PL/SQL DBMS-SQL

# Introducere

Pachetul DBMS\_SQL ofera o interfata (API – Application Programming Interface) pentru a ajuta SQL dinamic sa execute comenzi/afirmatii de tip DML si DDL, sa execute blocuri PL/SQL anonime si sa apeleze proceduri si functii PL/SQL stocate.

De exemplu, putem introduce comanda **DROP TABLE** dintr-o procedura stocata deja folosind comanda **PARSE** oferita de pachetul DBMS\_SQL.

Folosind pachetul DBMS\_SQL, putem construi interogari(queries) sau alte comenzi in timpul rularii, in loc sa folosim acele comenzi atunci cand scriem aplicatia.

Pachetul acesta ofera o metoda de a folosi SQL dinamic ce e compatibila cu bazele de date Oracle fara sa modificam aplicatia.

Un lucru de observat este ca acest pachet DBMS\_SQL presupune ca userul current are permisiunile necesare cand ruleaza comenzi din SQL dinamic.

# Cum functioneaza

Pachetul DBMS\_SQL este utilizat pentru a genera interogari SQL in mod dinamic – adica interogari a caror text nu este cunoscut inainte de executia efectiva a codului.

De exemplu, de ce nu avem voie sa executam afirmatii non-query de tip DDL si DML, SQL queries si blocuri anonime direct din PL/SQL?

Problema apare cand executam cod intr-un mediul SQL. Un query SQL tipic porneste totdeauna un cursor, apoi analizeaza SQL-ul pentru erori. Oricare variabile sunt apoi legate de acea sesiune. Inainte sa fie executat codul in PL/SQL, este verificat de erori.

De exemplu, daca incercam sa cream un table fara sa folosim SQL dinamic, codul PL/SQL ar verifica codul mai intai, apoi ar incerca sa valideze faptul ca toate tabelele exista si toate variabilele sunt valide inainte sa execute blocul PL/SQL. Dar, cum tabelul nu exista inca, PL/SQL va sesiza eroare si nu se va executa niciodata pentru ca tabelul nu poate fi legat/asignat.

PL/SQL este designat sa lege toate variabilele mai intai, pentru a permite o executie mai rapida.

Asadar, DBMS\_SQL ocoleste aceasta limitare permitandu-ne efectiv sa cream si sa executam cod SQL in mod dinamic.

# Ce vom discuta

1. Functii si proceduri stocate in acest pachet. Variabile publice din acest pachet
2. Cum folosim DBMS\_SQL
3. Cum folosim comenzi non-query de tip DDL si DML
4. Cum executam ‘queries’ cu DBMS\_SQL
5. **Functii si proceduri stocate in acest pachet. Variabile publice din acest pachet**

Functii si proceduri: BIND\_VARIABLE (c, name, value [, out\_value\_size])

BIND\_VARIABLE\_CHAR (c, name, value [.out\_value\_size])

BIND\_VARIABLE\_RAW (c, name, value [, out\_value\_size])

CLOSE\_CURSOR (c IN OUT)

COLUMN\_VALUE (c, position, value OUT [, column\_error OUT [, actual\_length OUT]])

COLUMN\_VALUE\_CHAR (c, position, value OUT [, column\_error OUT [, actual\_length OUT]])

COLUMN\_VALUE\_RAW (c, position, value OUT [, column\_error OUT [, actual\_length OUT]])

DEFINE\_COLUMN (c, position, column [, column\_size]) + de tip CHAR si RAW

DESCRIBE\_COLUMNS

EXECUTE(c)

EXECUTE\_AND\_FETCH (c [, exact])

FETCH\_ROWS(c)

IS\_OPEN(c)

LAST\_ROW\_COUNT

OPEN\_CURSOR

PARSE (c, statement, language\_flag)

Variabile publice :

* Native: de tip INTEGER, value 1
* V6: de tip INTEGER, value 2
* V7: de tip INTEGER, value 3

Obervatie: Pentru a utiliza DBMS\_SQL, este nevoie de versiunea Oracle7 sau una mai noua, pachetul trebuie instalat si userul trebuie sa aiba toate privilegiile adecvate cum ar fi: DROP TABLE, CREATE TABLE si asa mai departe. Oracle ofera si compatibilitate pentru versiunea 6.

1. **Cum folosim DBMS\_SQL**
2. Deschidem cursorul cu functia OPEN\_CURSOR.
3. Parsam afirmatia SQL cu comanda PARSE.
4. Legam variabilele cu comanda BIND\_VARIABLE.
5. Daca folosim comenzi de tipul SELECT, trebuie sa indicam coloanele pentru output folosind DEFINE\_COLUMNS.
6. Executam statementul SQL cu comanda EXECUTE.
7. Folosim FETCH\_ROWS pentru a aduce/accesa randuri din cursor. Putem folosi comanda EXECUTE\_AND\_FETCH pentru a face 2 lucruri intr-un pas.
8. Daca vrem randuri, trebuie sa folosim comanda COLUMN\_VALUE pentru a lua/recupera data din cursor si sa o stocam intr-o variabila locala.
9. Cand s-a terminat toata procesarea, trebuie sa inchidem cursorul folosind comanda CLOSE\_CURSOR.

Dupa cum am spus, sunt trei tipuri de afirmatii/’statements’ ce pot fi procesate cu DBMS\_SQL : Non-query DDL si DML statements, SQL queries si blocuri anonime PL/SQL.

1. **Cum folosim comenzi non-query de tip DDL si DML**

DDL si DML ne permite sa folosim CREATE, DROP, INSERT, UPDATE, DELETE s.a.m.d in codul nostru PL/SQL. Urmatorii pasi sunt necesari :

1. Deschidem Cursorul
2. Parsam afirmatia/statementul
3. Legam variabilele de intrare (daca este nevoie)
4. Executam statementul
5. Inchidem cursorul
6. Sintaxa pentru functia OPEN\_CURSOR este :

FUNCTION OPEN\_CURSOR RETURN INTEGER;

Aceasta functie returneaza un integer, care este ID-ul cursorului ce va fi retinut pana cand cursorul este inchis la sfarsit.

1. Dupa ce cursorul este deschis, afirmatiile sunt analizate pentru erori de sintaxa. In mod normal, o procedura ar arata toate erorile inainte sa poata fi executata cu success. Dar noi, acum riscam sa avem erori de sintaxa in timpul producerii, rularii, deci trebuie sa avem grija sa nu avem erori de sintaxa, etc.. Sintaxa :

PROCEDURE PARSE(cursor\_id IN INTEGER,

statement\_to\_parse IN VARCHAR2,

version IN VARCHAR2);

cursor\_id este cel assignat de OPEN\_CURSOR de mai sus, statement\_to\_parse este statementul SQL ce vrem sa-l parsam.

1. Asocierea variabilelor: ne permite sa legam o variabila specifica la un placeholder, ce e identificat de caracterul “:”. Dat fiind faptul ca putem lega toate tipurile de variabile PL/SQL, procedura supraincarcata (overloaded) BIND\_VARIABLES se poate descurca cu o multime de tipuri de date.

Sintaxa pentru tipul NUMBER:

PROCEDURE BIND\_VARIABLE(cursor\_id IN INTEGER,

placeholder\_name IN VARCHAR2,

value IN NUMBER);

Sintaxa pentru VARCHAR2 :

PROCEDURE BIND\_VARIABLE(cursor\_id IN INTEGER,

placeholder\_name IN VARCHAR2,

value IN VARCHAR2);

PROCEDURE BIND\_VARIABLE(cursor\_id IN INTEGER,

placeholder\_name IN VARCHAR2,

value IN VARCHAR2,

out \_value\_size IN INTERGER);

Sintaxa pentru celelalte tipuri de variabile este asemanatoare cu exceptia parametrului value IN

1. Executarea afirmatiilor/statements pentru afirmatii non-query de tip DDL si DML

Dupa ce am legat variabilele, daca a fost nevoie, suntem gata sa executam statement-uri de tip DDL si DML cu functia EXECUTE

**Sintaxa** : FUNCTION EXECUTE (cursor\_id IN INTEGER) RETURN INTEGER;

1. Inchidem cursorul

Dupa ce toata procesarea s-a terminat, urmeaza inchiderea cursorului folosind CLOSE\_CURSOR pentru a elibera resursele.

PROCEDURE CLOSE\_CURSOR (cursor\_id IN OUT INTEGER);

Cursor\_id este assignat la inceput de comanda OPEN\_CURSOR. Dupa finalizarea executiei, valoarea cursor\_id devine NULL.

1. **Folosind DDL pentru a crea un tabel**

a.Incepem analiza SQL dynamic prin crearea unui tabel.

DECLARE

v\_CursorID NUMBER;

v\_CreateTableString VARCHAR2(500);

v\_NUMRows INTEGER;

BEGIN

v\_CursorID := DBMS\_SQL.OPEN\_CURSOR;

v\_CreateTableString := 'CREATE TABLE MyTable(

MyRow INTEGER,

MyDesc VARCHAR2(50))';

DBMS\_SQL.PARSE ( v\_CursorID, v\_CreateTableString, DBMS\_SQL.V7);

v\_NumRows := DBMS\_SQL.EXECUTE(v\_CursorID);

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -955 THEN

RAISE;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Table Already Exists!');

END IF;

DBMS\_SQL.CLOSE\_CURSOR(v\_CursorID);

END;

Dupa executarea codului de mai sus va fi afisat urmatorul mesaj :

PL/SQL procedure successfully completed.

Codul de mai sus utilizeaza trei variabile :

* v\_CursorID – este atribuita valoarea ID-ului cursorului returnata de comanda OPEN\_CURSOR.
* v\_CreateTableString – este atribuit codul SQL utilizat la crearea tabelei.
* v\_NUMRows – este atribuit numarul de randuri procesate, care nu are utilitate in comenzile de tip DDL, dar necesar PL/SQL pentru a tine valoarea returnata.

1. **Utilizand DML pentru a adauga inregistrari in tabel**

In tabela creata in scriptul de mai sus acum se pot adauga inregistrari folosind comenzi DML.

DECLARE

v\_CursorID NUMBER;

v\_InsertRecords VARCHAR2(500);

v\_NUMRows INTEGER;

BEGIN

v\_CursorID := DBMS\_SQL.OPEN\_CURSOR;

v\_InsertRecords := 'INSERT INTO MyTable(MyRow,MyDesc)

VALUES (:mynum,:mytext)';

DBMS\_SQL.PARSE(v\_CursorID,v\_InsertRecords,DBMS\_SQL.V7);

/\* definim si introducem prima inregistrare \*/

DBMS\_SQL.BIND\_VARIABLE(v\_CursorID, ':mynum',1);

DBMS\_SQL.BIND\_VARIABLE(v\_CursorID, ':mytext','One');

v\_NumRows := DBMS\_SQL.EXECUTE(v\_CursorID);

DBMS\_OUTPUT.PUT\_LINE('The number of records just processed is: ' || v\_NUMRows);

/\* definim si introducem a doua inregistrare \*/

DBMS\_SQL.BIND\_VARIABLE(v\_CursorID, ':mynum',2);

DBMS\_SQL.BIND\_VARIABLE(v\_CursorID, ':mytext','Two');

v\_NumRows := DBMS\_SQL.EXECUTE(v\_CursorID);

DBMS\_OUTPUT.PUT\_LINE('The number of records just processed is: ' || v\_NUMRows);

/\* definim si introducem a treia inregistrare \*/

DBMS\_SQL.BIND\_VARIABLE(v\_CursorID, ':mynum',3);

DBMS\_SQL.BIND\_VARIABLE(v\_CursorID, ':mytext','Three');

v\_NumRows := DBMS\_SQL.EXECUTE(v\_CursorID);

DBMS\_OUTPUT.PUT\_LINE('The number of records just processed is: ' || v\_NUMRows);

/\* definim si introducem a patra inregistrare \*/

DBMS\_SQL.BIND\_VARIABLE(v\_CursorID, 'mynum',4);

DBMS\_SQL.BIND\_VARIABLE(v\_CursorID, ':mytext','Four');

v\_NumRows := DBMS\_SQL.EXECUTE(v\_CursorID);

DBMS\_OUTPUT.PUT\_LINE('The number of records just processed is: ' || v\_NUMRows);

/\* duplicam a patra inregistrare \*/

v\_NumRows := DBMS\_SQL.EXECUTE(v\_CursorID);

DBMS\_OUTPUT.PUT\_LINE('The number of records just processed is: ' || v\_NUMRows);

EXCEPTION

WHEN OTHERS THEN

RAISE;

DBMS\_SQL.CLOSE\_CURSOR(v\_CursorID);

COMMIT;

END;

OUTPUT : The number of records just processed is: 1 The number of records just processed is: 1 The number of records just processed is: 1 The number of records just processed is: 1 The number of records just processed is: 1

Folosind scriptul de mai sus au fost introduse 5 inregistrari duplicand a patra inregistrare.

1. **Cum folosim pachetul DBMS\_SQL cu interogari**

Aici vom vedea cum putem executa interogari cu pachetl DBMS\_SQL. Metoda de procesare este destul de asemanatoare cu cea de DDL si DML.

**Pasi:**

1. Deschidem cursorul.
2. Parsarea comenzii pentru a-i verifica corectitudinea.
3. Asocierea variabilelor cu anumite valori (daca este nevoie).
4. Definirea variabilelor de iesire.
5. Executarea Comenzii.
6. Utilizarea comenzii FETCH\_ROWS pentru a prelua randurile din cursor.
7. Stocarea rezultatelor de la FETCH\_ROWS in variabile PL/SQL.
8. Inchiderea cursorului.
9. **Folisim pachetul DBMS\_SQL cu interogari .**

DECLARE

v\_cursorID NUMBER;

v\_selectRecords VARCHAR2(500);

v\_NUMRows INTEGER;

v\_MyNum INTEGER;

v\_MyText VARCHAR2(50);

BEGIN

v\_cursorID := DBMS\_SQL.OPEN\_CURSOR;

v\_selectRecords := 'SELECT \* FROM MyTable';

DBMS\_SQL.PARSE(v\_cursorID,v\_selectRecords,DBMS\_SQL.V7);

DBMS\_SQL.DEFINE\_COLUMN(v\_cursorID,1,v\_MyNum);

DBMS\_SQL.DEFINE\_COLUMN(v\_cursorID,2,v\_MyText,50);

v\_NumRows := DBMS\_SQL.EXECUTE(v\_cursorID);

LOOP

IF DBMS\_SQL.FETCH\_ROWS(v\_cursorID) = 0 THEN

EXIT;

END IF;

DBMS\_SQL.COLUMN\_VALUE(v\_cursorID,1,v\_MyNum);

DBMS\_SQL.COLUMN\_VALUE(v\_cursorID,2,v\_MyText);

DBMS\_OUTPUT.PUT\_LINE(v\_MyNum || ' ' || v\_MyText);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

RAISE;

DBMS\_SQL.CLOSE\_CURSOR(v\_CursorID);

END;

Dupa executia blocului, vom primi ca output :

PL/SQL procedure successfully completed.

Acum , daca rulam comanda SELECT \* FROM MyTable vom avea ca output :

**OUTPUT**

1 One

2 Two

3 Three

4 Four

4 Four

**Folosim pachetul DBMS\_SQL cu blocuri anonime de tip PL/SQL**

Pasii sunt asemanatori cu cei de la executarea comenzilor DDL si DML , si folosirea interogarilor.

Pasi: 1. Deschidem cursorul

2. Parsam comanda pentru verificare (exceptia fiind ca putem folosi caracterul “;”)

3. Asociem variabilele unde este cazul

4. Executam comanda , codul sql

5. Este necesara recuperarea valorilor variabilelor din buffer.

6. Inchidem cursorul.

Pentru a recupera variabilele , folosim procedura VARIABLE\_VALUE . Cand functia EXECUTE este executata, ea stocheaza valori in buffer. Comanda VARIABLE\_VALUE va recupera aceste valori si le va introduce in variabile PL/SQL.

Aceasta procedura supraincarcata are o multimer de sintaxe pentru o multimer de tipuri de date.

Sintaxa pentru tipul NUMBER este :

PROCEDURE VARIABLE\_VALUE(cursor\_id IN INTEGER

placeholder\_name IN VARCHAR2,

output\_variable OUT NUMBER);

Etc.

Exemplu practic : cum folosim blocuri anonime PL/SQL cu pachetul DBMS\_SQL.

Scriptul de mai jos va afisa pe ecran valorile pe care le-am introdus in tabel cu valoarea egala cu 2.

DECLARE

v\_CursorID NUMBER;

v\_MatchRecord VARCHAR2(500);

v\_NUMRows INTEGER;

v\_MyNum INTEGER;

v\_MyText VARCHAR2(50);

BEGIN

v\_CursorID := DBMS\_SQL.OPEN\_CURSOR;

v\_MatchRecord := 'BEGIN

SELECT MyRow,MyDesc

INTO :MyRow, :MyText FROM TibiTable

WHERE MyRow = 2;

END;';

DBMS\_SQL.PARSE(v\_CursorID,v\_MatchRecord,DBMS\_SQL.V7);

DBMS\_SQL.BIND\_VARIABLE(v\_CursorID, ':MyRow',v\_MyNum);

DBMS\_SQL.BIND\_VARIABLE(v\_CursorID, ':MyText',v\_MyText,50);

v\_NumRows := DBMS\_SQL.EXECUTE(v\_CursorID);

DBMS\_SQL.VARIABLE\_VALUE(v\_CursorId,':MyRow',v\_MyNum);

DBMS\_SQL.VARIABLE\_VALUE(v\_CursorId,':MyText',v\_MyText);

DBMS\_OUTPUT.PUT\_LINE(v\_MyNum || ' ' || v\_MyText);

EXCEPTION

WHEN OTHERS THEN

RAISE;

DBMS\_SQL.CLOSE\_CURSOR(v\_CursorID);

END;

Ca output, ar trebui sa avem :

**OUTPUT**

2 Two

Tabelul nostru contine :

1 One

2 Two

3 Three

4 Four

4 Four

Daca schimbam la blocul PL/SQL anonim sa afiseze randurile unde valoare este egala cu 4 , am primi o eroare de tipul:

Error report -

ORA-01422: exact fetch returns more than requested number of rows

ORA-06512: at line 28

01422. 00000 - "exact fetch returns more than requested number of rows"

\*Cause: The number specified in exact fetch is less than the rows returned.

\*Action: Rewrite the query or change number of rows requested

Avem 2 randuri cu valoare egala cu 4, ce sunt selectate de comenzile SQL din acel bloc. Iar dat fiind faptul ca nu putem folosi comanda FETCH\_ROWS, ci putem returna doar un singur rand, ne intampina aceasta eroare.

Aceasta este diferenta principala intre interogari si blocuri anonime.

**In concluzie.**

**Folosind acest pachet, putem executa comenzi DDL (create, drop, alter items) si DML(update, delete, insert rows).**

**Putem executa interogari direct din PL/SQL si putem schimba aceste interogari in mod dinamic**

**In plus, acest pachet ne ofera o modalitate de a executa blocuri anonime din PL/SQL .**

**Inca un lucru important, acest pachet ofera si o multime de modalitati pentru error checking si are posibilitatea de a stoca si primi date de tip LONG pana in 2GB spre si din buffer.**

Referinte: <https://profs.info.uaic.ro/~bd/wiki/index.php/PLSQL_9>

<https://profs.info.uaic.ro/~pmihaela/SGBD/lab4/0420-0422.html>

<https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/DBMS_SQL.html#GUID-9723D6EA-3BD2-4B10-BE29-0E1FD64FDA2B>

[DBMS\_SQL - Developer Guide| Alibaba Cloud Documentation Center](https://www.alibabacloud.com/help/doc-detail/116934.htm)