



UNIUNEA EUROPEANĂ



GUVERNUL ROMÂNIEI



Instrumente Structurale
2007-2013



Platformă de e-learning și curriculum e-content
pentru învățământul superior tehnic

Baze de date 1

9. Actualizarea datelor în SQL

ACTUALIZAREA DATELOR

- ◆ Scopul acestui capitol este descrierea operațiilor care fac parte din limbajul de manipulare al datelor: **adăugarea** de linii într-o tabelă, **modificarea** valorilor unor linii și **ștergerea** acestora.
- ◆ Este de asemenea prezentat mecanismul **tranzacțiilor** și **consistența la citire** care permit utilizatorilor și aplicațiilor care accesează o bază de date să utilizeze o versiune consistentă a acesteia.

STUD

MATR	NUME	AN	GRUPA	DATAN	LOC	TUTOR	PUNCTAJ	CODS
----	-----	--	-----	-----	-----	-----	-----	----
1456	GEORGE	4	1141A	12-MAR-82	BUCURESTI		2890	11
1325	VASILE	2	1122A	05-OCT-84	PITESTI	1456	390	11
1645	MARIA	3	1131B	17-JUN-83	PLOIESTI		1400	11
3145	ION	1	2112B	24-JAN-85	PLOIESTI	3251	1670	21
2146	STANCA	4	2141A	15-MAY-82	BUCURESTI		620	21
3251	ALEX	5	2153B	07-NOV-81	BRASOV		1570	21
2215	ELENA	2	2122A	29-AUG-84	BUCURESTI	2146	890	21
4311	ADRIAN	3	2431A	31-JUL-83	BUCURESTI		450	24
3514	FLOREA	5	2452B	03-FEB-81	BRASOV		3230	24
1925	OANA	2	2421A	20-DEC-84	BUCURESTI	4311	760	24
2101	MARIUS	1	2412B	02-SEP-85	PITESTI	3514	310	24
4705	VOICU	2	2421B	19-APR-84	BRASOV	4311	1290	24

SPEC si BURSA

CODS	NUME	DOMENIU			

11	MATEMATICA	STIINTE EXACTE			
21	GEOGRAFIE	UMANIST			
24	ISTORIE	UMANIST			
TIP			PMIN	PMAX	SUMA

FARA BURSA			0	399	
BURSA SOCIALA			400	899	100
BURSA DE STUDIU			900	1799	150
BURSA DE MERIT			1800	2499	200
BURSA DE EXCEPTIE			2500	9999	300

CERERI PREZENTATE

- ◆ Principalele cereri SQL prin care sunt efectuate aceste operații sunt:
 - ◆ **INSERT** efectuează adăugarea de noi linii,
 - ◆ **UPDATE** permite actualizarea (modificarea) valorilor dintr-o tabelă,
 - ◆ **DELETE** șterge linii dintr-o tabelă,
 - ◆ **COMMIT, ROLLBACK și SAVEPOINT** permit controlul explicit al tranzacțiilor.

INSERT ... VALUES (1)

Inserarea unei linii prin specificarea valorilor acesteia

- ◆ Sintaxa cererii este următoarea:

```
INSERT INTO tabela [(lista_de_coloane)]  
VALUES (lista_de_valori)
```

În cazul în care lista de coloane nu este prezentă în cerere efectul va fi următorul:

- ◆ Se inserează în tabelă o linie completă,
- ◆ Numărul valorilor din listă trebuie să fie același cu numărul de coloane din tabelă,
- ◆ Ordinea valorilor din listă trebuie să fie aceeași cu ordinea coloanelor din cererea de creare a tablei,
- ◆ Tipul valorilor trebuie să fie compatibil cu tipul coloanelor în care se stochează,

INSERT ... VALUES (2)

- ◆ În cazul în care pe o anumită coloană se dorește inserarea unei valori nule, în lista de valori se folosește cuvântul cheie NULL,
- ◆ Dacă la crearea tabelului unei coloane i-a fost asociată o valoare implicită, pentru inserarea ei se folosește cuvântul cheie DEFAULT. În cazul în care se folosește DEFAULT dar coloana nu are o valoare implicită definită, în coloană se va insera o valoare nulă,
- ◆ Valorile pentru coloanele de tip șir de caractere sau dată calendaristică (doar cele în formatul standard) se pun între apostrofi,
- ◆ În cazul în care o valoare de tip dată calendaristică nu este în format standard se folosește funcția TO_DATE pentru conversia șirului respectiv în dată calendaristică,
- ◆ Fiecare linie se inserează cu o cerere separată.

EXEMPLU

- ◆: Inserarea specializărilor în tabela SPEC se face cu cererile:

```
INSERT INTO SPEC VALUES (11,  
    'MATEMATICA', 'STIINTE EXACTE');
```

```
INSERT INTO SPEC VALUES (21,  
    'GEOGRAFIE', 'UMANIST');
```

```
INSERT INTO SPEC VALUES (24, 'ISTORIE',  
    'UMANIST');
```

- ◆ Pentru inserarea unei noi specializări având codul 30, nume încă necunoscut și un domeniu implicit se poate folosi cererea:

```
INSERT INTO SPEC VALUES (30, NULL,  
    DEFAULT);
```


EFFECT

- ◆ Rezultatul celor patru inserări va fi următorul (am considerat că la crearea tabelului SPEC, pentru coloana DOMENIU s-a asociat valoarea implicită 'UMANIST'):

CODS	NUME	DOMENIU
11	MATEMATICA	STIINTE EXACTE
21	GEOGRAFIE	UMANIST
24	ISTORIE	UMANIST
30		UMANIST

INSERT ... VALUES (3)

- ◆ În cazul în care cererea conține și o **listă de coloane**, rezultatul acesteia este inserarea unei linii incomplete: pentru coloanele care lipsesc din listă se vor insera automat valorile implicite - dacă există - sau valori nule în caz contrar.
- ◆ Observații:
 - ◆ Listele de coloane și valori trebuie să conțină **același număr de elemente**,
 - ◆ Cele două liste **se corespund pozițional**: prima valoare va fi stocată în prima coloană din listă, s.a.m.d.
 - ◆ **Tipul valorilor trebuie să fie compatibil** cu tipul coloanelor în care se stochează.

EXEMPLU

- ◆ Exemplu: inserarea specializării cu codul 30 se putea face și astfel:

```
INSERT INTO SPEC (CODS) VALUES (30);
```

rezultatul obținut fiind același ca mai sus.

- ◆ În toate cazurile, inserarea va eșua dacă noua linie nu respectă vreuna din constrângerile de integritate care sunt active în acel moment pentru tabela în care se face adăugarea.

CE PUTEM PUNE IN LISTA

- ◆ În lista de valori se mai pot folosi:
 - ◆ Expresii
 - ◆ Subcereri

EXPRESII

- ◆ In acest caz valoarea expresiei trebuie să fie compatibilă cu tipul coloanei.
- ◆ Expresiile pot conține operatori și funcții.
- ◆ Exemplu: următoarea cerere de inserare este validă:

```
INSERT INTO STUD (MATR, NUME, DATAN,  
CODS) VALUES (1200+15, 'ION ' ||  
'DOBRE', SYSDATE, 24);
```

- ◆ Valorile inserate vor fi: 1215 , 'ION DOBRE', data curentă și 24.

SUBCERERI

◆ Rezultatul unei subcereri necelelate care întoarce o singură valoare poate fi folosit în lista de valori a cererii INSERT.

◆ Exemplu: cererea:

```
INSERT INTO SPEC(CODS)
```

```
VALUES ((SELECT MAX(CODS) FROM SPEC)+1);
```

va insera o specializare având codul 25 (în cazul în care valoarea maximă pe coloana CODS este 24).

INSERT .. SELECT (1)

- ◆ Sintaxa este în acest caz următoarea:

```
INSERT INTO tabela [(lista_de_coloane)]  
cerere_select
```

- ◆ Modul de execuție al acestei cereri este următorul:
 - ◆ Se execută cererea SELECT,
 - ◆ Liniile rezultatului sunt inserate în tabelă,
 - ◆ Numărul de coloane al tabelii/listei de coloane trebuie să fie egal cu cel al rezultatului cererii SELECT iar tipurile trebuie să fie compatibile,
 - ◆ În cazul în care o linie din rezultatul cererii SELECT nu satisface constrângerile de integritate ale tabelii în care se face inserarea, întreaga comandă eșuează (nici celelalte linii nu sunt inserate).

EXEMPLU

◆ Fie o tabelă NRSTUD(CODS, NRSTUD) care conține pentru fiecare specializare numărul de studenți al acesteia.

◆ Umplerea ei se poate face prin cererea:

```
INSERT INTO NRSTUD
```

```
    SELECT CODS, COUNT(*) FROM STUD GROUP  
    BY CODS;
```

◆ Conținutul tablei NRSTUD va fi:

```
CODS NRSTUD
```

```
-----
```

```
11      3
```

```
21      4
```

```
24      5
```


DELETE

- ◆ Sintaxa cererii care efectuează ștergerea unor linii dintr-o tabelă este următoarea:

```
DELETE FROM tabela  
[WHERE conditie]
```

- ◆ În clauza WHERE pot să apară aceleași elemente ca și în cazul cererilor SELECT (deci **inclusiv subcereri**).
- ◆ Efectul cererii este ștergerea tuturor liniilor pentru care condiția este verificată.
- ◆ În cazul absenței clauzei WHERE se șterg **toate liniile tablei**.

EXAMPLE

- ◆ Exemplul 1: dacă se dorește ștergerea tuturor studenților specializării cu codul 11 cererea este:

- ◆ **DELETE FROM STUD WHERE CODS = 11;**

- ◆ Exemplul 2: ștergerea liniilor corespunzătoare studenților cu cel mai mic punctaj din fiecare specializare se face cu cererea:

```
DELETE FROM STUD
```

```
WHERE MATR IN
```

```
(SELECT MATR FROM STUD
```

```
WHERE (CODS, PUNCTAJ) IN
```

```
(SELECT CODS, MIN(PUNCTAJ)
```

```
FROM STUD GROUP BY CODS));
```

OBSERVATIE

- ◆ Stergerea nu se va efectua și va semnala o eroare dacă una dintre liniile care îndeplinesc condiția conține o cheie referită printr-o constrângere de tip FOREIGN KEY într-o tabelă din baza de date.

UPDATE

- ◆ Sintaxa cererii de actualizare (modificare) a valorilor dintr-o tabelă este:

```
UPDATE tabela
SET coloana1 = expresie1 [,
    coloana2=expresie2, ...]
[WHERE conditie];
```

- ◆ Efectul este următorul:
 - ◆ În WHERE pot să apară aceleași elemente ca și în cazul cererilor SELECT,
 - ◆ Toate liniile care îndeplinesc condiția din WHERE vor fi actualizate conform cu specificațiile din SET. În cazul absenței clauzei WHERE, toate liniile tablei vor fi actualizate,
 - ◆ Expresiile din clauza SET se evaluează pornind de la valorile conținute în linia care se modifică și ele trebuie să aibă un tip compatibil cu al coloanelor asociate,
 - ◆ O aceeași coloană nu poate apare de două ori în clauza SET.

EXEMPLE

- ◆ Exemplul 1: Mărirea cu 10% a punctajului studenților de la specializarea cu codul 11 se face cu cererea:

```
UPDATE STUD
```

```
SET PUNCTAJ = PUNCTAJ * 1.1
```

```
WHERE CODS = 11
```

- ◆ Exemplul 2: În cazul în care, pentru toți studenții, pe lângă mărirea punctajului cu 10% se dorește și incrementarea anului de studii pentru studenții din anii 1-4 și stocarea pe coloana AN a valorii 0 pentru studenții de anul 5, cererea este:

```
UPDATE STUD
```

```
SET PUNCTAJ = PUNCTAJ * 1.1,
```

```
AN = DECODE (AN, 5, 0, AN+1);
```

EXAMPLE – cont.

- ◆ Exemplul 3: În cazul în care pentru o coloană se dorește actualizarea valorilor prin aducerea lor la valoarea implicită asociată la crearea tabelii sau la o valoare nulă se folosesc de asemenea cuvintele cheie **DEFAULT** și **NULL**.
- ◆ În cazul folosirii lui **DEFAULT**, dacă nu există o valoare implicită asociată, în coloana respectivă se va stoca o valoare nulă.

```
UPDATE SPEC SET DOMENIU = DEFAULT;
```

```
UPDATE SPEC SET DOMENIU = NULL;
```

SUBCERERI IN SET

- ◆ Expresiile din clauza SET pot conține **subcereri** pe tabela care se actualizează **sau** pe o altă tabelă din baza de date.
- ◆ Exemplu: Pentru actualizarea numărului de studenți din tabela NRSTUD descrisă anterior se poate folosi următoarea cerere corelată:

```
UPDATE NRSTUD N
SET NRSTUD =
  (SELECT COUNT (*)
   FROM STUD
   WHERE CODS = N.CODS);
```

SUBCERERI IN SET – cont.

- ◆ Subcererile pot fi prezente atât pe clauza WHERE cât și în locul numelui tabelului (acest caz mai târziu).
- ◆ De exemplu, dacă se dorește actualizarea numărului de studenți doar pentru specializările având mai mult de trei studenți cererea este:

```
UPDATE NRSTUD
SET NRSTUD =
  (SELECT COUNT (*)
   FROM STUD WHERE CODS = NRSTUD.CODS)
WHERE CODS IN
  (SELECT CODS FROM STUD GROUP BY CODS
   HAVING COUNT (*) > 3);
```


EFFECT CONSTRANGERI

- ◆ În cazul în care noile valori ale unei linii nu satisfac constrângerile de integritate active pentru tabela respectivă, **cererea va eșua** și va returna un mesaj de eroare.
- ◆ Este cazul actualizărilor în care:
 - ◆ Noua valoare este nulă și coloana are o constrângere de integritate de tip NOT NULL,
 - ◆ Noua valoare face ca linia să conțină o valoare de cheie care mai există în tabelă, într-o altă linie,
 - ◆ Noua valoare nu verifică o constrângere de tip CHECK,
 - ◆ Vechea valoare a cheii este referită prin constrângeri de tip FOREIGN KEY în alte tabele,
 - ◆ Noua valoare a unei chei străine nu se regăsește în tabela referită prin FOREIGN KEY.

CONSISTENTA LA CITIRE

- ◆ Sistemul Oracle implementează mecanisme care îi permit fiecărui utilizator (uman sau aplicație care utilizează baza de date) să aibă în orice moment o **viziune consistentă** a datelor.
- ◆ În continuare sunt prezentate elementele de bază privind **consistența la citire, blocările implicite** efectuate de sistem și **mecanismele de implementare a tranzacțiilor**.

CONSISTENTA.. (2)

- ◆ În mod curent pe aceeași bază de date operează simultan mai mulți utilizatori care pot efectua două tipuri de operații:
 - ◆ **Operații de citire.** În această categorie intră în principal regăsirile de date prin cereri SELECT.
 - ◆ **Operații de scriere.** În această categorie sunt operațiile de INSERT, UPDATE, DELETE și MERGE descrise anterior.
- ◆ În lipsa unei tratări necorespunzătoare, execuția simultană a unei operații de scriere cu o altă operație (de citire sau de scriere) poate duce la inconsistențe în utilizarea bazei de date.

EXEMPLU

- ◆ Fie doi utilizatori care lucrează simultan pe tabela STUD, **primul incrementând** anul de studii (o cerere UPDATE) și **al doilea afișând** conținutul tablei (cerere SELECT). Fără mecanisme de păstrare a unei viziuni consistente asupra datelor este posibilă următoarea situație:
- ◆ Unele dintre linile afișate pentru al doilea utilizator conțin anul de studii încă nemodificat (cererea UPDATE nu a ajuns încă la aceste linii),
- ◆ Celelalte linii afișate conțin anul modificat de cererea UPDATE în curs.

CUM FACE ORACLE:

- ◆ Aceleași date pot fi **citite simultan** de oricâți utilizatori,
- ◆ Aceleași date **nu pot fi modificate simultan de doi utilizatori**. În momentul în care unul din ei modifică o valoare, sistemul efectuează o **blocare implicită a liniei** care o conține blocând accesul pentru scriere al altor utilizatori.
- ◆ Aceleași date **pot fi modificate la un moment dat de un utilizator** și citite de oricâți alții,
- ◆ Modificările făcute de un utilizator în conținutul tabelelor **nu sunt vizibile** pentru ceilalți utilizatori până în momentul în care se execută implicit sau explicit **înscrierea lor permanentă** în baza de date (operație numită în terminologia de specialitate **comitere**).

CUM FACE ORACLE – cont.

- ◆ Până la comiterea modificărilor, operațiile de citire efectuate de alți utilizatori returnează conținutul de date **anterior modificării lor**,
- ◆ Până nu au fost comise, **modificările pot fi revocate**, anulându-se efectul cererilor care le-au efectuat,
- ◆ În momentul comiterii sau revocării modificărilor **se ridică toate blocările** aferente acestora,

În acest fel operațiile de citire și modificare a datelor **se pot executa simultan** fără să interfereze unele cu celelalte.

De asemenea **sunt prevenite inconsistențele** care pot apare la modificarea simultană a acelorași date.

EXEMPLU

- ◆ Exemplu: să considerăm trei posturi de lucru, U1, U2 și U3 de la care se execută următoarele cereri, la momentele de timp specificate ($t1 < t2 < \dots < t6$).
- ◆ Inițial presupunem că tabela NRSTUD conține informații corecte despre specializările și numărul de studenți asociat lor:

```
U1/t1: UPDATE NRSTUD SET NRSTUD = 0;
```

```
U1/t2: SELECT * FROM NRSTUD;
```

```
U2/t3: SELECT * FROM NRSTUD;
```

```
U3/t4: UPDATE NRSTUD SET CODS = CODS + 1;
```

```
U1/t5: COMMIT;
```

```
U2/t6: SELECT * FROM NRSTUD;
```

```
U1/t1: UPDATE NRSTUD SET NRSTUD = 0;
U1/t2: SELECT * FROM NRSTUD;
U2/t3: SELECT * FROM NRSTUD;
U3/t4: UPDATE NRSTUD SET CODS = CODS + 1;
U1/t5: COMMIT;
U2/t6: SELECT * FROM NRSTUD;
```

- ◆ La momentul t1, U1 execută o actualizare a datelor, aducând la 0 valorile de pe coloana NRSTUD. Noile valori nu sunt încă permanente, fiind vizibile doar pentru U1.
- ◆ În același timp sistemul blochează liniile modificate, ele fiind accesibile pentru scriere doar pentru U1.
- ◆ La momentul t2, U1 afișează datele. Rezultatul conține valorile modificate.


```
U1/t1: UPDATE NRSTUD SET NRSTUD = 0;  
U1/t2: SELECT * FROM NRSTUD;  
U2/t3: SELECT * FROM NRSTUD;  
U3/t4: UPDATE NRSTUD SET CODS = CODS + 1;  
U1/t5: COMMIT;  
U2/t6: SELECT * FROM NRSTUD;
```

- ◆ La momentul t3, U2 afișează de asemenea conținutul tabelii NRSTUD. Datele afișate vor conține vechile valori din coloana NRSTUD (diferite de 0), cele modificate fiind disponibile decăt pentru U1.
- ◆ La momentul t4, U3 lansează o cerere de actualizare pe tabela NRSTUD. Deoarece U1 nu a comis sau revocat încă modificările, cererea lui U3 intră în așteptare, liniile necesare fiind blocate de sistem.
- ◆ La momentul t5 U1 comite modificările. Ele devin permanente iar liniile blocate din NRSTUD sunt deblocate. În consecință se execută și cererea lui U3 care fusese plasată în așteptare.

```
U1/t1: UPDATE NRSTUD SET NRSTUD = 0;  
U1/t2: SELECT * FROM NRSTUD;  
U2/t3: SELECT * FROM NRSTUD;  
U3/t4: UPDATE NRSTUD SET CODS = CODS + 1;  
U1/t5: COMMIT;  
U2/t6: SELECT * FROM NRSTUD;
```

- ◆ Sistemul blochează pentru U3 liniile modificate din NRSTUD. Ele nu mai pot fi scrise decât de acesta.
- ◆ La momentul t6 se execută din nou o cerere de regăsire a lui U1. Datele afișate vor conține valorile modificate ale numărului de studenți (0) și cele încă nemodificate pentru codul specializării (U3 nu a comis încă modificările).
- ◆ Pentru testarea acestui comportament se pot folosi trei ferestre SQL*Plus deschise simultan pe același calculator, cu același nume de utilizator Oracle.

TRANZACTII

- ◆ În contextul acestui capitol, o **tranzacție** poate fi definită ca o **succesiune de cereri DML** (printre care pot exista și cereri de regăsire a informațiilor) executate într-o aceeași sesiune de lucru.
- ◆ Modificarile facute in baza de date pe parcursul unei tranzactii pot fi la final fie toate comise fie toate revocate
- ◆ Tranzactia este din acest punct de vedere o operatie **atomica**
- ◆ La încheierea unei tranzacții, toate liniile blocate de sistem pentru aceasta sunt eliberate.

TRANZACTII – cont.

- ◆ Tranzacția începe cu prima cerere DML și se termină în momentul în care:
 - ◆ Se execută o cerere COMMIT care face permanente modificările efectuate de tranzacție.
 - ◆ Se execută o cerere ROLLBACK care revocă modificările efectuate de tranzacție.
 - ◆ Se execută o cerere DDL (de exemplu CREATE) sau o cerere DCL. Tranzacția se încheie înainte de execuția acesteia prin comiterea modificărilor
 - ◆ Se încheie sesiunea de lucru (ieșire din SQL*Plus). În acest caz modificările efectuate de tranzacție sunt de asemenea comise.
 - ◆ Apare o cădere hardware sau software a sistemului. Modificările efectuate de tranzacție sunt revocate.

CERERI DDL SI DCL

- ◆ În ceea ce privește cererile DDL și DCL, acestea se constituie de asemenea în tranzacții.
- ◆ Execuția unei astfel de cereri se face astfel:
 - ◆ Anterior execuției este finalizată prin comitere tranzacția în curs.
 - ◆ Se execută cererea DDL sau DCL.
 - ◆ Efectul acesteia este de asemenea comis. Cererile de acest tip nu pot fi revocate prin ROLLBACK.

SQL*PLUS

- ◆ În SQL*Plus există de asemenea posibilitatea de a specifica faptul că fiecare cerere DML trebuie comisă automat după execuție. Setarea poate fi efectuată astfel:
 - ◆ **SET AUTOCOMMIT OFF** - Comiterea și revocarea sunt la nivel de tranzacție și nu la nivel de cerere DML. Este opțiunea implicită.
 - ◆ **SET AUTOCOMMIT ON** - Fiecare cerere DML este o tranzacție.
- ◆ **Atentie:** închiderea ferestrei SQL*Plus (din butonul x) este considerată incident software și tranzacția curentă este revocată!

COMMIT SI ROLLBACK

- ◆ Sintaxa cererilor COMMIT și ROLLBACK este următoarea:
 - ◆ **COMMIT;** - Toate modificările efectuate de cererile DML ale tranzacției sunt comise.
 - ◆ **ROLLBACK;** - Toate modificările efectuate de cererile DML ale tranzacției sunt revocate.
- ◆ In ambele cazuri tranzactia curenta se incheie.

SAVEPOINT

- ◆ În sistemul Oracle există și posibilitatea ca o tranzacție să fie **revocată parțial**.
- ◆ Pentru asta însă trebuie setate **puncte de revenire** cu cererea SAVEPOINT. Sintaxa aferentă acestor operații este următoarea:

SAVEPOINT nume;

ROLLBACK TO [SAVEPOINT] nume;

- ◆ **SAVEPOINT** nume: specifică definirea unui punct de revenire, având asociat un nume. Pe cuprinsul execuției unei tranzacții se pot fixa mai multe astfel de puncte.
- ◆ **ROLLBACK TO nume**: specifică revocarea tuturor modificărilor efectuate după fixarea punctului de revenire cu numele respectiv. Modificările efectuate înainte de acest punct rămân și nu sunt revocate. Ele pot fi comise sau revocate ulterior, revocarea putând fi de asemenea parțială. În același timp, toate punctele de revenire fixate ulterior celui în cauză se pierd.

EXEMPLU

- ◆ Fie următoarea succesiune de cereri SQL, executate imediat după intrarea într-o sesiune de lucru SQL*Plus.

```
INSERT INTO SPEC ...
```

```
SAVEPOINT P1;
```

```
UPDATE STUD ...
```

```
SAVEPOINT P2
```

```
DELETE SPEC ...
```

```
ROLLBACK TO P2
```

```
INSERT INTO STUD ...
```

```
ROLLBACK TO P1;
```

```
CREATE TABLE ...
```

```
INSERT INTO SPEC ...  
    SAVEPOINT P1;  
UPDATE STUD ...  
    SAVEPOINT P2  
DELETE SPEC ...  
ROLLBACK TO P2  
INSERT INTO STUD ...  
ROLLBACK TO P1;  
CREATE TABLE ...
```

- ◆ Tranzacția începe cu o inserare în tabela SPEC. După efectuarea acesteia se fixează punctul de revenire P1.
- ◆ Se execută o actualizare a tabelului STUD și fixarea punctului de revenire P2.
- ◆ Se execută o ștergere de linii din SPEC. Urmează însă o revenire la punctul P2 care anulează efectele acestei ștergeri.

```
INSERT INTO SPEC ...  
    SAVEPOINT P1;  
UPDATE STUD ...  
    SAVEPOINT P2  
DELETE SPEC ...  
ROLLBACK TO P2  
INSERT INTO STUD ...  
ROLLBACK TO P1;  
CREATE TABLE ...
```

- ◆ Se execută o inserare în tabela STUD. Ulterior însă se revine în punctul P1. Sunt revocate astfel efectele inserării și actualizării tabelii STUD.
- ◆ Ultima cerere din exemplu este un CREATE. Înainte de execuția sa sunt comise modificările executate de tranzacție. Singura cerere DML care încă nu a fost revocată este prima inserare în SPEC și va fi singura comisă.
- ◆ Se execută CREATE care fiind cerere DDL se comite automat.

CONCLUZII

- ◆ Mecanismele de tranzacție și blocările implicite puse la dispoziție de sistem permit asigurarea consistenței bazei de date în contextul operării concurente cu aceleași date.
- ◆ Pe lângă blocările implicite, Oracle pune la dispoziția utilizatorilor și mecanisme de blocare explicită care nu fac obiectul lucrării de față.

Bibliografie

1. **Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer D. Widom:** *Database Systems: The Complete Book*, Prentice-Hall, Englewood Cliffs, NJ, 2002.
2. **F. Rădulescu :** *Oracle SQL, PL/SQL*, Editura Printech, ISBN 973-718-203-02005