Sisteme de baze de date

Curs 8 – Accesul concurent la date şi păstrarea consistenţei acestora

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

Acces concurent

- **Baza de date Oracle** permite acces concurent, adică **poate fi accesată simultan de mai mulţi utilizatori în cadrul mai multor sesiuni de lucru**.
- O **problemă fundamentală** într-o BD cu acces concurent este **păstrarea consistenţei datelor.** Aceasta înseamnă pe de o parte că în cadrul fiecărei sesiuni de lucru, utilizatorul trebuie să aibă o “vedere” consistentă asupra BD, incluzând modificările vizibile făcute de către alţi utilizatori, iar pe de altă parte Oracle trebuie să împiedice modificările incorecte ale datelor, care ar putea compromite integritatea acestora.

Acces concurent - cont.

- Modalitatea cea mai simplă de a gestiona problema accesului concurent la date ar fi ca fiecare utilizator să-şi aştepte rândul pentru a accesa baza de date.
  Evident, această soluţie este total neconvenabilă, scopul unui SGBD fiind de a reduce aceste aşteptări a.î. ele să fie inexistente sau neglijabile pentru alţi utilizatori.
  Cu alte cuvinte, nu se pot face compromisuri nici în ceea ce priveşte consistenţa datelor, dar nici în ceea ce priveşte performanţele sistemului.
  Mecanismul prin care **Oracle** gestionează accesul concurent la date **foloseşte un model de consistenţă multiversiune şi diverse tipuri de blocări**, care vor fi discutate mai târziu în acest curs.
  Aceste mecanisme au la bază conceptul de **tranzacţie**.

Tranzacţiile şi asigurarea consistenţei la scriere

- Conceptul de tranzacţie este fundamental pentru modul în care Oracle asigură consistenţa datelor.
- **O tranzacţie este alcătuită din una sau mai multe instrucţiuni SQL.**
  O tranzacţie este cea mai mică unitate de lucru în Oracle, în sensul că ∀ tranzacţie, fie sunt executate toate schimbările făcute BD de către tranzacţie, fie nu este executată nici una dintre modificări.
  Cu alte cuvinte, o tranzacţie nu poate fi executată parţial.
  La sfârşitul unei tranzacţii, schimbările făcute BD **sunt fie permanentizate** (commited), **fie sunt anulate**, în acest ultim caz spunem că tranzacţia a fost derulată înapoi (rolled back).
  După ce o tranzacţie a fost permanentizată sau derulată înapoi, următoarea tranzacţie va începe cu următoarea instrucţiune SQL.

Tranzacţiile şi asigurarea consistenţei la scriere

- În timpul executării unei tranzacţii, modificările făcute asupra BD de către acestea nu sunt vizibile altor sesiuni de lucru.
  Dacă tranzacţia este permanentizată, atunci schimbările făcute de aceasta devin vizibile pentru alte sesiuni care încep după permanentizarea acesteia.
  Dacă tranzacţia este derulată înapoi, atunci modificările asupra datelor efectuate de către aceasta sunt anulate, astfel încât datele afectate vor rămâne neschimbate, ca şi când instrucţiunile SQL din tranzacţie nu au fost niciodată executate.

Tranzacţiile şi asigurarea consistenţei la scriere

- Să luăm de exemplu o bază de date a unei bănci. Când un client transferă o sumă de bani dintr-un cont de depozit într-un cont curent, tranzacţia poate să conţină trei operaţii separate: retragerea sumei din contul de depozit, depunerea sumei în contul curent şi înregistrarea operaţiunii într-un jurnal:

```
UPDATE cont_depozit
SET balanta = balanta - 700
WHERE nr_cont = 87410;

UPDATE cont_depozit
SET balanta = balanta + 700
WHERE nr_cont = 87411;
```

Tranzacţiile şi asigurarea consistenţei la scriere

```
INSERT INTO jurnal(nr_operatiune, suma, cont_debitor, cont_creditor)
VALUES (jurnal_seq.NEXTVAL, 700, 87410, 87411);

COMMIT;
```

- Pentru a asigura corectitudinea datelor, Oracle trebuie să garanteze că toate cele 3 operaţii sunt efectuate.
  Dacă ceva neaşteptat, de exemplu o defecţiune hard, împiedică executarea uneia dintre instrucţiuni, atunci celelalte instrucţiuni ale tranzacţiei trebuie anulate, adică tranzacţia trebuie derulată înapoi.
  Deci, dacă una dintre instrucţiuni nu se poate executa cu succes, atunci celelalte trebuie anulate pentru a menţine consistenţa bazei de date.

Tranzacţiile şi asigurarea consistenţei la scriere

- O tranzacţie începe cu prima comandă executabilă SQL şi se încheie când apare una dintre următoarele comenzi sau evenimente:
  - COMMIT [WORK] sau ROLLBACK [WORK].
    Comanda COMMIT sau COMMIT WORK permanentizează modificările făcute de către tranzacţie.
    Comanda ROLLBACK sau ROLLBACK WORK derulează înapoi tranzacţia;
  - o comandă DDL (CREATE, ALTER, DROP).
    Deci orice comandă DDL este implicit permanentizată, astfel încât orice tranzacţie nu poate conţine decât cel mult o comandă DDL;

Tranzacţiile şi asigurarea consistenţei la scriere

- sfârşitul sesiunii curente de lucru, în acest caz făcându-se implicit o permanentizare;
- defecţiune hard sau soft, caz în care tranzacţia este derulată înapoi pentru recuperarea datelor (rollback on recovery) şi păstrarea integrităţii acestora.

- După terminarea tranzacţiei, următoarea comandă executabilă SQL va începe în mod automat o nouă tranzacţie.

Comanda AUTOCOMMIT

- Dacă se foloseşte utilitarul SQL*Plus, există posibilitatea ca după fiecare comandă DML să aibă loc o permanentizare automată a datelor (un COMMIT implicit).
  Acest lucru se poate realiza folosind comanda SQL SET AUTO[COMMIT], având sintaxa:
  SET AUTO[COMMIT] [ON|OFF]
- În cazul folosirii acestei comenzi cu opţiunea ON (SET AUTO ON sau SET AUTOCOMMIT ON) o tranzacţie nu va putea conţine decât cel mult o singură comandă DML, iar instrucţiunea ROLLBACK nu va mai avea nici un efect, datele fiind permanentizate implicit ori de câte ori este executată o comandă DML. Permanentizarea implicită a unei comenzi DML este anulată la executarea comenzii SET AUTO[COMMIT] cu opţiunea OFF (SET AUTO OFF sau SET AUTOCOMMIT OFF).

Puncte de salvare

- În cazul tranzacţiilor mai lungi, care conţin multe instrucţiuni SQL, **pentru a împărţi tranzacţia în părţi mai mici, pot fi declaraţi delimitatori intermediari**, numite puncte de salvare (savepoints).
  Un punct de salvare poate fi declarat folosind comanda SAVEPOINT, având sintaxa:
  SAVEPOINT nume_punct_salvare.
- Declararea unor puncte de salvare în interiorul unei tranzacţii permite posibilitatea ca la un moment ulterior să fie derulate înapoi toate instrucţiunile executate începând cu un punct de salvare specificat.
  Acest lucru se poate face cu comanda ROLLBACK sau ROLLBACK WORK cu specificaţia TO [SAVEPOINT]:
  ROLLBACK [WORK] TO [SAVEPOINT] nume_punct_salvare

Puncte de salvare - cont.

- De exemplu, să considerăm următoarea secvenţă de comenzi SQL:

```
INSERT INTO departament (cod_dept, cod_tara, nume_dept)
VALUES (dept_seq.NEXTVAL, 40, 'Proiectare');

SAVEPOINT alfa;

INSERT INTO departament (cod_dept, cod_tara, nume_dept)
VALUES (dept_seq.NEXTVAL, 40, 'Vanzari');

SAVEPOINT beta;

INSERT INTO departament (cod_dept, cod_tara, nume_dept)
VALUES (dept_seq.NEXTVAL, 40, 'IT');
```

Puncte de salvare - cont.

- În acest punct, anularea celei de-a 3-a inserări în tabelul departament se poate face folosind comanda:

  ```
  ROLLBACK TO beta;
  ```

  Anularea ultimelor 2 inserări se poate face folosind comanda:

  ```
  ROLLBACK TO alfa;
  ```

  Anularea tuturor celor 3 inserări şi încheierea tranzacţiei se face folosind comanda:

  ```
  ROLLBACK;
  ```

Asigurarea consistenţei cu ajutorul tranzacţiilor

- Tranzacţiile furnizează utilizatorului bazei de date sau dezvoltatorului de aplicaţii capacitatea de a garanta consistenţa modificărilor operate asupra datelor.
  Pentru a asigura această consistenţă, comenzile SQL trebuie să fie grupate în mod logic în tranzacţii.
  O tranzacţie trebuie să fie o unitate logică de lucru, nici mai mult, nici mai puţin.
  Datele din baza de date trebuie să fie consistente înainte de începerea tranzacţiei şi la sfârşitul acesteia.
  În plus, o tranzacţie trebuie să cuprindă doar o singură modificare consistentă a datelor.

Asigurarea consistenţei cu ajutorul tranzacţiilor

- De exemplu, să considerăm exemplul transferului bancar anterior.
  Cele 3 acţiuni efectuate (retragerea sumei din contul de depozit, depunerea ei în contul curent şi înregistrarea în jurnal) trebuiau ori să fie toate executate, ori nici una.
  Pentru a asigura consistenţa datelor, orice altă acţiune fără legătură cu operaţiunea dată (de exemplu, un nou depozit într-un alt cont) nu trebuie inclusă în aceeaşi tranzacţie.

Modelul multiversiune şi consistenţa la citire

- Modelul multiversiune, furnizat de către Oracle, asigură consistenţa la citire (read consistency), adică:
  - Garantează că setul de date văzut de orice instrucţiune SQL este consistent şi nu se schimbă în timpul execuţiei unei instrucţiuni; cu alte cuvinte - se spune că Oracle asigură o consistenţă la citire la nivel de instrucţiune;
  - Operaţiile de citire (SELECT) nu trebuie să vadă datele care sunt în proces de schimbare;
  - Operaţiile de scriere (INSERT, UPDATE, DELETE) nu trebuie să afecteze consistenţa datelor şi să întrerupă sau să intre în conflict cu alte operaţii de scriere concurente.

Modelul multiversiune şi consistenţa la citire

- Cea mai simplă modalitate de a ne imagina un sistem care asigură consistenţa la citire este de ne închipui că fiecare utilizator operează asupra unei copii proprii a bazei de date - de unde şi numele de model multiversiune.

Implementarea modelului multiversiune

- Consistenţa la citire este implementată de către Oracle prin păstrarea unei copii a datelor în segmentele de revenire.
  Când este efectuată o operaţie de INSERT, UPDATE sau DELETE asupra unui tabel, Oracle va face o copie a datelor înainte ca acestea să se modifice într-un segment de revenire.
  Toate operaţiile de citire efectuate asupra tabelului în alte sesiuni de lucru vor vedea datele aşa cum erau ele înainte de schimbare - deci vor vedea datele din segmentul de revenire.
  Înainte ca datele fie permanentizate (prin COMMIT explicit sau implicit), doar utilizatorul din sesiunea curentă va vedea datele modificate, toţi ceilalţi vor vedea datele din segmentul de revenire.

Implementarea modelului multiversiune

- Dacă schimbările sunt permanentizate, atunci ele vor deveni vizibile şi pentru ceilalţi utilizatori.
  În acest caz, spaţiul ocupat de datele vechi în segmentul de revenire este eliberat şi poate fi utilizat din nou.
  Dacă tranzacţia este derulată înapoi (prin ROLLBACK sau în cazul unei defecţiuni) atunci datele din segmentul de revenire sunt scrise înapoi în baza de date, iar ceilalţi utilizatori văd în continuare datele iniţiale, ca şi cum modificările făcute de către tranzacţie nu s-ar fi efectuat.

Tranzacţii de citire şi consistenţa la citire la niv. de tranzacţie

- **În mod implicit consistenţa la citire asigurată de către Oracle este la nivel de instrucţiune**, adică datele nu se schimbă în timpul efectuării unei instrucţiuni.
  În acest caz, p**utem avea, în cadrul aceleiaşi sesiuni de lucru, 2 interogări identice care produc rezultate diferite** - aceasta se poate întâmpla dacă între cele 2 interogări au fost permanentizate schimbările făcute de către o altă tranzacţie.
  În anumite situaţii însă, dacă o tranzacţie cuprinde mai multe interogări, se poate dori ca toate aceste interogări să aibă o vedere consistentă asupra datelor în raport cu acelaşi moment de timp.

Tranzacţii de citire şi consistenţa la citire la niv. de tranzacţie

- Cu alte cuvinte, interogările din această tranzacţie nu vor simţi efectul permanentizărilor făcute de către alte tranzacţii după începerea tranzacţiei curente.
  În acest caz se spune că tranzacţia este de citire (read only transaction) iar consistenţa la citire asigurată de către Oracle este la nivel de tranzacţie (transaction-level read consistency).
  Consistenţa la citire la nivel de tranzacţie este implementată similar cu cea la nivel de instrucţiune, adică folosind segmente de revenire.

Tranzacţii de citire şi consistenţa la citire la niv. de tranzacţie

- Pentru a începe o tranzacţie de citire se foloseşte comanda
  SET TRANSACTION READ ONLY.
  Această comandă trebuie să fie prima instrucţiune din tranzacţie.
  După executarea acestei instrucţiuni, toate permanentizările făcute de către alte tranzacţii nu vor fi vizibile în tranzacţia curentă.
  O tranzacţie de citire nu poate conţine decât interogări (instrucţiuni SELECT); comenzile DML nu sunt permise iar comanda SELECT ... FOR UPDATE va produce o eroare.
  O tranzacţie de citire se va termina atunci când se vor executa comenzile COMMIT [WORK], ROLLBACK [WORK] sau la executarea unei comenzi DDL - deoarece o comandă DDL realizează un COMMIT implicit.
  În timpul unei tranzacţii de citire, alţi utilizatori pot continua să interogheze şi să modifice datele.

Blocări

- Blocările (locks) sunt folosite de Oracle pentru a asigura integritatea datelor, permiţând în acelaşi timp accesul concurent la date de către un număr infinit de utilizatori.
- În principal, blocările folosite de Oracle sunt de 2 feluri:

Blocări - cont.

- **Blocări de date sau blocări DML**: Aceste blocări protejează datele.
- **Blocări de dicţionar sau blocări DD**L: Acestea protejează definiţia unui obiect al schemei (de exemplu a unui tabel) în timp ce o operaţie DDL acţionează asupra acestuia sau face referire la acesta (după cum am menţionat înainte, fiecare operaţie DDL permanentizează implicit tranzacţia din care face parte, a.î. blocarea este necesară doar pe durata unei astfel de operaţii).
  De exemplu, dacă un utilizator creează o procedură atunci toate obiectele la care se face referinţă în acea procedură vor fi blocate, prevenindu-se modificarea sau distrugerea lor înainte de încheierea compilării procedurii.
- Comenzile SQL de interogare (SELECT fără clauza FOR UPDATE) nu provoacă nici un fel de blocare.

Blocări - cont.

- Din punct de vedere al **resursei blocate**, blocările DML pot fi de 2 feluri:
  - **blocări la nivel de rând**, atunci când blocarea afectează un singur rând;
  - **blocări la nivel de tabel**, atunci când blocarea afectează întreg tabelul.
- Din punct de vedere al **modului de declanşare** a blocării, blocările DML sunt de 2 feluri:
  - **Implicite**, atunci când blocarea este făcută în mod automat de către Oracle în urma efectuării unor operaţii de INSERT, UPDATE sau DELETE şi nu necesită nici o acţiune din partea utilizatorului.
    Rândul asupra căruia se efectuează o astfel de operaţie este blocat pentru evitarea unor alte operaţii DML simultane asupra sa.

Blocări - cont.

- **Explicite**, atunci când ele apar ca urmare a executării de către utilizator a următoarelor comenzi SQL
  SELECT cu clauza FOR UPDATE
  LOCK TABLE
- O blocare a unei resurse este obţinută de către o tranzacţie, deci blocarea va fi eliberată la încheierea tranzacţiei.
  Deci toate blocările obţinute în timpul unei tranzacţii sunt eliberate atunci când tranzacţia este permanentizată sau derulată înapoi.
  În plus, toate blocările obţinute după un punct de salvare sunt eliberate atunci când tranzacţia este derulată înapoi până la acel punct de salvare.

Blocări la nivel de rând

- Blocările la nivel de rând apar în mod implicit la efectuarea unor operaţii de INSERT, UPDATE şi DELETE, cât şi în mod explicit la executarea comenzii SELECT cu clauza FOR UPDATE.
  În această secțiune ne vom referi la blocările implicite la nivel de rând, comanda SELECT ... FOR UPDATE urmând să fie prezentată mai târziu.

Blocări la nivel de rând

- În momentul efectuării unor operaţii de INSERT, UPDATE sau DELETE asupra datelor, rândul sau rândurile afectate de aceste operaţii sunt blocate.
  Blocarea se efectuează la nivel de rând, adică la nivelul cel mai de jos posibil, şi nu la nivel de tabel, asigurându-se astfel cel mai bun acces concurent posibil.
  Combinaţia între modelul multiversiune descris mai înainte şi blocările la nivel de rând asigură faptul că **utilizatorii nu intră în competiţie pentru date decât dacă încearcă accesarea aceluiaşi rând**.
  Mai precis, mecanismul folosit de Oracle pentru gestionarea concurenţei asigură următoarele:

Blocări la nivel de rând

- Operaţiile de citire (SELECT) nu trebuie să aştepte până la terminarea operaţiilor de scriere (INSERT, UPDATE, DELETE) sau a altor operaţii de citire efectuate asupra aceloraşi rânduri.
- Operaţiile de scriere nu trebuie să aştepte până la terminarea operaţiilor de citire efectuate asupra aceloraşi rânduri, cu excepţia situaţiei când acest lucru este cerut explicit de o comandă SELECT ... FOR UPDATE
- Operaţiile de scriere trebuie doar să aştepte pentru alte operaţii de scriere care încercă să modifice aceleaşi rânduri în tranzacţii concurente.

Blocări la nivel de rând

- La nivel de rând, blocările se pot face **numai în modul exclusiv (X)**, adică un utilizator nu poate modifica un rând până ce tranzacţia care l-a blocat s-a terminat (prin permanentizare sau derulare înapoi).
- Dacă o tranzacţie obţine o blocare pentru un rând, atunci ea obţine şi o blocare la nivel de tabel pentru tabelul corespunzător.
  O blocare la nivel de tabel este de asemenea necesară pentru a preveni operaţii DDL care ar interacţiona cu modificările de date din tranzacţia curentă.

Blocări la nivel de tabel

- O tranzacţie obţine o blocare la nivel de tabel în mod implicit atunci când asupra tabelului este executată una dintre comenzile INSERT, UPDATE sau DELETE sau în mod explicit, prin comenzile SELECT ... FOR UPDATE şi LOCK TABLE.
  Blocările la nivel de tabel au următoarele 2 scopuri: rezervarea accesului la tabel pentru tranzacţia curentă şi prevenirea de operaţii DDL care ar intra în conflict cu această tranzacţie – de exemplu, asupra unui tabel nu pot fi executate operaţiile ALTER sau DROP dacă există o tranzacţie neterminată care deţine asupra acestuia o blocare la nivel de tabel.

Blocări la nivel de tabel

- Blocările la nivel de tabel pot fi făcute în mai multe moduri, în funcţie de caracterul mai mult sau mai puţin restrictiv al blocării:
  - RS - row share
  - RX - row exclusive
  - S - share
  - SRX - share row exclusive
  - X - exclusive

Blocări la nivel de tabel

- Modul de blocare al unui tabel determină modurile în care alte tranzacţii pot bloca acelaşi tabel.
  Enumerarea de mai înainte este făcută în ordinea crescătoare a caracterului restrictiv al modului de blocare, de la cel mai puţin restrictiv (RS) la cel mai restrictiv (X).
  În continuare trecem în revistă fiecare dintre aceste moduri, arătând acţiunile care produc modul respectiv de blocare şi ce acţiuni sunt permise sau nu în alte tranzacţii concurente cu tranzacţia care deţine blocarea.

Mod de blocare RS la nivel de tabel

- O blocare în mod RS la nivel de tabel arată că tranzacţia care blochează tabelul a blocat rânduri din tabel şi intenţionează să le modifice.
  O blocare în mod RS se obţine la executarea comenzilor SELECT cu clauza FOR UPDATE şi LOCK TABLE cu opţiunea ROW SHARE.
  Modul de blocare RS este cel mai puţin restrictiv dintre toate modurile de blocare, permiţând gradul cel mai mare de acces concurent pentru un tabel.
- **Operaţii permise**: O blocare în mod RS permite acces (SELECT, INSERT, UPDATE, DELETE) concurent la tabel şi blocarea concurentă a tabelului de către altă tranzacţie în orice mod, în afara de cel X.
- **Operaţii nepermise**: O blocare în mod RS nu permite altor tranzacţii concurente de a bloca tabelul în mod X.

Mod de blocare RX la nivel de tabel

- O blocare în mod RX la nivel de tabel arată în general că tranzacţia care deţine blocarea a făcut una sau mai multe modificări asupra rândurilor din tabel.
  O blocare în mod RX este obţinută la executarea comenzilor DML (INSERT, UPDATE şi DELETE) şi a comenzii LOCK TABLE cu opţiunea ROW EXCLUSIVE.
  Blocarea în mod RX este ceva mai restrictivă decât blocarea în mod RS.
- **Operaţii permise**: O blocare în mod RX permite acces (SELECT, INSERT, UPDATE, DELETE) concurent la tabel şi blocarea concurentă a tabelului de către altă tranzacţie în modurile RS şi RX.
- **Operaţii nepermise**: O blocare în mod RX nu permite altor tranzacţii concurente de a bloca tabelul în modurile X, SRX şi S.

Mod de blocare S la nivel de tabel

- O blocare în mod S este obţinută la executarea comenzii LOCK TABLE cu opţiunea SHARE.
- **Operaţii permise**: O blocare în mod S permite altor tranzacţii doar interogarea (SELECT) tabelului şi blocarea sa în modurile S şi RS.
- **Operaţii nepermise**: O blocare în mod S nu permite altor tranzacţii concurente de a efectua operaţii de INSERT, UPDATE sau DELETE şi de a bloca tabelul în modurile SRX şi X.

Mod de blocare SRX la nivel de tabel

- Modul de blocare SRX este mai restrictiv decât cel S.
  O blocare în mod SRX este obţinută la executarea comenzii LOCK TABLE cu opţiunea SHARE ROW EXCLUSIVE.
- **Operaţii permise:** O blocare în mod SRX permite altor tranzacţii doar interogarea (SELECT) tabelului şi blocarea sa în modul RS.
  Asupra unui tabel nu poate deţine simultan blocări în modul SRX decât cel mult o singură tranzacţie.
- **Operaţii nepermise:** O blocare în mod SRX nu permite altor tranzacţii concurente de a efectua operaţii de INSERT, UPDATE sau DELETE şi de a bloca tabelul în orice mod în afara de RS.

Mod de blocare X la nivel de tabel

- Modul de blocare X este cel mai restrictiv mod de blocare.
  O blocare în mod X este obţinută la executarea comenzii LOCK TABLE cu opţiunea EXCLUSIVE.
- **Operaţii permise:** O blocare în mod X permite altor tranzacţii doar interogarea (SELECT) tabelului.
  Asupra unui tabel nu poate deţine simultan blocări în modul X decât cel mult o singură tranzacţie.
- **Operaţii nepermise:** O blocare în mod X nu permite altor tranzacţii concurente de a efectua operaţii de INSERT, UPDATE sau DELETE şi de a bloca tabelul în orice mod.
- Următorul tabel rezumă modul de blocare implicit precum şi posibilitatea existenţei simultane a mai multor tipuri de blocare la nivel de tabel, în urma executării comenzilor SQL corespunzătoare:

Comanda SQL	Mod de blocare implicit	Moduri de blocare permise simultan				
		RS	**RX**	**S**	**SRX**	**X**
SELECT fără clauza FOR UPDATE	Nici unul	Da	Da	Da	Da	Da
INSERT	RX	Da	Da	Nu	Nu	Nu
UPDATE	RX	Da*	Da*	Nu	Nu	Nu
DELETE	RX	Da*	Da*	Nu	Nu	Nu
SELECT cu clauza FOR UPDATE	RS	Da*	Da*	Da*	Da*	Nu
LOCK TABLE … în ROW SHARE MODE	RS	Da	Da	Da	Da	Nu
LOCK TABLE … în ROW EXCLUSIVE MODE	RX	Da	Da	Nu	Nu	Nu
LOCK TABLE … în SHARE MODE	S	Da	Nu	Da	Nu	Nu
LOCK TABLE … în ROW SHARE EXCLUSIVE MODE	SRX	Da	Nu	Nu	Nu	Nu
LOCK TABLE … în EXCLUSIVE MODE	X	Nu	Nu	Nu	Nu	Nu

Da* = Da, dacă nu există altă tranzacţie care deţine blocări la nivel de rând care intră în conflict cu blocarea la nivel de rând produsă de această comandă SQL; în caz contrar, tranzacţia va aştepta eliberarea acestor blocări la nivel de rând.

Blocări la nivel de tabel - cont.

- În general, blocarea implicită la nivel de rând este suficientă în aplicaţii.
  Totuşi, aşa cum am menţionat mai devreme, blocarea se poate face şi în mod implicit, folosind comanda SELECT cu clauza FOR UPDATE (la nivel de rând şi tabel) şi comanda LOCK TABLE (la nivel de tabel).
  Acestea sunt prezentate în următoarele 2 secţiuni.

Comanda SELECT cu clauza FOR UPDATE

- Folosirea clauzei FOR UPDATE într-o comandă SELECT determină blocarea rândurilor selectate în modul X şi blocarea întregului tabel sau tabelelor pe care se face interogarea în modul RS.
  Deci comanda SELECT cu clauza FOR UPDATE nu modifică rândurile selectate ci doar le blochează.
  Această comandă este foarte folositoare pentru “aducerea” unui rând mai devreme într-o tranzacţie, înainte de a-l actualiza.
  La actualizarea rândurilor (prin comanda UPDATE) blocarea la nivel de rând rămâne neschimbată în timp ce modul de blocare al tabelului devine RX.

Comanda SELECT cu clauza FOR UPDATE

- Sintaxa acestei comenzi este următoarea:

```
SELECT lista_coloane
FROM tabel [,tabel] ...
WHERE condiţie
FOR UPDATE [OF coloana [,coloana] ...] [NOWAIT]
```

- Dacă se specifică NOWAIT şi rândul sau rândurile selectate sunt deja blocate de altă tranzacţie, atunci utilizatorul este înştiinţat de acest lucru, returnându-i-se controlul. Dacă NOWAIT nu este specificat, atunci comanda aşteaptă până când rândul este deblocat.
  Să luăm de exemplu, tranzacţia următoare:

Comanda SELECT cu clauza FOR UPDATE

```
SELECT salariu
FROM salariat
WHERE cod_salariat = 102
FOR UPDATE OF salariu NOWAIT;

UPDATE salariat
SET salariu = 3500
WHERE cod_salariat = 102;

COMMIT;
```

Comanda SELECT cu clauza FOR UPDATE

- Atunci, la executarea primei comenzi, rândul cu cod_salariat = 2 este blocat în mod X în timp ce tabelul salariat este blocat în mod RS.
  La executarea celei de-a 2 comenzi, blocarea la nivel de rând se menţine în timp ce blocarea la nivel de tabel este schimbată în modul RX.
  La executarea instrucţiunii COMMIT, tranzacţia este permanentizată şi toate blocările sunt eliberate.

Comanda LOCK TABLE

- Unul sau mai multe tabele pot fi blocate în oricare din modurile prezentate anterior folosind comanda LOCK TABLE, având sintaxa:

```
LOCK TABLE nume_tabel [, nume_tabel] ...
IN mod_blocare MODE [NOWAIT]
```

unde mod_blocare poate avea valorile ROW SHARE, ROW EXCLUSIVE, SHARE, ROW SHARE EXCLUSIVE sau EXCLUSIVE.
Folosirea lui NOWAIT este opţională şi are aceeaşi semnificaţie ca în cazul comenzii SELECT FOR UPDATE.

Comanda LOCK TABLE

În principal, blocarea manuală a unui tabel folosind comanda LOCK TABLE poate fi preferată în următoarele situaţii:

- Este necesară o “vedere” consistentă asupra mai multor tabele. În acest caz, tabelele pot fi blocate în modul S, împiedicând operaţiile DML asupra acestuia.
- Se doreşte împiedicarea altor utilizatori de a bloca tabelele asupra cărora operează tranzacţia curentă. Acest lucru se poate face blocând tabelele în modul X.
- Se doreşte ca o instrucţiune sa nu aştepte pentru deblocarea unei resurse. În acest caz se poate folosi opţiunea NOWAIT.

Următorul tabel rezumă tipurile de blocare la nivel de rând şi tabel obţinute la executarea diverselor comenzi SQL.

Comanda SQL	**Blocare la nivel de rând**	**Mod de blocare La nivel de tabel**
SELECT fără clauza FOR UPDATE		
INSERT	X	RX
UPDATE	X	RX
DELETE	X	RX
SELECT cu clauza FOR UPDATE	X	RS
LOCK TABLE … în ROW SHARE MODE		RS
LOCK TABLE … în ROW EXCLUSIVE MODE		RX
LOCK TABLE … în SHARE MODE		S
LOCK TABLE … în ROW SHARE EXCLUSIVE MODE		RSX
LOCK TABLE … în EXCLUSIVE MODE		X

Interblocarea

- Datorită accesului concurent la date este posibil ca mai mulţi utilizatori să se blocheze reciproc.
  Această situaţie se numeşte interblocare (deadlock), pentru că **fiecare dintre utilizatori aşteaptă ca celălalt să elibereze resursa blocată**.
  În cazul acesta problema nu se poate rezolva prin simpla aşteptare, una din tranzacţii trebuind să fie derulată înapoi.
  Să luăm, de exemplu, următoarele 2 tranzacţii care se execută simultan:

Interblocarea

Tranzacţia A	Tranzacţia B

`UPDATE salariat`
`SET salariu = 3500`
`WHERE cod_salariat = 102;`
 	
`UPDATE departament`
`SET localitate = 'Ploiesti'`
`WHERE cod_dept = 2 AND`
`cod_tara = 40;`
	`UPDATE departament`
	`SET nume_dept = 'Proiectare'`
	`WHERE cod_dept = 2`
	`AND cod_tara = 40;`
	
	`UPDATE salariat`
	`SET manager = 101`
	`WHERE cod_salariat = 102;`

Interblocarea

- La executarea primelor instrucţiuni din fiecare tranzacţie nu este nici o problemă.
  Când însă tranzacţiile încercă să obţină blocări pentru instrucţiunile respective se va ajunge la interblocare deoarece tranzacţia A trebuie să aştepte ca tranzacţia B să deblocheze rândul din tabelul departament în timp ce tranzacţia B trebuie să aştepte ca tranzacţia A să deblocheze rândul din tabelul salariat.
  În general o interblocare poate să fie cauzată de mai mult de 2 tranzacţii, de exemplu tranzacţia 2 aşteaptă după tranzacţia 1, tranzacţia 3 după tranzacţia 2, etc., iar tranzacţia 1 aşteptă după tranzacţia n.

Detectarea interblocării

- Oracle detectează interblocările în mod automat.
  În acest caz, Oracle semnalează o eroare uneia dintre tranzacţiile implicate şi derulează înapoi ultima instrucţiune din această tranzacţie.
  Acest lucru rezolvă interblocarea, deşi cealaltă tranzacţie poate încă să aştepte până la deblocarea resursei pentru care aşteaptă.
  De obicei, tranzacţia semnalată trebuie derulată înapoi în mod explicit.

Evitarea interblocării

- Interblocările pot fi de obicei evitate dacă tranzacţiile care accesează aceleaşi tabele blochează aceste tabele în aceeaşi ordine, prin blocare implicită sau explicită.
  De exemplu, putem impune regula ca, atunci când este accesat atât un tabel master cât şi un tabel detaliu, să fie blocat întâi tabelul master şi după aceea cel de detaliu.
  Dacă astfel de reguli sunt bine alcătuite şi aplicate, atunci probabilitatea de apariţie a interblocărilor este foarte rară.

Bibliografie

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