care interogarea interioară nu întoarce nici o înregistrare, interogarea exterioară nu va selecta la rândul ei nici o înregistrare.

Subinterogări care returnează un singur rând

- **Observaţie**: Dacă care se utilizează operatorii: =, <, <=, >, >=, <> în condiţia interogării exterioare, atunci interogarea interioară trebuie în mod obligatoriu să returneze o singură valoare.
  În caz contrar va apărea un mesaj de eroare, ca în exemplul următor:

```
SELECT nume, prenume, salariu
FROM profesor
WHERE salariu = (SELECT MIN (salariu)
                 FROM profesor
                 GROUP BY grad);
ERROR:
ORA-01427: single-row subquery returns more than one row
```

Subinterogări care returnează un singur rând

- Subinterogările pot fi folosite nu numai în clauza WHERE a interogării exterioare, ci şi în clauza HAVING.
  Următoarea interogare afişează toate gradele didactice pentru care salariul minim este mai mare decât salariul mediu al tuturor cadrelor didactice.

```
SELECT grad
FROM profesor
GROUP BY grad
HAVING min(salariu) > (SELECT avg(salariu)
                       FROM profesor);
```

```
GRAD
-----
CONF
LECT
PROF
```

Subinterog. care returnează mai multe rânduri

- În cazul când interogarea întoarce mai multe rânduri **nu mai este posibilă folosirea operatorilor de comparaţie**.
  În locul acestora **se foloseşte operatorul IN**, care aşteaptă o listă de valori şi nu doar una.
- Următoarea interogare selectează pentru fiecare grad didactic acele persoane care au salariul minim.
  Salariul minim pentru fiecare grad didactic este aflat printr-o subinterogare, care, evident, va întoarce mai multe rânduri:

Subinterog. care returnează mai multe rânduri

```
SELECT nume, salariu, grad
FROM profesor
WHERE (salariu, grad) IN (SELECT MIN (salariu), grad
                          FROM profesor
                          GROUP BY grad)
ORDER BY salariu;
```

NUME	SALARIU	GRAD
VOINEA	1200	ASIST
STANESCU	1200	ASIST
ALBU	2200	LECT
MARIN	2500	PROF
GEORGESCU	2800	CONF

Subinterog. care returnează mai multe rânduri

- **Observaţie**: Spre deosebire de celelalte interogări de până acum, interogarea de mai înainte compară perechi de coloane.
  În acest caz trebuie respectate următoarele reguli:
  - coloanele din dreapta condiţiei de căutare sunt în paranteze şi fiecare coloană este separată prin virgulă;
  - coloanele returnate de interogarea interioară trebuie să se potrivească ca număr şi tip cu coloanele cu care sunt comparate în interogarea exterioară; în plus, ele trebuie să fie în aceeaşi ordine cu coloanele cu care sunt comparate.

Subinterog. care returnează mai multe rânduri

- Alături de operatorul IN, o subinterogare care returnează mai multe rânduri poate folosi **operatorii ANY, ALL sau EXISTS.**
  Operatorii ANY şi ALL sunt prezentaţi în continuare, iar operatorul EXISTS va fi prezentat în secţiunea “Subinterogări corelate”.
- Operatorii ANY şi ALL sunt folosiţi în mod obligatoriu în combinaţie cu operatorii relaţionali =, !=, <, >, <=, >=;
- Operatorii IN şi EXISTS nu pot fi folosiţi în combinaţie cu operatorii relaţionali, dar pot fi utilizaţi cu operatorul NOT, pentru negarea expresiei.

Operatorul ANY

- Operatorul ANY (**sau sinonimul său SOME**) este folosit pentru a compara o valoare cu oricare dintre valorile returnate de o subinterogare.
  Pentru a înţelege modul de folosire al acestui operator să considerăm următorul exemplu ce afişează cadrele didactice ce câştigă mai mult decât profesorii care au cel mai mic salariu:

```
SELECT nume, salariu, grad
FROM profesor
WHERE salariu > ANY (SELECT DISTINCT salariu
                     FROM profesor
                     WHERE grad='PROF');
```

NUME	SALARIU	GRAD
GHEORGHIU	3000	PROF
GEORGESCU	2800	CONF

Operatorul ANY - cont.

- Interogarea anterioar[este evaluată astfel: dacă salariul unui cadru didactic este mai mare decât cel puţin unul din salariile returnate de interogarea interioară, acea înregistrare este inclusă în rezultat.
  Cu alte cuvinte,
  >ANY înseamnă mai mare decât minimul dintre valorile returnate de interogarea interioară,
  <ANY înseamnă mai mic ca maximul, iar
  =ANY este echivalent cu operatorul IN.
- **Observaţie**: opţiunea DISTINCT este folosită frecvent atunci când se foloseşte operatorul ANY pentru a preveni selectarea de mai multe ori a unor înregistrări.

Operatorul ALL

- Operatorul ALL este folosit pentru a compara o valoare cu toate valorile returnate de o subinterogare.
- Considerăm următorul exemplu ce afişează cadrele didactice care câştigă mai mult decât asistenţii cu salariul cel mai mare:

```
SELECT nume, salariu, grad
FROM profesor
WHERE salariu > ALL (SELECT DISTINCT salariu
                     FROM profesor
                     WHERE grad='ASIST');
```

NUME	SALARIU	GRAD
GHEORGHIU	3000	PROF
MARIN	2500	PROF
GEORGESCU	2800	CONF
ALBU	2200	LECT

Operatorul ALL - cont.

- Interogarea de mai sus este evaluată astfel: dacă salariul unui cadru didactic este mai mare decât toate valorile returnate de interogarea interioară, acea înregistrare este inclusă în rezultat.
  Cu alte cuvinte,
  >ALL înseamnă mai mare ca maximul dintre valorile returnate de interogarea interioară, iar <ALL înseamnă mai mic ca minimul dintre acestea.
- **Observaţie**: operatorul ALL nu poate fi utilizat cu operatorul = deoarece interogarea nu va întoarce nici un rezultat cu excepţia cazului în care toate valorile sunt egale, situaţie care nu ar avea sens.

Operatorul ALL - cont.

- Subinterogările pot fi imbricate (utilizate cu alte subinterogări) până la 255 de nivele, indiferent de nr. de valori returnate de fiecare subinterogare.
- Pentru a selecta cadrele didactice care au salariul mai mare decât cel mai mare salariu al cadrelor didactice care aparţin catedrei de Electronică, vom folosi următoarea interogare:

```
SELECT nume, prenume, salariu
FROM profesor
WHERE salariu > (SELECT MAX(salariu)
                 FROM profesor
                 WHERE cod_catedra = (SELECT cod_catedra
                                      FROM catedra
                                      WHERE nume='ELECTRONICĂ'));
```

NUME	PRENUME	SALARIU
GHEORGHIU	STEFAN	3000
GEORGESCU	CRISTIANA	2800

Subinterogări corelate

- **În exemplele considerate până acum** interogarea interioară era evaluată prima, după care valoarea sau valorile rezultate erau utilizate de către interogarea exterioară. Subinterogările de acest tip sunt numite **subinterogări simple**.
- O altă formă de subinterogare o reprezintă **interogarea corelată**, caz în care interogarea exterioară transmite repetat câte o înregistrare pentru interogarea interioară.
- Interogarea interioară este evaluată de fiecare dată când este transmisă o înregistrare din interogarea exterioară, care se mai numeşte şi înregistrare candidată.
- Subinterogarea corelată poate fi identificată prin faptul că interogarea interioară nu se poate executa independent ci depinde de valoarea transmisă de către interogarea exterioară.
- Dacă ambele interogări accesează aceeaşi tabelă, trebuie asigurate alias-uri pentru fiecare referire la tabela respectivă.

Subinterogări corelate - cont.

- Subinterogările corelate reprezintă o cale de a accesa fiecare înregistrare din tabel şi de a compara anumite valori ale acesteia cu valori ce depind tot de ea.
- **Evaluarea unei subinterogări corelate** se execută în următorii paşi:
  1. Interogarea exterioară trimite o înregistrare candidată către interogarea interioară;
  2. Interogarea interioară se execută în funcţie de valorile înregistrării candidate;
  3. Valorile rezultate din interogarea interioară sunt utilizate pentru a determina dacă înregistrarea candidată va fi sau nu inclusă în rezultat;
  4. Se repetă procedeul începând cu pasul 1 până când nu mai există înregistrări candidate.

Subinterogări corelate - cont.

- De exemplu, pentru a regăsi cadrele didactice care câştigă mai mult decât salariul mediu din propria catedră, putem folosi următoarea interogare corelată:

```
SELECT nume, prenume, salariu
FROM profesor p
WHERE salariu > (SELECT AVG(salariu)
                 FROM profesor
                 WHERE cod_catedra = p.cod_catedra);
```

NUME	PRENUME	SALARIU
GHEORGHIU	STEFAN	3000
MARIN	VLAD	2500
ALBU	GHEORGHE	2200

Subinterogări corelate - cont.

- În exemplul anterior coloana interogării exterioare care se foloseşte în interogarea interioară este p.cod_catedra.
  Deoarece p.cod_catedra poate avea valoare diferită pentru fiecare înregistrare, interogarea interioară se execută pentru fiecare înregistrare candidată transmisă de interogarea exterioară.

Subinterogări corelate - cont.

- Atunci când folosim subinterogări corelate împreună cu clauza HAVING, coloanele utilizate în această clauză trebuie să se regăsească în clauza GROUP BY.
  În caz contrar, va fi generat un mesaj de eroare datorat faptului că nu se poate face comparaţie decât cu o expresie de grup.
- De exemplu, următoarea interogare este corectă, ea selectând gradele didactice pentru care media salariului este mai mare decât maximul primei pentru acelaşi grad:

```
SELECT grad
FROM profesor p
GROUP BY grad
HAVING AVG(salariu) > (SELECT MAX(prima)
                       FROM profesor
                       WHERE grad = p.grad);
```

Operatorul EXISTS

- Operatorul EXISTS verifică dacă, pentru fiecare înregistrare transmisă de interogarea exterioară, există sau nu înregistrări care satisfac condiţia interogării interioare, returnând interogării exterioare valoarea TRUE sau FALSE.
- Cu alte cuvinte, operatorul EXISTS cere în mod obligatoriu corelarea interogării interioare cu interogarea exterioară.
- Datorită faptului că operatorul EXISTS verifică doar existenţa rândurilor selectate şi nu ia în considerare nr. sau valorile atributelor selectate, **în subinterogare** pot fi specificate orice număr de atribute (deşi acest lucru nu este recomandabil din punct de vedere al eficienţei).
- În particular, **poate fi folosită o constantă şi chiar simbolul '*'** .
  De altfel EXISTS este singurul operator care permite acest lucru.

Operatorul EXISTS - cont.

- Următoarea interogare selectează toate cadrele didactice care au **măcar** un subordonat:

```
SELECT cod, nume, prenume, grad
FROM profesor p
WHERE EXISTS (SELECT '1'
              FROM profesor
              WHERE profesor.sef = p.cod)
ORDER BY cod;
```

COD	NUME	PRENUME	GRAD
100	GHEORGHIU	STEFAN	PROF
102	GEORGESCU	CRISTIANA	CONF
103	IONESCU	VERONICA	ASIST

Operatorul EXISTS - cont.

- La fel ca şi operatorul IN, operatorul EXISTS poate fi negat, luând forma **NOT EXISTS**. Totuşi, o remarcă foarte importantă este faptul că pentru subinterogări **NOT IN nu este la fel de eficient ca NOT EXISTS**.
  Astfel dacă în lista de valori transmisă operatorului NOT IN există una sau mai multe valori NULL, atunci condiţia va lua valoarea de adevăr FALSE, indiferent de celelalte valori din listă.
- De exemplu, următoarea interogare încearcă să returneze toate cadrele didactice care nu au nici un subaltern:

```
SELECT nume, grad
FROM profesor
WHERE cod NOT IN (SELECT sef FROM profesor);
```

Operatorul EXISTS - cont.

- Interogarea anterioară nu va întoarce nici o înregistrare deoarece coloana sef conţine şi valoarea NULL.
  Pentru a obţine rezultatul corect trebuie să folosim următoarea interogare:

```
SELECT nume, grad
FROM profesor p
WHERE NOT EXISTS (SELECT sef
                  FROM profesor
                  WHERE sef=p.cod);
```

NUME	GRAD
MARIN	PROF
ALBU	LECT
VOINEA	ASIST
STANESCU	ASIST

Operatorul EXISTS - cont.

- În general, operatorul EXISTS se foloseşte în cazul subinterogărilor corelate şi este câteodată cel mai eficient mod de a realiza anumite interogări.
- Performanţa interogărilor depinde de folosirea indecşilor, de numărul rândurilor returnate, de dimensiunea tabelei şi de necesitatea creării tabelelor temporare pentru evaluarea rezultatelor intermediare.
  Tabelele temporare generate de Oracle nu sunt indexate, iar acest lucru poate degrada performanţa subinterogărilor dacă se folosesc operatorii IN, ANY sau ALL.

Operatorul EXISTS - cont.

- Alte exemple ce folosesc operatorul [NOT] EXISTS, inclusiv **operația de diviziune** menționată în Curs1, precum și
  clauza WITH
  se găsesc la:
  https://drive.google.com/file/d/1Z0W0fmIScrDPB5IgqvnkKYylMwmK3OHk/view?usp=sharing

Subinterogări - cont.

- Subinterogările mai pot apărea şi în alte comenzi SQL cum ar fi: UPDATE, DELETE, INSERT şi CREATE TABLE.
- Aşa cum am văzut, există în principal 2 moduri de realizare a interogărilor ce folosesc date din mai multe tabele: joncţiuni şi subinterogări.
- **Joncţiunile reprezintă forma de interogare relaţională** (în care sarcina găsirii drumului de acces la informaţie revine SGBD-ului), iar **subinterogările forma procedurală** (în care trebuie indicat drumul de acces la informaţie).
  Fiecare dintre aceste forme are avantajele sale, depinzând de cazul specific în care se aplică.

Operaţii pe tabele ce conţin informaţii de structură arborescentă

- O bază de date relaţională nu poate stoca înregistrări în mod ierarhic, dar la nivelul înregistrării pot exista informaţii care determină o relaţie ierarhică între înregistrări.
- SQL permite afişarea rândurilor dintr-o tabelă ţinând cont de relaţiile ierarhice care apar între rândurile tabelei.
- Parcurgerea în mod ierarhic a informaţiilor se poate face doar la nivelul unei singure tabele.
- Operaţia se realizează cu ajutorul clauzelor **START WITH** şi **CONNECT BY** din comanda SELECT (**Cereri ierarhice**).

Cereri ierarhice

- De exemplu, în tabela PROFESOR există o relaţie ierarhică între înregistrări datorată valorilor din coloanele cod şi sef.
  Fiecare înregistrare aferentă unui cadru didactic conţine în coloana sef codul persoanei căreia îi este direct subordonat.
  Pentru a obţine o situaţie ce conţine nivelele ierarhice, vom folosi următoarea interogare:

```
SELECT LEVEL, nume, prenume, grad
FROM profesor
START WITH sef IS NULL
CONNECT BY PRIOR cod = sef;
```

LEVEL	NUME	PRENUME	GRAD
1	GHEORGHIU	STEFAN	PROF
2	MARIN	VLAD	PROF
2	GEORGESCU	CRISTIANA	CONF
3	IONESCU	VERONICA	ASIST
4	STANESCU	MARIA	ASIST
2	ALBU	GHEORGHE	LECT
2	VOINEA	MIRCEA	ASIST

Cereri ierarhice - cont.

- **Explicarea sintaxei şi a regulilor de funcţionare pentru exemplul anterior**:
  - Clauza standard SELECT poate conţine pseudo-coloana **LEVEL** ce indică nivelul înregistrării în arbore (cât de departe este de nodul rădăcină).
    Astfel, nodul rădăcină are nivelul 1, fiii acestuia au nivelul 2, ş.a.m.d.;
  - În clauza FROM nu se poate specifica decât o tabelă;
  - Clauza WHERE poate apărea în interogare pentru a restricţiona vizitarea nodurilor (înregistrărilor) din cadrul arborelui;

Cereri ierarhice - cont.

- Clauza CONNECT BY specifică coloanele prin care se realizează relaţia ierarhică; acesta este clauza cea mai importantă pentru parcurgerea arborelui şi este obligatorie;
- Operatorul PRIOR stabileşte direcţia în care este parcurs arborele.
  Dacă clauza apare înainte de atributul cod arborele este parcurs de sus în jos, iar dacă apare înainte de atributul sef arborele este parcurs de jos în sus;
- Clauza START WITH specifică nodul (înregistrarea) de început al arborelui.
  Ca punct de start nu se poate specifica un anumit nivel (LEVEL), ci trebuie specificată valoarea; această clauză este opţională, dacă ea lipseşte, pentru fiecare înregistrare se va parcurge arborele care are ca rădăcină această înregistrare.

Cereri ierarhice - cont.

- În sintaxa interogării anterioare, pentru a ordona înregistrările returnate, poate apărea clauza ORDER BY, dar este recomandabil să nu o folosim deoarece ordinea implicită de parcurgere a arborelui va fi distrusă.
- **Pentru a elimina doar un anumit nod din arbore putem folosi clauza WHERE, iar pentru a elimina o întreagă ramură dintr-un arbore (o anumită înregistrare împreună cu fiii acesteia) folosim o condiţie compusă în clauza CONNECT BY**.

Cereri ierarhice - cont.

- Următorul exemplu elimină doar înregistrarea cu numele 'GEORGESCU', dar nu şi copiii acesteia:

```
SELECT LEVEL, nume, prenume, grad
FROM profesor
WHERE nume != 'GEORGESCU'
START WITH sef IS NULL
CONNECT BY PRIOR cod = sef;
```

LEVEL	NUME	PRENUME	GRAD
1	GHEORGHIU	STEFAN	PROF
2	MARIN	VLAD	PROF
3	IONESCU	VERONICA	ASIST
4	STANESCU	MARIA	ASIST
2	ALBU	GHEORGHE	LECT
2	VOINEA	MIRCEA	ASIST

Cereri ierarhice - cont.

- Pentru a elimina toată ramura care conţine înregistrarea cu numele 'GEORGESCU' şi înregistrările pentru subordonaţii acesteia se foloseşte următoarea interogare:

```
SELECT LEVEL, nume, prenume, grad
FROM profesor
START WITH sef IS NULL
CONNECT BY PRIOR cod = sef AND nume != 'GEORGESCU' ;
```

LEVEL	NUME	PRENUME	GRAD
1	GHEORGHIU	STEFAN	PROF
2	MARIN	VLAD	PROF
2	ALBU	GHEORGHE	LECT
2	VOINEA	MIRCEA	ASIST

Comanda INSERT

- Această comandă este utilizată pentru adăugarea unor rânduri noi într-o tabelă creată anterior sau în tabelele de bază ale unei vederi.
- Comanda INSERT poate fi utilizată în 2 moduri:
  1. Pentru introducerea datelor într-un tabel, câte o înregistrare la un moment dat.
  În acest caz sintaxa este următoarea:

```
INSERT INTO tabela [(coloana1, coloana 2, ....)]
VALUES (valoare1, valoare2, ...);
```

Comanda INSERT - cont.

- În momentul inserării datelor, trebuie respectate următoarele reguli:
  - coloanele pot fi specificate în orice ordine, însă trebuie asigurată corespondenţa între coloane şi valorile furnizate (coloanei 1 îi corespunde valoarea 1, coloanei 2 îi corespunde valoarea 2, ş.a.m.d.) iar coloanelor nespecificate le va fi ataşată valoarea Null;
  - în cazul în care coloanele nu sunt specificate explicit, se impune ca ordinea în care apar valorile în comanda INSERT să coincidă cu cea în care coloanele au fost definite la crearea tabelei (dacă nu se cunoaşte ordinea de declarare a coloanelor se poate folosi comanda SQL* Plus DESCRIBE nume_tabela care va afişa lista coloanelor definite pentru tabela respectivă, tipul, lungimea şi restricţiile de integritate);

Comanda INSERT - cont.

- valorile trebuie să aibă acelaşi tip de dată ca şi câmpurile în care sunt adăugate;
- dimensiunea valorilor introduse trebuie să fie mai mică sau cel mult egală cu dimensiunea coloanei (un şir de 20 de caractere nu poate fi adăugat într-o coloană cu dimensiunea de 15 caractere);
- valorile introduse trebuie să respecte restricţiile de integritate definite la crearea tabelei (de exemplu, câmpuri definite ca NOT NULL sau UNIQUE).

- **Atunci când se inserează valori de tip dată calendaristică în format predefinit (DD-MON-YY), sistemul presupune în mod automat secolul 20, ora 00:00:00 (miezul nopţii).**

Comanda INSERT - cont.

- Următoarea instrucţiune exemplifică introducerea unei noi înregistrări în tabela profesor:

```
INSERT INTO profesor (cod, nume, prenume, data_nast, sef,
salariu, cod_catedra)
VALUES (107, 'POPESCU', 'SERGIU', '09-DEC-71', 100, 1200, 20);
```

Se poate observa că valorile coloanelor grad şi primă, care nu au fost specificate, vor fi NULL (a se vedea slide-ul 5 din cursul curent).

Comanda INSERT - cont.

- În cazul în care nu se specifică implicit numele coloanelor, valorile trebuie introduse în ordinea în care au fost definite şi nu se poate omite valoarea nici unei coloane. Următoarea instrucţiune va produce acelaşi efect ca cea de mai sus:

```
INSERT INTO profesor
VALUES (107, 'POPESCU', 'SERGIU', '09-DEC-71', NULL, 100,  1200,
NULL, 20);
```

Comanda INSERT - cont.

- **2. Pentru introducerea datelor într-un tabel, prin copierea mai multor înregistrări dintr-un alt tabel sau grup de tabele**; aceste înregistrări sunt rezultatul unei comenzi SELECT.
  În acest caz sintaxa este următoarea:
  `INSERT INTO tabela [(coloana1, coloana 2, ....)] comanda_select;`
- Şi în acest caz trebuie respectate regulile de inserare, singura diferenţă fiind faptul că valorile nou-introduse sunt extrase cu ajutorul unei interogări, acest lucru creând posibilitatea de inserare a mai multor înregistrări în funcţie de anumite condiţii.

Comanda INSERT - cont.

- De exemplu, pentru a insera în tabela nou_profesor, având coloanele cod, nume, prenume şi data_nastere, înregistrările din tabela profesor care au gradul didactic de asistent se poate folosi următoarea instrucţiune:

```
INSERT INTO nou_profesor (cod, nume, prenume, data_nastere)
SELECT cod, nume, prenume, data_nast
FROM profesor
WHERE grad='ASIST';
```

Comanda UPDATE

- Comanda UPDATE este folosită pentru a modifica valorile datelor existente într-un tabel sau în tabelele de bază ale unei vederi şi are următoarea sintaxă generală:

```
UPDATE tabela [alias]
SET atribuire_coloane, [atribuire_coloane, ...]
[WHERE condiţie];
```

unde atribuire_coloane poate avea una dintre următoarele forme:
coloana = {expresie | (subinterogare)} sau
(coloana [,coloana] ...) = (subinterogare)

Comanda UPDATE - cont.

- Se observă că ∃ 2 posibilităţi de modificare:
  - furnizarea în mod explicit a fiecărei valori sau expresii pentru câmpurile ce trebuie modificate;
  - obţinerea valorilor cu ajutorul unei subinterogări.
- Comanda UPDATE modifică valorile înregistrărilor în funcţie de condiţia clauzei WHERE. În lipsa clauzei WHERE, vor fi actualizate toate înregistrările din tabelul dat.
- Expresia furnizată ca nouă valoare a unei coloane poate cuprinde valorile curente ale câmpurilor din înregistrarea care este actualizată.

Comanda UPDATE - cont.

- De exemplu, pentru a mări salariul cu 20% şi prima cu 100 pentru cadrele didactice ce au gradul de asistent, se va folosi următoarea comandă:

```
UPDATE profesor
SET salariu = salariu * 1.2,
    prima = prima + 100
WHERE grad='ASIST';
```

- Pentru a exemplifica actualizarea datelor utilizând subinterogări presupunem că mai avem o tabelă numită PRIMA ce conţine sumele de bani primite suplimentar de unele cadre didactice:

COD	PRIMA
102	100
103	200
102	50

Comanda UPDATE - cont.

- Pentru a modifica datele din tabela PROFESOR pe baza datelor din tabela PRIMA se poate folosi următoarea comandă care conţine o subinterogare corelată şi o subinterogare imbricată:

```
UPDATE profesor
SET prima = (SELECT SUM(prima)
             FROM prima a
             WHERE a.cod=profesor.cod)
WHERE cod IN (SELECT cod
              FROM prima);
```

Comanda UPDATE - cont.

- O altă posibilitate este ca sumele suplimentare conţinute în tabela PRIMA să fie adăugate la prima existentă în tabela PROFESOR:

```
UPDATE profesor
SET prima = (SELECT SUM (prima) + profesor.prima
             FROM prima a
             WHERE a.cod = profesor.cod)
WHERE cod IN (SELECT cod
              FROM prima);
```

Comanda UPDATE - cont.

- Să presupunem acum că toţi asistenţii sunt transferaţi la catedra din care face parte cadrul didactic cu codul 104 şi vor primi acelaşi salariu cu acesta:

```
UPDATE profesor
SET (cod_catedra, salariu) = (SELECT cod_catedra, salariu
                              FROM profesor
                              WHERE cod = 104)
WHERE grad = 'ASIST';
```

Comanda DELETE

- Comanda DELETE realizează ştergerea înregistrărilor dintr-o tabelă sau din tabelele de bază ale unei vederi în funcţie de o anumită condiţie şi are următoarea sintaxă generală:

```
DELETE FROM tabela
[WHERE condiţie];
```

- Similar comenzii UPDATE, comanda DELETE şterge anumite înregistrări în funcţie de condiţia din clauza WHERE.
  În lipsa clauzei WHERE vor fi şterse toate înregistrările din tabelul dat.
  În această clauză pot fi incluse şi subinterogări.

Comanda DELETE - cont.

- De exemplu următoarea comandă şterge toate înregistrările pentru care gradul didactic este asistent
  DELETE FROM profesor
  WHERE grad = 'ASIST';
- **Observaţii**: Comanda DELETE nu poate fi folosită pentru ştergerea valorii unui câmp individual (pentru aceasta folosiţi comanda UPDATE) ci şterge înregistrări complete dintr-un singur tabel.
  În plus, comanda DELETE şterge numai înregistrări din tabel nu şi tabelul.
  Pentru a şterge un tabel se foloseşte comanda DROP TABLE.

Comanda DELETE - cont.

- Un alt aspect important este faptul că, similar comenzilor INSERT şi UPDATE, ştergerea înregistrărilor dintr-un tabel poate determina apariţia unor probleme legate de integritatea referenţială.
  Pentru a evita aceste probleme se pot defini constrângeri de integritate care împiedică operaţiile de inserare, actualizare sau ştergere care ar distruge integritatea referenţială a datelor.

Comanda TRUNCATE

- Pentru a şterge în mod rapid toate înregistrările dintr-o tabelă sau dintr-un cluster se poate folosi comanda TRUNCATE.
- Comanda TRUNCATE este mult mai rapidă decât comanda DELETE din următoarele motive:
  - Comanda TRUNCATE este o comandă DDL, prin urmare se execută dintr-o singură tranzacţie şi deci nu foloseşte segmentul de revenire.
    Comanda trebuie folosită cu precauţie deoarece nu mai poate fi derulată înapoi.
  - Comanda TRUNCATE nu declanşează triggerul DELETE.

Comanda TRUNCATE - cont.

- Comanda are următoarea sintaxă generală:

```
TRUNCATE {TABLE tabel | CLUSTER cluster} [ {DROP | REUSE}
STORAGE]
```

unde:

  - Clauza TABLE specifică numele unei tabele iar clauza CLUSTER specifică numele unui cluster.
    După cum se observă din sintaxă, aceste 2 opţiuni sunt alternative, deci nu se poate specifica într-o comandă TRUNCATE ştergerea rândurilor dintr-o tabelă şi dintr-un cluster în acelaşi timp.

Comanda TRUNCATE - cont.

În cazul în care se specifică clauza TABLE, tabela la care se referă această clauză nu poate face parte dintr-un cluster.
Comanda TRUNCATE se poate executa şi asupra tabelelor organizate pe index.
La truncherea unei tabele, Oracle şterge automat datele din indecşii tabelei.
În cazul în care se specifică clauza CLUSTER, clusterul la care se referă această clauză nu poate fi un cluster hash ci numai un cluster de index.
De asemenea, la truncherea unui cluster, Oracle şterge automat datele din indecşii tabelelor clusterului.

Comanda TRUNCATE - cont.

- Clauza DROP STORAGE eliberează spaţiul alocat înregistrărilor şterse din tabel sau cluster.
  Clauza REUSE STORAGE păstrează spaţiul alocat înregistrărilor şterse din tabel sau cluster.
  Acest spaţiu care nu a fost dealocat poate fi reutilizat doar la operaţii de inserare sau modificare asupra tabelei sau clusterului.
  Aceste 2 opţiuni nu modifică efectul pe care îl are comanda TRUNCATE asupra spaţiului eliberat de datele şterse din indecşii asociaţi.
  Opţiunea implicită este DROP STORAGE.

Comanda TRUNCATE - cont.

- Ştergerea înregistrărilor cu ajutorul comenzii TRUNCATE este mult mai avantajoasă decât eliminarea tabelului şi recrearea lui ulterioară deoarece:
  - Eliminarea tabelului face ca obiectele dependente de acesta să devină invalide, pe când în cazul folosirii comenzii TRUNCATE nu se întâmplă acest lucru;
  - Comanda TRUNCATE nu necesită reacordarea de drepturi asupra tabelului aşa cum se întâmplă dacă acesta a fost eliminat şi apoi recreat;
  - Eliminarea tabelului necesită recrearea indecşilor, constrângerilor de integritate, declanşatoarelor, precum şi specificarea parametrilor de stocare.
- **De exemplu, dacă un utilizator execută comanda `SELECT COUNT (*) FROM nume_tabel`, iar această interogare returnează după un interval destul de îndelungat valoarea 0, se recomandă trunchierea tabelului.**

Bibliografie

F. Ipate, M. Popescu, Dezvoltarea aplicațiilor de baze de date în Oracle 8 și Oracle Forms 6, Editura ALL, 2000.