

CUPRINS

8. <i>PL/SQL – Trigger-i</i>	2
8.1. <i>Trigger-i LMD</i>	5
8.1.1. <i>Trigger-i LMD la nivel de comandă</i>	7
8.1.2. <i>Trigger-i LMD la nivel de linie</i>	7
8.1.3. <i>Ordinea de execuție a trigger-ilor LMD</i>	8
8.1.4. <i>Predicate condiționale</i>	11
8.2. <i>Trigger-i INSTEAD OF</i>	12
8.3. <i>Trigger-i sistem</i>	13
8.3.1. <i>Trigger-i pentru comenzi LDD</i>	14
8.3.2. <i>Trigger-i pentru evenimente sistem</i>	14
8.4. <i>Modificarea și ștergerea trigger-ilor</i>	15
8.5. <i>Informații despre trigger-i</i>	16
8.6. <i>Privilegii sistem</i>	17
8.7. <i>Tabele mutating</i>	18
Bibliografie	19

8. PL/SQL – Trigger-i

- Un *trigger* (declanșator) este un bloc *PL/SQL* cu nume, stocat în baza de date (independent), care se execută automat ori de câte ori are loc evenimentul „declanșator” de care este asociat.
- Evenimentul declanșator poate consta din:
 - modificarea unui tabel sau a unei vizualizări;
 - acțiuni sistem;
 - acțiuni utilizator.
- Un *trigger* poate fi asociat cu un eveniment care are loc asupra unui tabel, unei vizualizări, unei scheme sau unei baze de date.



❖ Față de procedurile stocate *trigger*-ii:

- pot fi activați sau dezactivați;
- nu pot fi invocați explicit;
- nu pot conține comenzile *COMMIT*, *SAVEPOINT* sau *ROLLBACK*.

❖ Un *trigger* activ (*enable*) va fi invocat automat de către sistem ori de câte ori are loc evenimentul asociat acestuia.

❖ Un *trigger* dezactivat (*disable*) nu va fi declanșat, chiar dacă evenimentul asociat are loc.



În mod asemănător pachetelor, *trigger*-i nu pot fi definiți local în blocuri *PL/SQL* sau pachete.



❖ Ca și în cazul subprogramelor independente sau al pachetelor, atunci când un *trigger* este depus în dicționarul datelor alături de codul sursă este depus și *p-codul* compilat.

❖ Dacă *trigger*-ul este valid, atunci va fi apelat fără recompilare.

❖ *Trigger*-ii pot fi invalidați în aceeași manieră ca pachetele și subprogramele.

Dacă *trigger*-ul este invalidat, acesta va fi recompilat la următoarea activare.



Când utilizăm *trigger*-i?

Atunci când dorim ca efectuarea unei anumite operații să implice întotdeauna execuția unor acțiuni asociate.



- ❖ Nu trebuie definiți *trigger*-i care duplică sau înlocuiesc acțiuni ce pot fi implementate mai simplu. De exemplu, nu are sens să fie definiți *trigger*-i care să implementeze regulile de integritate ce pot fi definite prin constrângeri declarative.
 - ❖ Utilizarea excesivă a *trigger*-ilor poate determina interdependențe complexe ce pot fi dificil de menținut.
 - ❖ Atunci când sunt definiți *trigger*-i trebuie să se țină cont de recursivitate și de efectele în cascadă.
- *Trigger*-ii pot fi definiți la nivel de:
 - aplicație (*application triggers*);
 - bază de date (*database triggers*).

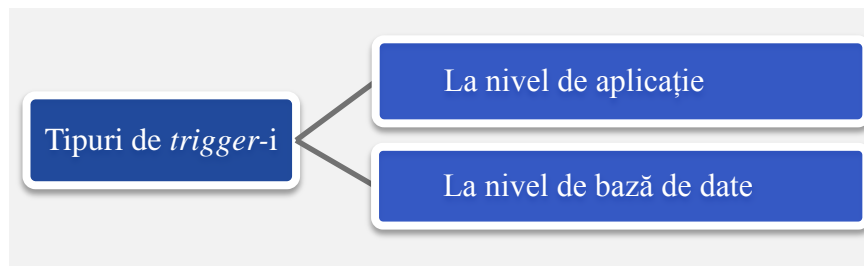


Fig. 8.1. Tipuri de *trigger*-i

Trigger-i aplicație

- Se pot executa automat ori de câte ori apare un anumit eveniment într-o aplicație (de exemplu, o aplicație dezvoltată cu *Developer Suite*).
 - *Form Builder* (utilitar *Developer Suite*) utilizează frecvent acest tip de *trigger*-i (*form builder triggers*). Aceștia pot fi declanșați prin apăsarea unui buton, prin navigarea pe un anumit câmp etc.

Trigger-i bază de date

- Se pot executa automat ori de câte ori are loc:
 - o comandă *LMD* asupra datelor unui tabel;
 - o comandă *LMD* asupra datelor unei vizualizări;
 - o comandă *LDD* (*CREATE*, *ALTER*, *DROP*) referitoare la anumite obiecte ale schemei sau ale bazei de date;
 - un eveniment sistem (*SHUTDOWN*, *STARTUP*);

- o acțiune a utilizatorului (*LOGON*, *LOGOFF*);
- o eroare (*SERVERERROR*, *SUSPEND*).

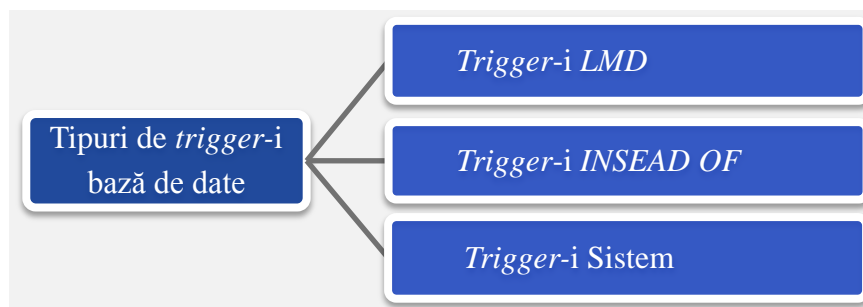


Fig. 8.2. Tipuri de *trigger*-i bază de date

- *Trigger*-ii bază de date sunt de trei tipuri:
 - *trigger*-i *LMD*
 - sunt activați de comenzi *LMD* (*INSERT*, *UPDATE* sau *DELETE*) executate asupra unui tabel al bazei de date
 - *trigger*-i *INSTEAD OF*
 - sunt activați de comenzi *LMD* executate asupra unei vizualizări (relaționale sau obiect)
 - *trigger*-i sistem
 - sunt activați de un eveniment sistem (oprirea sau pornirea bazei), de comenzi *LDD* (*CREATE*, *ALTER*, *DROP*), de conectarea/deconectarea unui utilizator
 - sunt definiți la nivel de schemă sau la nivel de bază de date



- ❖ *Trigger*-ii asociați unui tabel vor acționa indiferent de aplicația care a efectuat operația *LMD*.
- ❖ Dacă operația *LMD* se referă la o vizualizare, *trigger*-ul *INSTEAD OF* definește acțiunile care vor avea loc, iar dacă aceste acțiuni includ comenzi *LMD* referitoare la tabele, atunci *trigger*-ii asociați acestor tabele sunt și ei, la rândul lor, activați.
- ❖ *Trigger*-ii asociați unei baze de date se declanșează pentru fiecare eveniment, pentru toți utilizatorii.
- ❖ *Trigger*-ii asociați unei scheme sau unui tabel se declanșează numai dacă evenimentul declanșator implică acea schemă sau acel tabel.



În acest capitol se face referire doar la *trigger*-ii bază de date.

8.1. Trigger-i LMD

- Sunt activați de comenzi *LMD* (*INSERT*, *UPDATE* sau *DELETE*) executate asupra unui tabel al bazei de date.
- În comanda de creare a unui *trigger* pot fi precizate mai multe comenzi declanșatoare diferite, dar care se execută asupra unui singur tabel.
- Sintaxa:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_trigger
--momentul când este declanșat
{BEFORE | AFTER}
--comanda/comenzile care îl declanșează
{ DELETE|INSERT|UPDATE [OF coloana[, coloana ...] ] }
[OR {DELETE|INSERT|UPDATE [OF coloana[, coloana ...] ] ...}
ON [schema.]nume_tabel

[REFERENCING {OLD [AS] vechi NEW [AS] nou
               | NEW [AS] nou OLD [AS] vechi } ]
[FOR EACH ROW]
[WHEN (condiție) ]

corp_trigger (bloc anonim PL/SQL sau comanda CALL);
```

- Numele *trigger*-ului:
 - trebuie să fie unic printre numele *trigger*-ilor din cadrul aceleiași scheme;
 - poate să coincidă cu numele altor obiecte ale schemei în care este definit (de exemplu, tabele, vizualizări sau proceduri).
- Momentul declanșării *trigger*-ului:
 - reprezintă momentul în care va fi executat corpul *trigger*-ului;
 - poate fi înainte (*BEFORE*) sau după (*AFTER*) comanda declanșatoare;
- Comanda declanșatoare:
 - poate fi specificată o singură comandă *LMD* (*INSERT*, *DELETE* sau *UPDATE*) sau o combinație disjunctivă celor trei comenzi *LMD* (folosind operatorul logic *OR*);
 - poate fi urmată de o enumerare a coloanelor a căror actualizare va declanșa *trigger*-ul (comanda declanșatoare este un *UPDATE*).

- Tabelul asupra căruia este executată comanda declanșatoare poate fi:
 - un tabel (*table*);
 - un tablou imbricat (*nested table*).
- Valorile coloanelor înainte și după modificarea unei linii:
 - Înainte de modificare valoarea unei coloane este referită prin atributul *OLD*, iar după modificare este referită prin atributul *NEW*.
 - În interiorul blocului *PL/SQL*, coloanele prefixate prin *OLD* sau *NEW* sunt considerate variabile externe, deci trebuie prefixate cu ":".
- Clauza *REFERENCING* permite redenumirea atributelor *NEW* și *OLD*.
- Clauza *FOR EACH ROW* declară un *trigger* la nivel de linie.
 - Lipsa acestei clauze determină definirea unui *trigger* la nivel de instrucțiune.
- Clauza *WHEN* precizează o expresie *booleană* care este verificată pentru fiecare linie corespunzătoare din tabel.
 - Este valabilă doar pentru *trigger*-ii la nivel de linie.
- Corpul *trigger*-ului
 - nu poate depăși 32KB;
 - pentru a evita depășirea dimensiunii maxime se pot defini proceduri stocate ce pot fi invocate din corpul *trigger*-ului;
 - poate consta dintr-un bloc *PL/SQL*;
 - poate consta dintr-o singură comandă *CALL*:
 - comanda *CALL* poate apela un subprogram *PL/SQL* stocat, o rutină *C* sau o metodă *Java*;
 - în acest caz, comanda *CALL* nu poate conține clauza *INTO* care este specifică funcțiilor;
 - pentru a referi coloanele tabelului asociat *trigger*-ului, acestea trebuie prefixate de atributele *:NEW* sau *:OLD*;
 - în expresia parametrilor nu pot să apară variabile *bind*.



- ❖ Un *trigger* poate activa alt *trigger*, iar acesta la rândul său poate activa alt *trigger* etc (*trigger*-i în cascadă).
- ❖ Sistemul permite maximum 32 de *trigger*-i în cascadă.
- ❖ Numărul acestora poate fi limitat (utilizând parametrul de inițializare *OPEN_CURSORS*), deoarece pentru fiecare execuție a unui *trigger* trebuie deschis un nou cursor.

- *Trigger-ii LMD* pot fi:
 - la nivel de comandă (*statement level trigger*);
 - la nivel de linie (*row level trigger*).

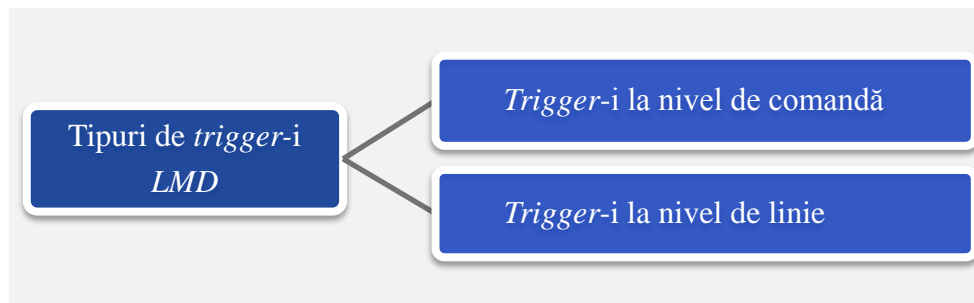


Fig. 8.3. Tipuri de *trigger-i LMD*

8.1.1. *Trigger-i LMD* la nivel de comandă

- Sunt executați o singură dată pentru comanda declanșatoare, indiferent de numărul de linii afectate (chiar dacă nicio linie nu este afectată).
- Sunt utilizați atunci când acțiunea *trigger*-ului nu depinde de informațiile menținute în liniile afectate de comandă.

Exemplul 8.1 – **vezi curs**

Exemplul 8.2 – **vezi curs**

8.1.2. *Trigger-i LMD* la nivel de linie

- Sunt creați atunci când în comanda *CREATE TRIGGER* este precizată clauza *FOR EACH ROW*.
 - Lipsa acestei clauze determină definirea unui *trigger* la nivel de instrucțiune.
- Sunt executați pentru fiecare linie afectată de instrucțiunea declanșatoare.
 - Dacă instrucțiunea declanșatoare nu afectează nicio linie, atunci nu sunt executați.
- Restricțiile acestor tipuri de *trigger-i* pot fi specificate în clauza *WHEN* (printr-o expresie *booleană*).
 - Expresia *booleană* este evaluată de *trigger* pentru fiecare linie afectată de comanda declanșatoare.
 - *Trigger*-ul este executat pentru o anumită linie, doar dacă expresia este adevărată pentru acea linie.
 - În expresia *booleană* nu sunt permise funcții definite de utilizator sau subcereri *SQL*.



- ❖ *Trigger*-ii la nivel linie nu sunt performanți dacă se fac frecvent reactualizări pe tabele de dimensiuni foarte mari.

Exemplul 8.3 – vezi curs

- Accesul la vechile, respectiv noile valori ale coloanelor liniei curente, afectată de evenimentul ce a declanșat *trigger*-ul, se realizează prin:
 - OLD.nume_coloană
 - NEW.nume_coloană



- ❖ În interiorul blocului *PL/SQL*, coloanele tabelului prefixate cu *OLD* sau *NEW* sunt considerate variabile externe și deci, trebuie precedate de caracterul „:”.
- ❖ În expresia *booleană* din clauza *WHEN* nu trebuie utilizată prefixarea cu „:” pentru *OLD* sau *NEW*.

- În cazul celor trei comenzi *LMD*, aceste valori devin:

COMANDA	NEW.nume_coloană	OLD.nume_coloană
INSERT	noua valoare	null
DELETE	null	vechea valoare
UPDATE	noua valoare	vechea valoare

Exemplul 8.4 – vezi curs

8.1.3. Ordinea de execuție a *trigger*-ilor *LMD*

- *PL/SQL* permite definirea a 12 tipuri de *trigger*-i *LMD* care sunt obținuți prin combinarea proprietăților ce pot fi specificate în comanda de definire a acestora:
 - momentul declanșării (*BEFORE* sau *AFTER*);
 - nivelul la care acționează (nivel comandă sau nivel linie – *FOR EACH ROW*);
 - comanda declanșatoare (*INSERT*, *UPDATE* sau *DELETE*).



- ❖ De exemplu, un *trigger BEFORE INSERT* acționează o singură dată, înaintea execuției unei instrucțiuni *INSERT*, iar un *trigger BEFORE INSERT FOR EACH ROW* acționează înainte de inserarea fiecărei noi înregistrări.

- O comandă declanșatoare sau o comandă din corpul unui *trigger* poate determina verificarea mai multor constrângeri de integritate. De asemenea, *trigger*-ii pot conține comenzi care pot determina declanșarea altor *trigger*-i (*trigger*-i în cascadă).
- Pentru a menține o secvență adecvată de declanșare a *trigger*-ilor și de verificare a constrângerilor de integritate, sistemul *Oracle* utilizează următorul model de execuție a *trigger*-lor *LMD* multipli:
 1. Se execută toți *trigger*-ii *BEFORE* comandă care sunt declanșați de comanda *LMD*.
 2. Pentru fiecare linie afectată de comanda *LMD*:
 - 2.1. se execută toți *trigger*-ii *BEFORE* linie care sunt declanșați de comanda *LMD*;
 - 2.2. se blochează și se modifică linia afectată de comanda *LMD* (se rulează comanda *LMD*); se verifică constrângerile de integritate (blocarea rămâne valabilă până în momentul în care tranzacția este permanentizată);
 - 2.3. se execută toți *trigger*-ii *AFTER* linie care sunt declanșați de comanda *LMD*;
 3. Se execută toți *trigger*-ii *AFTER* comandă care sunt declanșați de comanda *LMD*.
- Definiția modelului de execuție este recursivă.
 - De exemplu, o comandă *SQL* poate determina execuția unui *trigger BEFORE* linie și verificarea unei constrângeri de integritate. Acel *trigger BEFORE* linie poate realiza o actualizare (*UPDATE*) care la rândul său determină verificarea unei constrângeri de integritate și execuția unui *trigger AFTER* comandă. *Trigger*-ul *AFTER* comandă determină la rândul său o verificare a unei constrângeri de integritate.
 - În acest caz, modelul de execuție rulează pașii recursiv, după cum urmează:

Este lansată comanda *SQL* declanșatoare.

 1. Se execută *trigger*-ii *BEFORE* linie.
 - 1.1. Se execută *trigger*-ii *AFTER* comandă declanșați de comanda *UPDATE* din corpul *trigger*-ului *BEFORE* linie.
 - 1.1.1. Se execută comenzile din corpul *trigger*-ilor *AFTER* comandă.
 - 1.1.2. Se verifică dacă sunt îndeplinite constrângerile de integritate definite pentru tabelele modificate de *trigger*-ii *AFTER* comandă.
 - 1.2. Se execută comenzile din corpul *trigger*-ilor *BEFORE* linie.
 - 1.3. Se verifică dacă sunt îndeplinite constrângerile de integritate definite pentru tabelele modificate de *trigger*-ii *BEFORE* linie.

2. Se execută comanda *SQL*.
 3. Se verifică dacă sunt îndeplinite constrângerile de integritate ce ar putea fi încălcate de comanda *SQL*.
- Există două excepții privind recursivitatea.
 - Atunci când o comandă modifică un tabel (cheia primară sau cheia externă) care face parte dintr-o constrângere referențială și declanșează un *trigger* ce modifică celălalt tabel referit în constrângere, doar comanda declanșatoare va determina verificarea constrângerii de integritate. Astfel, se permite *trigger*-ilor la nivel de linie să forțeze integritatea referențială.
 - *Trigger*-ii la nivel de comandă declanșați datorită opțiunilor *DELETE CASCADE* și *DELETE SET NULL* sunt executați înainte și după lansarea comenzii *DELETE* de către utilizator. În acest mod se previne apariția erorilor *mutating*.



- ❖ O proprietate importantă a modelului de execuție a *trigger*-ilor este acela că toate acțiunile și verificările realizate datorită execuției unei comenzii declanșatoare trebuie să se termine cu succes.
- ❖ Dacă apare o excepție în interiorul unui *trigger* și aceasta nu este explicit tratată, atunci toate acțiunile realizate ca rezultat al comenzii declanșatoare, incluzând toate acțiunile realizate de *trigger*-ii declanșați de comandă vor fi anulate (*rollback*).
- ❖ În acest mod, *trigger*-i nu pot compromite constrângerile de integritate.



- ❖ *Trigger*-ii de tipuri diferite sunt executați într-o ordine specifică.
- ❖ *Trigger*-ii de același tip definiți pentru aceeași cerere nu au garantată o ordine specifică. De exemplu, toți *trigger*-ii *BEFORE* linie definiți pentru o singură comandă *LMD* nu sunt declanșați mereu în aceeași ordine. Din acest motiv, în aplicații nu se recomandă utilizarea mai multor *trigger*-i de același tip care sunt declanșați de aceeași comandă.



- ❖ *Trigger*-ii bază de date pot fi definiți numai pe tabele (excepție, *trigger*-ul *INSTEAD OF* care este definit pe o vizualizare).
- ❖ Totuși, dacă o comandă *LMD* este aplicată unei vizualizări, pot fi activați *trigger*-ii asociați tabelor care definesc vizualizarea.



Restricții:

- ❖ În versiunile *Oracle* anterioare corpul unui *trigger* nu poate conține o interogare sau o reactualizare a unui tabel aflat în plin proces de modificare, pe timpul acțiunii *trigger*-ului (*mutating table*). Începând cu versiunea *Oracle 11g* această problemă este rezolvată prin utilizarea *trigger*-ilor compuși (*compound triggers*).
- ❖ Blocul *PL/SQL* care descrie acțiunea *trigger*-ului nu poate conține comenzi pentru gestiunea tranzacțiilor (*COMMIT*, *ROLLBACK*, *SAVEPOINT*).
 - Controlul tranzacțiilor este permis, însă, în procedurile stocate.
 - Dacă un *trigger* apelează o procedură stocată care execută o comandă referitoare la controlul tranzacțiilor, atunci va apărea o eroare la execuție și tranzacția va fi anulată.
- ❖ Comenzile *LDD* nu pot să apară decât în *trigger*-ii sistem.
- ❖ În corpul *trigger*-ului pot fi referite și utilizate coloane *LOB*, dar nu pot fi modificate valorile acestora.
- ❖ În corpul *trigger*-ului se pot insera date în coloanele de tip *LONG* și *LONGRAW*, dar nu pot fi declarate variabile de acest tip.
- ❖ Dacă un tabel este șters, automat sunt șterși toți *trigger*-ii asociați acestuia.

8.1.4. Predicate condiționale

- În interiorul unui *trigger* care poate fi executat pentru diferite tipuri de instrucțiuni *LMD* se pot folosi trei funcții *booleene* (din pachetul *DBMS_STANDARD*) prin care se stabilește tipul operației executate:
 - *INSERTING* – întoarce valoarea *TRUE* atunci când comanda declanșatoare este o comandă *INSERT*;
 - *DELETING* – întoarce valoarea *TRUE* atunci când comanda declanșatoare este o comandă *DELETE*;
 - *UPDATING* – întoarce valoarea *TRUE* atunci când comanda declanșatoare este o comandă *UPDATE*;
 - *UPDATING*('nume_coloană') întoarce valoarea *TRUE* atunci când comanda declanșatoare este o comandă *UPDATE* asupra coloanei *nume_coloană*.



- ❖ Utilizând aceste predicate, în corpul *trigger*-ului se pot executa secvențe de instrucțiuni diferite, în funcție de tipul operației *LMD* declanșatoare.

Exemplul 8.5 – vezi curs

8.2. Trigger-i *INSTEAD OF*

- Oferă o modalitate de actualizare a vizualizărilor obiect și a celor relaționale.
- Sintaxa:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_trigger
--momentul când este declanșat
    INSTEAD OF
--comanda/comenzile care îl declanșează
    { DELETE|INSERT|UPDATE [OF coloana[, coloana ...] ] }
    [OR {DELETE|INSERT|UPDATE [OF coloana[, coloana ...] ] ...}
    ON [NESTED TABLE (nume_coloană) OF]
        [schema.]nume_vizualizare |
    [REFERENCING {OLD [AS] vechi NEW [AS] nou
                  | NEW [AS] nou OLD [AS] vechi } ]
FOR EACH ROW
[WHEN (condiție) ]

corp_trigger (bloc anonim PL/SQL sau comanda CALL);
```

- *Trigger*-ul *INSTEAD OF* permite reactualizarea unei vizualizări prin comenzi *LMD*.
- Datorită regulilor stricte existente pentru reactualizarea vizualizărilor, nu orice tip de vizualizare permite reactualizări *LMD*.
 - De exemplu, o vizualizare care este definită pe baza *join*-ului mai multor tabele nu permite reactualizarea tuturor acestor tabelelor.
 - O vizualizare nu poate fi modificată prin comenzi *LMD* dacă aceasta conține operatori pe mulțimi, funcții grup, clauzele *GROUP BY*, *CONNECT BY*, *START WITH* sau operatorul *DISTINCT*.
- *Trigger*-ul *INSTEAD OF* este utilizat pentru a executa operații *LMD* direct pe tabelele de bază ale vizualizării.
 - Comenzile *LMD* lansate asupra unei vizualizări, sunt preluate de *trigger*-ul *INSTEAD OF* (care poate lansa comenzile direct pe tabelele de bază).

- *Trigger*-ul *INSTEAD OF* poate fi definit asupra vizualizărilor ce au drept câmpuri tablouri imbricate, *trigger*-ul furnizând o modalitate de reactualizare a elementelor tabloului imbricat.
 - În acest caz, *trigger*-ul se declanșează doar în cazul în care comenzile *LMD* operează asupra tabloului imbricat (numai când elementele tabloului imbricat sunt modificate folosind clauzele *THE()* sau *TABLE()*) și nu atunci când comanda *LMD* operează doar asupra vizualizării.
 - Atributele *:OLD* și *:NEW* se referă la liniile tabloului imbricat, iar pentru a referi linia curentă din tabloul „părinte” s-a introdus atributul *:PARENT*.



- ❖ Spre deosebire de *trigger*-ii *LMD*, *trigger*-ii *INSTEAD OF* se execută în locul instrucțiunii *LMD* (*INSERT*, *UPDATE*, *DELETE*) specificate.
- ❖ Opțiunea *UPDATE OF* nu este permisă pentru acest tip de *trigger*.
- ❖ *Trigger*-ii *INSTEAD OF* se definesc pentru o vizualizare, nu pentru un tabel.
- ❖ *Trigger*-ii *INSTEAD OF* acționează la nivel de linie.

Exemplul 8.6 – vezi curs

8.3. *Trigger*-i sistem

- Pot fi definiți la nivelul:
 - bazei de date;
 - schemei.
- Sunt declanșați de:
 - comenzi *LDD* (*CREATE*, *DROP*, *ALTER*);
 - evenimente sistem (*STARTUP*, *SHUTDOWN*, *LOGON*, *LOGOFF*, *SUSPEND*, *SERVERERROR*).
- Sintaxa:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_trigger
{BEFORE | AFTER}
{comenzi_LDD | evenimente_sistem}
ON {DATABASE | SCHEMA}
[WHEN (condiție) ]
corp_trigger;
```

- Pentru *trigger*-ii sistem se pot utiliza funcții speciale care permit obținerea de informații referitoare la evenimentul declanșator.

- Sunt funcții *PL/SQL* stocate care trebuie prefixate de numele proprietarului (*SYS*):
 - *SYSEVENT* – întoarce evenimentul sistem care a declanșat *trigger*-ul (este de tip *VARCHAR2(20)* și este aplicabilă oricărui eveniment);
 - *DATABASE_NAME* – întoarce numele bazei de date curente (este de tip *VARCHAR2(50)* și este aplicabilă oricărui eveniment);
 - *SERVER_ERROR* – întoarce codul erorii a cărei poziție în stiva erorilor este dată de argumentul de tip *NUMBER* al funcției (este de tip *NUMBER* și este aplicabilă evenimentului *SERVERERROR*);
 - *LOGIN_USER* – întoarce identificadorul utilizatorului care a declanșat *trigger*-ul (este de tip *VARCHAR2(30)* și este aplicabilă oricărui eveniment);
 - *DICTIONARY_OBJ_NAME* – întoarce numele obiectului referit de comanda *LDD* care a declanșat *trigger*-ul (este de tip *VARCHAR2(30)* și este aplicabilă evenimentelor *CREATE*, *ALTER*, *DROP*).

8.3.1. *Trigger-i pentru comenzi LDD*

- Sunt declanșați de comenzi *LDD* (*CREATE*, *ALTER*, *DROP*).
 - *ON DATABASE* determină declanșarea *trigger*-ului de comenzi *LDD* aplicate asupra obiectelor din orice schemă a bazei de date;
 - *ON SCHEMA* determină declanșarea *trigger*-ului de comenzi *LDD* aplicate asupra obiectelor din schema personală.
- Există restricții asupra expresiilor din condiția clauzei *WHEN*.
 - De exemplu, *trigger*-ii *LDD* pot verifica tipul și numele obiectelor definite, identificadorul și numele utilizatorului.

Exemplul 8.7 – vezi curs

8.3.2. *Trigger-i pentru evenimente sistem*

- Sunt declanșați de anumite evenimente sistem (*STARTUP*, *SHUTDOWN*, *LOGON*, *LOGOFF*, *SUSPEND*, *SERVERERROR*).
 - *ON DATABASE* determină declanșarea *trigger*-ului de evenimente sistem pentru orice schemă a bazei de date;
 - *ON SCHEMA* determină declanșarea *trigger*-ului de evenimente sistem din schema personală.

- Există restricții asupra expresiilor din condiția clauzei *WHEN*.
 - De exemplu, *trigger*-ii *LOGON* și *LOGOFF* pot verifica doar identificatorul (*userid*) și numele utilizatorului (*username*).

Exemplul 8.8 – vezi curs

Exemplul 8.9 – vezi curs

Exemplul 8.10 – vezi curs

8.4. Modificarea și ștergerea *trigger*-ilor

Ștergerea *trigger*-ilor

- Similar procedurilor și pachetelor, un *trigger* poate fi:
 - șters folosind comanda

```
DROP TRIGGER [schema.]nume_trigger;
```
 - recreat folosind clauza *OR REPLACE* din cadrul comenzii *CREATE TRIGGER*
 - clauza permite schimbarea definiției unui *trigger* existent fără ștergerea acestuia.

Modificarea *trigger*-ilor

- Modificarea unui *trigger* poate consta din:
 - recompilare (*COMPILE*);
 - redenumire (*RENAME*);
 - dezactivare (*DISABLE*);
 - activare (*ENABLE*).
- Sintaxa:

```
ALTER TRIGGER [schema.]nume_trigger  
{ENABLE | DISABLE | COMPILE | RENAME TO nume_nou};
```
- Uneori în loc de ștergerea unui *trigger* este preferabilă doar dezactivarea sa temporară.
 - Un *trigger* este activat implicit atunci când acesta este creat.
 - Un *trigger* dezactivat continuă să existe ca obiect în dicționarul datelor, dar este nu va mai fi executat de sistem (până când nu este din nou activat).
- Activarea, respectiv dezactivarea tuturor *trigger*-ilor asociați unui tabel se poate realiza utilizând comanda următoare:

```
ALTER TABLE [schema.]nume_tabel  
{ENABLE | DISABLE } ALL TRIGGERS;
```



- ❖ Comanda *ALTER TRIGGER* permite activarea, respectiv dezactivarea unui singur *trigger* (al cărui nume este specificat în comandă).
- ❖ Comanda *ALTER TABLE* permite activarea, respectiv dezactivarea tuturor *trigger*-ilor asociați unui tabel.



- ❖ Comanda *DROP TRIGGER* permite ștergerea unui singur *trigger* (al cărui nume este specificat în comandă).
- ❖ Comanda *DROP TABLE* determină implicit ștergerea tuturor *trigger*-ilor asociați unui tabel.

8.5. Informații despre *trigger*-i

- Vizualizări din dicționarul datelor ce conțin informații despre *trigger*-i:
 - *USER_OBJETS*
 - atunci când este creat un *trigger* în vizualizare apare o linie nouă cu informații despre acesta (nume, tip, id, data creării, data ultimei modificări, stare etc);
 - tipul obiectului creat este *trigger*.
 - *USER_TRIGGERS*
 - conține informații complete despre *trigger* (codul sursă detaliat, starea);
 - *USER_TRIGGER_COLS*
 - conține informații despre coloanele utilizate în *trigger*;
 - *USER_ERRORS*
 - conține informații despre erorile apărute la compilare *trigger*-ului;
 - *USER_DEPENDENCIES*
 - este utilizată pentru a determina dependențele *trigger*-ilor.

Vizualizarea *USER_TRIGGERS*

- Vizualizarea include următoarele informații:
 - numele *trigger*-ului (*TRIGGER_NAME*);
 - tipul *trigger*-ului (*TRIGGER_TYPE*);
 - evenimentul declanșator (*TRIGGERING_EVENT*);

- numele proprietarului tabelului (*TABLE_OWNER*);
- numele tabelului pe care este definit *trigger*-ul (*TABLE_NAME*);
 - dacă obiectul referit de *trigger* nu este un tabel sau o vizualizare, atunci *TABLE_NAME* este are valoarea *null*;
- clauza *WHEN* (*WHEN_CLAUSE*);
- corpul *trigger*-ului (*TRIGGER_BODY*);
- antetul (*DESCRIPTION*);
- starea *trigger*-ului (*STATUS*)
 - poate să fie *ENABLED* sau *DISABLED*;
 - numele utilizate pentru a referi parametrii *OLD* și *NEW* (*REFERENCING_NAMES*).
- În operațiile de gestiune a bazei de date este necesară uneori reconstruirea instrucțiunilor *CREATE TRIGGER*, atunci când codul sursă original nu mai este disponibil.
 - Aceasta se poate realiza utilizând informațiile din vizualizarea *USER_TRIGGERS*.

Exemplul 8.11 – vezi curs**8.6. Privilegii sistem**

- Sistemul furnizează privilegii sistem pentru gestiunea *trigger*-ilor:
 - *CREATE TRIGGER* (permite crearea *trigger*-ilor în schema personală);
 - *CREATE ANY TRIGGER* (permite crearea *trigger*-ilor în orice schemă cu excepția celei corespunzătoare utilizatorului *SYS*);
 - *ALTER ANY TRIGGER* (permite activarea, dezactivarea sau compilarea *trigger*-ilor în orice schemă cu excepția utilizatorului *SYS*);
 - *DROP ANY TRIGGER* (permite suprimarea *trigger*-ilor la nivel de bază de date în orice schemă cu excepția celei corespunzătoare utilizatorului *SYS*);
 - *ADMINISTER DATABASE TRIGGER* (permite crearea sau modificarea unui *trigger* sistem referitor la baza de date);
 - *EXECUTE* (permite referirea, în corpul *trigger*-ului, a procedurilor, funcțiilor sau pachetelor din alte scheme).

8.7. Tabele *mutating*

- Un tabel *mutating* este un tabel care este modificat curent de o comandă *LMD* (tabelul este aflat în proces de modificare).
- Un *trigger* la nivel de linie nu poate obține informații (*SELECT*) dintr-un tabel *mutating*, deoarece ar “vedea” date inconsistente (datele din tabel ar fi proces de modificare în timp ce *trigger*-ul ar încerca să le consulte).



- ❖ Tabelele nu sunt considerate *mutating* pentru *trigger*-ii la nivel de comandă.
- ❖ Vizualizările nu sunt considerate *mutating* în *trigger*-ii *INSTEAD OF*.



- ❖ Atunci când este definit un *trigger* trebuie respectată următoarea regulă:
Comenzile *SQL* din corpul *trigger*-ului nu pot consulta sau modifica date dintr-un tabel *mutating*.
- ❖ Chiar tabelul pe care este definit *trigger*-ul este un tabel *mutating*.
- ❖ Există o sigură excepție:
Dacă o comandă *INSERT* afectează numai o înregistrare, *trigger*-ii la nivel de linie pentru înregistrarea respectivă nu tratează tabelul ca fiind *mutating*.
- ❖ Comanda *INSERT INTO tabel SELECT ...* consideră tabelul *mutating* chiar dacă cererea întoarce o singură înregistrare.



- ❖ Fiecare versiune nouă a bazei de date *Oracle* reduce impactul erorilor *mutating*.
- ❖ Dacă un *trigger* determină o astfel de eroare, atunci singura opțiune este ca acesta să fie rescris (o soluție este utilizarea *trigger*-ilor la nivel de comandă).

Exemplul 8.12 – **vezi curs**

Exemplul 8.13 – **vezi curs**

Exemplul 8.14 – **vezi curs**

Exemplul 8.15 – **vezi curs**

Exemplul 8.16 – **vezi curs**

Exemplul 8.17 – **vezi curs**

Implementați cu ajutorul unui *trigger* următoarea restricție: un client poate beneficia într-un an de cel mult 3 perioade cu prețuri preferențiale.

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
4. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)