**Chapter-1**

**PL/SQL INTRODUCTION**

* PL/SQL is a procedural language extension for SQL.
* Oracle 6.0 ----------🡪 PL/SQL 1.0
* Oracle 7.0 ----------🡪 PL/SQL 2.0
* Oracle 8.0 ----------🡪 PL/SQL 8.0

------------------------------------------

------------------------------------------

* Oracle 11.2 ----------🡪 PL/SQL 11.2
* Basically PL/SQL is a block structure programming language.
* Whenever we are subheading PL/SQL stricture all SQL structure within PL/SQL engines all engines.

**PL/SQL STRUCTURE**

Oracle Server pl/sql engine

PL/SQL

BLOCKS

Procedural stmts

PROCEDURAL STATEMENT EXECUTION

PL/SQL

BLOCKS

s

q

l

s

t

m

t

s

Sql Engine

**Block Structure:**

Declare (optional)

- Variable declaration, cursor, userdefined exception;

Begin (mandatory)

- DML, TCL

- select…………..into clause

- Conditional, Control statements

Exception (optional)

- Handling runtime error;

End; (mandatory)

**Declare a variable:**

**Syntax:** variable datatype(size);

Sql> declare

A number(5);

B varchar2(15);

**String a value into variable:**

Using assignment operator (:=) we are string value into variable.

**Syntax:** variablename:=value;

Sql> declare

A number(10);

begin

a:=50;

dbms\_output.put\_line(a);

end;

**Display message (or) variable value:**

Dbms\_output.put\_line(‘message’); (or)

Dbms\_output.put\_line(variablename);

**To view pl/sql version:**

Sql> select \* from v$version;

**Program:**

Sql> set serveroutput on;

Sql> begin

dbms\_output.put\_line('welcome');

end;

**/**

**o/p:** welcome

Sql> declare

A number(5);

Begin

A:=50;

Dbms\_output.put\_line(a);

End;

**o/p:** 50

**Program:**

Sql> declare

A number(10):=&a;

Begin

Dbms\_output.put\_line(a);

End;

**o/p:** Enter value for a: 40

40

**1. SELECT……..INTO CLAUSE:**

This clause is used to retrieve data from table and string into pl/sql variables. Select…….into clause always returns **single record** (or) **single value** from a table.

**Syntax:** select col1,col2,………….into variable1,variable2,………….

From tablename where condition;

This clause used in executable section of the pl/sql block.

Sql> declare

v\_sal number(10);

begin

select max(sal) into v\_sal from emp;

dbms\_output.put\_line(v\_sal);

end;

**o/p:** 5000

**Q) Write pl/sql program for user entered empno then display name of the emp and sal from emp table?**

Sql> declare

v\_ename varchar2(15);

v\_sal number(10);

begin

select ename,sal into v\_ename,v\_sal from emp where empno=&empno;

dbms\_output.put\_line(v\_ename||' '||v\_sal);

end;

**o/p:** Enter value for empno: **7902**

FORD 3000

**Q) Write pl/sql program to retrieve total salary from emp table and storing into another table?**

Sql> create table test(totalsalary number(10));

Sql> declare

v\_sal number(10);

begin

select sum(sal) into v\_sal from emp;

insert into test values(v\_sal);

end;

Sql> select \* from test;

**TOTALSALARY**

29025

Sql> update emp set sal=sal+100;

Sql> declare

v\_sal number(10);

begin

select sum(sal) into v\_sal from emp;

insert into test values(v\_sal);

end;

Sql> select \* from test;

**TOTALSALARY**

29025

30425

Sql> update emp set sal=sal+100;

Sql> declare

v\_sal number(10);

begin

delete from test;

select sum(sal) into v\_sal from emp;

insert into test values(v\_sal);

end;

Sql> select \* from test;

**TOTALSALARY**

31825

Sql> declare

A number(10) not null;

begin

A:=50;

dbms\_output.put\_line(A);

end;

**ERROR:** ORA-06550: PLS-00218: a variable declared NOT NULL must have an initialization assignment

Sql> declare

A number(10) not null:=50;

begin

dbms\_output.put\_line(A);

end;

**o/p:** 50

**NOTE:**

When we are using not null (or) constant clause in variable then we must assign the value when we are declare the variable in declare section of the pl/sql block.

Sql> declare

A number not null:=50;

B constant number(10):=40;

begin

dbms\_output.put\_line(A);

dbms\_output.put\_line(B);

end;

**o/p:** 50

40

**NOTE:**

We can also use DEFAULT clause in place of assignment operator when we are assignment the value into variable in declare section of the pl/sql block.

Sql> declare

A number(10) default 50;

begin

dbms\_output.put\_line(A);

end;

**o/p:** 50

**Q) Write pl/sql program to retrieve maximum salary from emp table and storing pl/sql variable and display max(sal)?**

Sql> declare

v\_sal number(10);

begin

select max(sal) into v\_sal from emp;

dbms\_output.put\_line(v\_sal);

end;

**o/p:** 5000

sql> declare

A number(10);

B number(10);

C number(10);

begin

A:=90;

B:=50;

C:=greatest(a,b); (max(a,b) not allowed and not allowed

dbms\_output.put\_line(C); to group functions in pl/sql expressions)

end;

**o/p:** 90

**NOTE:**

In pl/sql expression we are not allowed use group functions, decode conversion function. But we allowed using number functions, character functions and date conversion functions into pl/sql expression.

Sql> declare

A varchar2(10);

B varchar2(10);

begin

A:='NARASIMHA';

B:=upper(A);

dbms\_output.put\_line(B);

end;

**o/p:** NARASIMHA

**2. VARIABLE ATTRIBUTES:**

Variable attribute are use in place of datatype in variable declaration. When we are using this attributes oracle server automatically allocate memory for the variable based on corresponding on column datatype. This type of notations is called as “ANCHOR NOTATION”. Pl/sql having 2 level variable attributes

1. Column Level Attribute

2. Row Level Attribute

**1. Column Level Attribute:**

In this method we are defining attribute for individual columns this attribute column are represented by **%TYPE.**

**Syntax:** variable tablename.colname%type;

**Program:** sql> declare

v\_ename emp.ename%type;

v\_sal emp.sal%type;

v\_hiredate emp.hiredate%type;

begin

select ename,sal,hiredate into v\_ename,v\_sal,v\_hiredate from emp where empno=&empno;

dbms\_output.put\_line(v\_ename||' '||v\_sal||' '||v\_hiredate);

end;

**o/p:** Enter value for empno: **7902**

FORD 3000 03-DEC-81

**2. Row Level Attribute:**

In this method a single variable can represent all difference datatype in a record within in a table it is also same as structure in ‘C’ language. This variable is also called as “record type variable”. Row level attribute are represented by using **%rowtype.**

**Syntax:** variablename tablename%rowtype;

**Program:** sql> declare

A emp% rowtype;

begin

select ename,sal,hiredate into A.ename,A.sal,A.hiredate from emp where empno=&empno;

dbms\_output.put\_line(A.ename||' '||A.sal||' '||A.hiredate);

end;

**o/p:** Enter value for empno: **7902**

FORD 3000 03-DEC-81

**Program:** sql> declare

A emp% rowtype;

begin

select \* into A from emp where empno=&empno;

dbms\_output.put\_line(A.ename||' '||A.sal||' '||A.hiredate);

end;

**o/p:** Enter value for empno: **7902**

FORD 3000 03-DEC-81

A

Empno ename job mgr hiredate sal comm deptno

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7902 | FORD | ANALYST | 7566 | 03-DEC-81 | 3000 |  | 20 |

**PL/SQL Datatypes & variables:**

1. pl/sql support all sql datatype (scalar datatype) + Boolean datatype

2. Ref Object

3. Composite datatype

4. Large Object (lobs)

5. Non pl/sql variables (or) Bind variables.

Pl/sql having two types of blocks

1. Anonymous

2. Named Blocks

2

Declare

Begin

-----------

-----------

End;

**Ex:** **Ex:** procedure, functions,

Packages

**Anonymous Named Blocks**

(No name given to these blocks) (Name given to these block)

(Not allowed to call in client application) (Calling these blocks in client application)

**Conditional Statement:**

1. If

2. If-else

3. Elsif

**1. If:**

**Syntax:** if condition then

Statements;

End if;

**2. If-else:**

**Syntax:** if condition then

Statements;

End if;

**3. Elsif:**

To check more number of conditions then we are using Elsif clause.

**Syntax:** if condition1 then stmts1;

Elsif conditio2 then stmts2;

Elsif condition3 then stmts3;

-----------------------------------

-----------------------------------

Else stmts;

End if;

**Program:** sql> declare

v\_deptno number(10);

begin

select deptno into v\_deptno from dept where deptno=&deptno;

if v\_deptno=10 then

dbms\_output.put\_line('TEN');

elsif v\_deptno=20 then

dbms\_output.put\_line('TWENTY');

elsif v\_deptno=30 then

dbms\_output.put\_line('THIRTY');

else

dbms\_output.put\_line('OTHERS');

end if;

end;

**o/p:** Enter value for deptno: **10**

TEN

Sql> **/**

Enter value for deptno: **40**

OTHERS

Sql> **/**

Enter value for deptno: **90**

**ERROR:** ORA-01403: no data found

**NOTE-1:**

When pl/sql block contains select…..….into clause and also if requested data not available in a table then oracle server returns an **ERROR:** ORA-01403: no data found

**NOTE-2:**

If pl/sql block contain DML stmts and also to this stmts if requested data not available in a table does not returns error to handle this types of error we most implicit cursor attributes.

**Program:** sql> declare

v\_deptno number(10);

begin

select deptno into v\_deptno from emp where deptno=&deptno;

if v\_deptno=10 then

dbms\_output.put\_line('TEN');

elsif v\_deptno=20 then

dbms\_output.put\_line('TWENTY');

elsif v\_deptno=30 then

dbms\_output.put\_line('THIRTY');

else

dbms\_output.put\_line('OTHERS');

end if;

end;

**o/p:** Enter value for deptno: 10

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

**NOTE-3:**

Whenever select………into clause try to returns more number of row then we are try to return multiple values at a time then oracle server returns on **ERROR:** ORA-01422: exact fetch returns more than requested number of rows.

**Control Statement:**

1. Simple Loop

2. While Loop

3. For Loop

**1. Simple Loop:**

It is also called as ‘Infinite Loop’. Here body of the loop execute repeatedly.

**Syntax:** loop stmts;

End loop;

**Program:** sql> begin

Loop

Dbms\_output.put\_line(‘WELCOME’);

End loop

End;

**o/p:** WELCOME

To exit from infinite loop we are using following two methods.

Method-1:

**Syntax:** exit when truecondition;

**Program:** sql> declare

N number(10):=1;

Begin

Loop

Dbms\_output.put\_line(N);

Exit when N>=5;

N:=N+1;

End loop;

End;

**o/p:** 1

2

3

4

5

Method-2: (using if)

**Syntax:** if condition then exit;

Exit;

**Program:** sql> declare

N number(10):=1;

begin

loop

dbms\_output.put\_line(N);

if n>=5 then

exit;

end if;

N:=N+1;

end loop;

end;

**o/p:** 1

2

3

4

5

**2. While Loop:**

Here body of the loop statement is executed until condition is false.

**Syntax:** while condition loop stmts;

End loop;

**Program:** sql> declare

N number(10):=1;

Begin

While n<=5

Loop

Dbms\_output.put\_line(N);

N:=N+1;

End loop;

End;

**o/p:** 1

----

5

**3. For Loop:**

**Syntax:** for indexvariablename in lowerbind…….upperbind

Loop stmts;

End loop;

**Program:** sql> declare

N number(10);

Begin

For N in 1……5

Loop

Dbms\_output.put\_line(N);

End loop;

End;

**o/p:** 1

----

5

Sql> declare

N number(10);

begin

for N in reverse 1..5

loop

dbms\_output.put\_line(N);

end loop;

end;

**o/p:** 5

----

1

**NOTE:**

Whenever we are using for loop index variable internally behavior like a integer variable.

Sql> begin

for N in 1..5

loop

dbms\_output.put\_line(N);

end loop;

end;

**o/p:** 1

-----

5

**3. BIND VARIABLE:**

These variables are session variable this variable are created in host environment. These variables are also called as “Host Variable”. These are non-pl/sql variable. These variables are used in sql, pl/sql and dynamic sql. When a subprogram having out parameter these subprogram are execute throw bind variable in pl/sql.

Step-1: (creating a bind variable)

**Syntax:** variable variablename datatype;

Step-2: (using bind variable)

**Syntax: :**variablename;

Step-3: (display value from bind variable)

**Syntax:** print variablename;

**Ex:** sql> variable G number;

Sql> declare

A number(10):=500;

begin

**:**G:=a/2;

end;

Sql> print G;

**G**

250

Chapter-2

**CURSOR**

To process multiple record

Cursor

Record by record process

Cursor is an private sql memory area which is used to process multiple records and also this is an record by record process all datatype systems having 2 types of static cursor.

1. Implicit Cursor

2. Explicit Cursor

**1. Implicit Cursor:**

For sql statements returns **single record** is called as “Implicit Cursor”. Implicit cursor memory area is also called as “Content Area”. In oracle when a pl/sql blocks contains select……..into clause are DML statements then oracle server automatically creates a memory area. This memory also called as “sql area” (or) “control area” (or) “implicit cursor”. This memory area returns single records. When pl/sql block contain select…………into clause this memory area all show records multiple records when pl/sql bocks contain DML statement. But this multiple records are processed in at a time sql engine that is developer does not control each individual control implicitly.

**Program:** sql> declare

v\_ename varchar2(10);

v\_sal number(10);

begin

select ename,sal into v\_ename,v\_sal from emp where empno=&empno;

dbms\_output.put\_line(v\_ename||' '||v\_sal);

end;

**o/p:** Enter value for empno: **7902**

FORD 3000

**2. Explicit Cursor:**

For sql statement returns **multiple records** is called as “Explicit Cursor” and also this is an record by record process explicit cursor memory area is also called as “Active Set Area”.

**Explicit cursor life cycle:**

1. Declare

2. Open

3. Fetch

4. Close

**1. Declare:**

In declare section of the pl/sql block we declare cursor memory area using following syntax.

**Syntax:** cursor cursorname is select \* from tablename where condition;

**2. Open:**

Whenever we are open in the cursor then only oracle server retrieve data from table into cursor memory area because in all database system when open in the cursor then only cursor select area executed.

**Syntax:** open cursorname;

This statement used in executed section of the pl/sql block.

**NOTE:**

Whenever we are open in the cursor implicit cursor pointer point to the 1st record in the cursor.

**3. Fetch (fetching data from cursor):**

Using fetch statement we are fetching data from cursor into pl/sql variables.

**Syntax:** fetch cursorname into variablename1,variablename2,………….;

**4. Close:**

Whenever we are closing the cursor all the resource allocated from cursor memory area automatically released.

**Syntax:** close cursorname;

Sql> declare

cursor C1 is select ename,sal from emp;

v\_ename varchar2(15);

v\_sal number(10);

begin

open C1;

fetch C1 into v\_ename,v\_sal;

dbms\_output.put\_line(v\_ename||' '||'salary is below 1000');

fetch C1 into v\_ename,v\_sal;

dbms\_output.put\_line('My Second Employee Name Is'||' '||v\_ename);

fetch C1 into v\_ename,v\_sal;

dbms\_output.put\_line(v\_ename||' '||'high salary');

close C1;

end;

**o/p:** SMITH salary is below 1000

My Second Employee Name Is ALLEN

WARD high salary

**Explicit Cursor Attributes:**

1. %notfound

2. %found

3. %isopen

4. %rowcount

When we are using this attribute in pl/sql blocks we must specify cursor name along with attribute name.

**Syntax:** cursorname%attributename;

Except %rowcount all others cursor attribute returns Boolean value either true (or) false %rowcount return number datatype.

**1. %notfound:**

This attribute return Boolean value either true (or) false this attributes returns true the cursor does not data fetching records from cursor (after fetch statement)

**Syntax:** cursorname%notfound;

**Q) Write pl/sql cursor program to display all emp name salary from emp table using %notfound attribute?**

Sql> declare

v\_ename varchar2(10);

v\_sal number(10);

cursor C1 is select ename,sal from emp;

begin

open C1;

loop

fetch C1 into v\_ename,v\_sal;

exit when C1%notfound;

dbms\_output.put\_line(v\_ename||' '||v\_sal);

end loop;

close C1;

end;

**o/p:** SMITH 800

----------- ------

MILLER 1300

**Q) Write pl/sql cursor program to display first 5 highest salary emp from emp table using %rowcount attribute?**

**Ans:** sql> declare

v\_ename varchar2(10);

v\_sal number(10);

cursor C1 is select ename,sal from emp order by sal desc;

begin

open C1;

loop

fetch C1 into v\_ename,v\_sal;

dbms\_output.put\_line(v\_ename||' '||v\_sal);

exit when C1%rowcount>=5;

end loop;

close C1;

end;

**o/p:** KING 5000

FORD 3000

SCOTT 3000

JONES 2975

BLAKE 2850

**Q) Write pl/sql cursor program to display even number of records from emp table using %rowcount attribute?**

Ans: sql> declare

V\_ename varchar2(15);

V\_sal number(10);

Cursor C1 is select ename,sal from emp;

begin

Open C1;

loop

Fetch C1 into v\_ename,v\_sal;

If mod(c1%rowcount,2)=1

Odd number of records

Exit when C1%notfound;

If mod(C1%rowcount,2)=0 then

Dbms\_output.put\_line(v\_ename||' '||v\_sal);

End if;

End loop;

Close C1;

End;

**o/p:** ALLEN 1600

JONES 2975

BLAKE 2850

SCOTT 3000

TURNER 1500

JAMES 950

MILLER 1300

Whenever we are creating cursor automatically **four memory** locations are created these memory locations are behaviors like a variable. These memory locations are identifier throw cursor attribute also these variable are only one value at a time.

C1%notfound C1%found C1%isopen C1%rowcount

|  |
| --- |
| 5 |

|  |
| --- |
| True |

|  |
| --- |
| False |

|  |
| --- |
| True |

Sql> Declare emp

-----------------------------------------

A number(10);

B Boolean; A cursor

-------------------------------------------

50

Begin

A:=50; B fetch

-----------------------------------------

True

B:=true;

**2. %found:**

These attribute returns Boolean value either true (or) false. It returns true if data available within the cursor after fetching the data from the cursor.

**Q) Write pl/sql cursor program to display all emp names and sal from emp table using %found?**

Sql> declare

v\_ename varchar2(10);

v\_sal number(10);

cursor C1 is select ename,sal from emp;

begin

open C1;

fetch C1 into v\_ename,v\_sal;

while C1%found

loop

dbms\_output.put\_line(v\_ename||' '||v\_sal);

fetch C1 into v\_ename,v\_sal;

end loop;

close C1;

end;

**o/p:** SMITH 800

---------- ------

MILLER 1300

**Program:** Sql> declare

cursor C1 is select \* from emp;

I emp% rowtype;

begin

open C1;

loop

fetch C1 into i;

exit when C1%notfound;

if i.sal>3000 then

dbms\_output.put\_line(i.ename||' '||'high salary');

else

dbms\_output.put\_line(i.ename||' '||i.sal);

end if;

end loop;

close C1;

end;

**o/p:** SMITH 800

---------- -----

KING high salary

---------- ------

MILLER 1300

**3. %isopen:**

This attribute always returns either true (or) false it returns true if cursor is already open else it return false.

Sql> declare

cursor C1 is select \* from emp;

i emp%rowtype;

begin

if not C1%isopen then

open C1;

end if;

loop

fetch C1 into i;

exit when C1%notfound;

dbms\_output.put\_line(i.ename||' '||i.sal);

end loop;

close C1;

end;

**o/p:** SMITH 800

---------- ------

MILLER 1300

**4. %rowcount:**

This attribute always returns number datatype that is it counts no.of records related from the cursor that’s why always this attributes returns numbers.

Sql> declare

cursor C1 is select ename,sal from emp;

v\_ename varchar2(15);

v\_sal number(10);

begin

open C1;

fetch C1 into v\_ename,v\_sal;

dbms\_output.put\_line(v\_ename||' '||v\_sal);

fetch C1 into v\_ename,v\_sal;

dbms\_output.put\_line(v\_ename||' '||v\_sal);

dbms\_output.put\_line('Number Of Records Fetch From The Cursor' ||' '||C1%rowcount);

close C1;

end;

**o/p:** SMITH 800

ALLEN 1100

Number Of Records Fetch From The Cursor 2

**NOTE:**

Cursor also used to transfer data from source table into target table.

Sql> declare

cursor C1 is select \* from emp where sal>2000;

i emp%rowtype;

N number;

begin

open C1;

loop

fetch C1 into i;

exit when C1%notfound;

N:=C1%rowcount;

insert into test(sno,ename,salary)values(N,i.ename,i.sal);

end loop;

close C1;

end;

**o/p:** Sql> select \* from test;

**SNO** **ENAME** **SALARY**

1 JONES 2675

2 BLAKE 2550

3 CLARK 2150

4 SCOTT 2900

5 KING 4700

6 FORD 2900

**NOTE:**

Whenever resource table having null value and also if we try to perform summation then database server returns null value to overcome this problem in oracle we are using NVL function.

N:=N+nvl(i.sal,0)

Sql> declare

cursor C1 is select \* from emp;

N number(10):=0;

begin

for i in C1

loop

N:=N+nvl(i.sal,0);

end loop;

dbms\_output.put\_line('Total Salary Is'||' '||N);

end;

**o/p:** Total Salary Is 29025

**Eliminating explicit cursor life cycle (or) cursor for loop:**

Using cursor for loop we are eliminating explicit cursor life cycle. That is explicitly we are not allowed to use declare, open, fetch, close statement. Whenever we are using cursor for loop internally oracle servers only open the cursor and fetch data from the cursor and close the cursor automatically this is also called as “Short Cut Method Of The Cursor”.

**Syntax:** for indexvariablename in cursorname

Loop stmts;

End loop;

Cursor for loop must be use executable section of the pl/sql block.

**NOTE:**

When we are using cursor for loop index variable internally behavior like a record by variable(%rowtype).

**Program:** sql> declare

cursor C1 is select \* from emp;

begin

for i in C1

loop

dbms\_output.put\_line(i.ename||' '||i.sal);

end loop;

end;

**o/p:** SMITH 800

---------- ------

MILLER 1300

**NOTE:**

We are also eliminate total life cycle of the cursor using cursor for loop.

**Syntax:** for indexvariablename in(select statement);

Loop statement;

End loop;

**Q) Write pl/sql cursor program to display all employees names where sal without using cursor name?**

Sql> begin

for i in (select \* from emp)

loop

dbms\_output.put\_line(i.ename||' '||i.sal);

end loop;

end;

Sql> declare

cursor C1 is select \* from emp;

begin

for i in C1

loop

if i.sal>2000 then

dbms\_output.put\_line(i.ename||' '||i.sal);

else

dbms\_output.put\_line(i.ename||' '||'Low Salary');

end if;

end loop;

end;

**o/p:** SMITH Low Salary

-------- -------------

JONES 2975

--------- --------------

KING 5000

TURNER Low Salary

----------- --------------

FORD 3000

**PARAMETERIZED CURSOR:**

In oracle we can also pass parameters to the cursor same like a subprogram in parameters this types of cursors are called as ”Parameterized cursor”. These parameters are used to pass the values into cursor in parameterized cursor. We are defining formal cursor whenever we are declaring the cursor where as we are passing actual parameter and opening the cursor.

**NOTE:**

In oracle when we are passing parameters to the cursor, procedure, functions then we are not allowed use datatype size. Then we are declaring formal parameter.

**Syntax:** cursor cursorname(parametername datatype) is select \* from tablename where columnname=parametername;

**Syntax:** open cursorname(actual parameters);

**Program:** sql> declare

Cursor C1(p\_deptno number) is select \* from emp where deptno=p\_deptno;

I emp%rowtype;

Begin

Open C1(10);

Loop

Fetch C1 into I;

Exit when C1%notfound;

dbms\_output.put\_line(I.ename||' '||I.sal||' '||I.deptno);

end loop;

end;

**o/p:** CLARK 2450 10

KING 5000 10

MILLER 1300 10

**Q) Write pl/sql parameterized cursor program for passing JOB as a parameters display to emp name who are working SALESMAN, MANAGER from emp table and also display output statically?**

**Ex:** employee working as SALESMAN----🡪 Allen, Ward, Martin, Turner

Employee working as MANAGER----🡪 Black, Clark, Jones

Sql> declare

cursor C1(p\_job varchar2) is select \* from emp where job=p\_job;

i emp%rowtype;

begin

open C1('SALESMAN');

dbms\_output.put\_line('Employee Working as Salesman');

loop

fetch C1 into i;

exit when C1%notfound;

dbms\_output.put\_line(i.ename);

end loop;

close C1;

open C1('MANAGER');

dbms\_output.put\_line('Employee Working as Manager');

loop

fetch C1 into i;

exit when C1%notfound;

dbms\_output.put\_line(i.ename);

end loop;

close C1;

end;

**o/p:** Employee Working as Salesman

ALLEN

WARD

MARTIN

TURNER

Employee Working as Manager

JONES

BLAKE

CLARK

**NOTE-1:**

Before we are scoping the cursor we must close the cursor oracle server returns an **ERROR:** ora-06511: Cursor Already Open.

**NOTE-2:**

When we are not opening the cursor but try to perform cursor operation then oracle server returns an **ERROR:** ora-01001: Invalid Cursor.

**NOTE-3:**

In parameterized cursor we can also run default values using default (or) := operator.

**Syntax:** parametername datatype [:=] actual value

**Program:** sql> declare

cursor C1(p\_deptno number default 10)is select \* from emp where deptno=p\_deptno;

i emp% rowtype;

begin

open C1();

loop

fetch C1 into i;

exit when C1%notfound;

dbms\_output.put\_line(i.ename||' '||i.deptno);

end loop;

close C1;

end;

**o/p:** CLARK 10

KING 10

MILLER 10

Parameterized cursor converts into short cut cursor for loop:

Sql> declare

cursor C1(p\_deptno number) is select \* from emp

where deptno=p\_deptno;

begin

for i in c1(20)

loop

dbms\_output.put\_line(i.ename||' '||i.deptno);

end loop;

end;

**o/p:** SMITH 20

JONES 20

SCOTT 20

ADAMS 20

FORD 20

**NOTE:**

In pl/sql blocks we can also defined multiple cursors and also pass the value into another cursor in this type of summation receiving cursor must be a parameterized cursor.

**Q) Write pl/sql program to retrieve all deptno from dept table receive in to static cursor to display emp details based on that deptno from emp table?**

Sql> declare

cursor c1 is select deptno from dept;

cursor c2 (p\_deptno number) is select \* from emp where deptno=p\_deptno;

begin

for i in c1

loop

dbms\_output.put\_line('My Deptno Is'||' '||i.deptno);

for j in c2(i.deptno)

loop

dbms\_output.put\_line(j.ename||' '||j.sal||' '||j.deptno);

end loop;

end loop;

end;

**o/p: My Deptno Is 10**

CLARK 2450 10

---------- ------- ---

**My Deptno Is 20**

SMITH 800 20

--------- ------ ---

**My Deptno Is 30**

BLAKE 2850 30

---------- ------ ---

**My Deptno Is 40**

Parameterized cursor converts into short cut cursor for loop:

Sql> declare

cursor c1(p\_job varchar2)is select \* from emp where job=p\_job;

begin

dbms\_output.put\_line('Employees working as Salesman');

for i in c1('SALESMAN')

loop

dbms\_output.put\_line(i.ename);

end loop;

dbms\_output.put\_line('Employee working as Manager');

for i in c1('MANAGER')

loop

dbms\_output.put\_line(i.ename);

end loop;

end;

**o/p:** Employees working as Salesman

ALLEN

WARD

MARTIN

TURNER

Employee working as Manager

JONES

BLAKE

CLARK

**FUNCTIONS, EXPRESSIONS USED IN CURSORS:**

We can also use functions (or) expressions in cursors select statement. In this case we must use alias name for functions (or) expressions and also declare cursor record type variable in declare section of the pl/sql block (%rowtype).

**Q) Write a pl/sql program to display total salary from emp table using SUM() function?**

Sql> declare

cursor C1 is select sum(sal) A from emp;

i C1%rowtype;

begin

open C1;

fetch C1 into i;

dbms\_output.put\_line('Total Salary Is'||' '||i.A);

close C1;

end;

**o/p:** Total Salary Is 29025

**Q) Write a pl/sql parameterized cursor for loop passing deptno as a parameter cursor can display number of emp’s, total salary, minimum, maximum salary, from emp table based on passed deptno?**

Sql> declare

cursor C1(p\_deptno number) is select count(\*) A, sum(sal) B, min(sal) C, max(sal) D from emp where deptno=p\_deptno;

i C1%rowtype;

begin

open C1(&deptno);

fetch C1 into i;

dbms\_output.put\_line('Total Number Of Employees Are'||' '||i.A);

dbms\_output.put\_line('Total Salary Is'||' '||i.B);

dbms\_output.put\_line('Minimum Salary Is'||' '||i.C);

dbms\_output.put\_line('Maximum Salary Is'||' '||i.C);

close C1;

end;

**o/p:** Enter value for deptno: **10**

Total Number Of Employees Are 3

Total Salary Is 8750

Minimum Salary Is 1300

Maximum Salary Is 1300

**JOINS USED IN CURSORS:**

**Q) Write a pl/sql cursor program to display employees names and their department names from emp, dept tables?**

Sql> declare

cursor C1 is select ename,dname from emp e,dept d where e.deptno=d.deptno;

i C1%rowtype;

begin

open C1;

loop

fetch C1 into i;

exit when C1%notfound;

dbms\_output.put\_line(i.ename||' '||i.dname);

end loop;

close C1;

end;

**o/p:** SMITH RESEARCH

ALLEN SALES

---------- ---------------

MILLER ACCOUNTING

**Update, Delete statement used in cursor (or) where cursor of for update clause used in cursor:**

**Q) Write pl/sql cursor pgm using emp table modify sal of the emp’s based on condition?**

**1) if job=’CLERK’ then increment sal ----------------🡪 200**

**2) if job=’SALESMAN’ then decrement sal ----------🡪 200**

**3) if job=’ANALYST’ then increment sal -------------🡪 200**

Sql> declare

cursor c1 is select \* from emp;

i emp%rowtype;

begin

open c1;

loop

fetch c1 into i;

exit when c1%notfound;

if i.job='CLERK' then

update emp set sal=i.sal+200 where empno=i.empno;

elsif i.job='SALESMAN' then

update emp set sal=i.sal-200 where empno=i.empno;

elsif i.job='ANALYST' then

update emp set sal=i.sal+200 where empno=i.empno;

end if;

end loop;

close c1;

end;

**o/p:** sql> select \* from emp;

**EMPNO** **ENAME** **JOB** **MGR** **HIREDATE** **SAL** **COMM** **DEPTNO**

7369 SMITH CLERK 7902 17-DEC-80 1000 20

------ --------- --------- ------ ------------ ------ --------- ----

7934 MILLER CLERK 7782 23-JAN-82 1500 10

In all database system whenever we are using update, delete statement then database server internally uses default locking mechanism if we want to perform locks on the resource. Before delete then all database system explicitly provides locking mechanism through cursor. Whenever we are using for update in cursor clause and also when we are opening the cursor then only oracle server establishes locks. By default this an exclusive lock.

**Syntax:** cursor cursorname is select \* from tablename where condition for update;

**Where current of:**

Where current of clause used in update, delete statement only this clause internally unique identifies record in each process. This process clause use internally rowid.

**Syntax:** update tablename set colname=newvalue where current of cursorname;

Delete from tablename where current of cursorname;

Whenever we are using where current of clause then we must use for update clause in cursor select statement.

**NOTE:**

Whenever current of clause is used to update, delete latently fetched row from the cursor after processing data we must release the locks using commit.

**Q) Write pl/sql cursor program modify salary of the CLERK from emp table using cursor locking mechanism?**

Sql> declare

cursor c1 is select \* from emp for update;

i emp%rowtype;

begin

open c1;

loop

fetch c1 into i;

exit when c1%notfound;

if i.job='CLERK' then

update emp set sal=i.sal+100 where current of c1;

end if;

end loop;

commit;

close c1;

end;

**o/p:** sql> select \* from emp;

**EMPNO** **ENAME** **JOB** **MGR** **HIREDATE** **SAL** **COMM** **DEPTNO**

7369 SMITH CLERK 7902 17-DEC-80 1000 20

------ --------- --------- ------ ------------ ------ --------- ----

7934 MILLER CLERK 7782 23-JAN-82 1500 10

**Q) Write pl/sql program using cursor locking mechanism all emp’s having KING has their MANAGER get a 5% sal increase from emp table?**

Sql> declare

cursor c1(p\_mgr number)is select \* from emp where mgr=p\_mgr for update;

v\_mgr number(10);

begin

select empno into v\_mgr from emp where ename='KING';

for i in c1(v\_mgr)

loop

update emp set sal=i.sal+100 where current of c1;

end loop;

commit;

end;

sql> select \* from emp;

**1. Implicit cursor attributes:**

Whenever pl/sql block contains select into clause are DML statement then oracle server automatically creates on memory area. This memory area is also called as “SQL AREA” (or) “CONTENT AREA”. This memory area returns single records. When pl/sql block contain memory block this block return multiple block the block process along with this memory area automatically create four memory location behavior variable. This variable are identifies through implicit cursor attributes this area.

1. sql%notfound

2. sql%found

3. sql%isopen

4. sql%rowcount

In implicit cursor always sql%isopen returns false. Where has sql%notfound, sql%found attributes returns Boolean value either true (or) false and also sql%rowcount returns number datatype.

C1%notfound C1%found C1%isopen C1%rowcount

|  |
| --- |
| Number |

|  |
| --- |
| False |

|  |
| --- |
| True/False |

|  |
| --- |
| True/False |

Sql> begin

delete from emp where ename='WELCOME';

if sql%found then

dbms\_output.put\_line('U R Record Deleted');

end if;

if sql%notfound then

dbms\_output.put\_line('U R Records does not exist');

end if;

end;

**o/p:** U R Records does not exist

**Program:** sql> begin

update emp set sal=sal-100 where job='CLERK';

dbms\_output.put\_line('Effected Number of Clerk Is:' ||' '||sql%rowcount);

end;

**o/p:** Effected Number of Clerk Is: 4

Chapter-3

**EXCEPTIONS**

Exception is an error occurred during runtime whenever exception is occur use an appropriate exception name handle in exception section of pl/sql block.

Oracle provided three types of exceptions:

1. Predefined Exception

2. Userdefined Exception

3. Unnamed Exception

**1. Predefined Exception:**

Oracle provided 20 predefined exception names for regularly occurred runtime errors whenever runtime error occurred used on appropriate exception name in exception handle under exception section.

**Syntax:** when predefinedexceptionname1 then

Statement;

When predefinedexceptionname2 then

Statement;

-----------------------------------------------

When others then

Statement;

**Predefined exception names:**

1. no\_data\_found

2. too\_many\_rows

3. zero\_divide

4. invalid\_cursor

5. cursor\_already\_open

6. invalid\_number

7. value\_error

**1. no\_data\_found:**

When pl/sql block contains select……….into clause and also if requested data not available in a table then oracle server returns an **ERROR: ORA-1403: No Data Found.** To handle this error oracle provided **no\_data\_found** exception name.

Sql> declare

v\_ename varchar2(15);

v\_sal number(10);

begin

select ename,sal into v\_ename,v\_sal from emp where empno=&empno;

dbms\_output.put\_line(v\_ename||' '||v\_sal);

exception

when no\_data\_found then

dbms\_output.put\_line('U R Employee Does Not Exists');

end;

**o/p:** Enter value for empno: **7902**

FORD 3000

Sql**> /**

Enter value for empno: **2222**

U R Employee Does Not Exists

**2. too\_many\_rows:**

Whenever select……….into clause try to return more than one value (or) more rows at a time then oracle server returns an **ERROR: ORA-1422: Exact Fetch Returns More Than Requested Number Of Rows.** To handle this error oracle provided **too\_many\_rows** exception name.

Sql> declare

v\_sal number(10);

begin

select sal into v\_sal from emp;

dbms\_output.put\_line(v\_sal);

exception

when too\_many\_rows then

dbms\_output.put\_line('Not Return More Rows');

end;

**o/p:** Not Return More Rows

**3. zero\_divide:**

Whenever we are try to perform division with zero then oracle server return an **ERROR: ORA-1476: Divisor Is Equal To Zero.** To handle this error oracle provided **zero\_divide** exception name.

Sql> begin

dbms\_output.put\_line(5/0);

exception

when zero\_divide then

dbms\_output.put\_line('Not To Perform Division With Zero');

end;

**o/p:** Not To Perform Division With Zero

**4. invalid\_cursor:**

If we not opening the cursor but we try to perform operation using cursor then oracle server return an **ERROR: ORA-1001: Invalid Cursor.** To handle this error oracle provided **invalid\_cursor** exception name.

Sql> declare

cursor C1 is select \* from emp where rownum<=2;

i emp%rowtype;

begin

loop

fetch C1 into i;

exit when C1%notfound;

dbms\_output.put\_line(i.ename||' '||i.sal);

end loop;

close C1;

exception

when invalid\_cursor then

dbms\_output.put\_line('First We Must Open The Cursor');

end;

**o/p:** First We Must Open The Cursor

**5. cursor\_already\_open:**

Before we are reopening the cursor we must close the properly otherwise oracle server return **ERROR: ORA-6511: Cursor Already Open.** The handle this error oracle provided **cursor\_already\_open** exception name.

Sql> declare

cursor C1 is select \* from emp where rownum<=2;

i emp%rowtype;

begin

open C1;

loop

fetch C1 into i;

exit when C1%notfound;

dbms\_output.put\_line(i.ename||' '||i.sal);

end loop;

open C1;

exception

when cursor\_already\_open then

dbms\_output.put\_line('Before Reopen The Cursor We Must Close The Cursor');

end;

**o/p:** SMITH 800

ALLEN 1600

Before Reopen The Cursor We Must Close The Cursor

**NOTE:**

If we are try to convert string type into number type (or) date string type into date type then oracle server return two types errors.

6. Invalid Number

7. Valid Error

**6. invalid\_number:**

When pl/sql blocks contain sql statement and also those statement try to convert string type into number type (or) date string into date type then oracle server returns an **ERROR: ORA-1722: Invalid Number.** To handle this error oracle provided **invalid\_number** exception name.

Sql> select '18-sep-04'+5 from dual;

**ERROR**: ORA-01722: invalid number

Sql> begin

insert into emp(empno,ename,sal)values(1,'narasimha','simha');

exception

when invalid\_number then

dbms\_output.put\_line('Insert Proper Data Only');

end;

**o/p:** Insert Proper Data Only

**7. value\_error:**

When pl/sql block contains procedure statement and also that statement content string type to date type then oracle server returns an **ERROR: ORA-6502: Numeric Converting Error.** To handle this error oracle provided **value\_error** exception name.

Sql> declare

A number(10);

begin

A:='&b'+'&c';

dbms\_output.put\_line(A);

exception

when value\_error then

dbms\_output.put\_line('Enter proper data only');

end;

**o/p:** Enter value for b: **x**

Enter value for c: **d**

Enter proper data only

**NOTE:**

If we are tried to store more data than the datatype size in character datatype then oracle server returns an **ERROR: ORA-06502: PL/SQL: numeric or value error: number precision too large.** To handle this error oracle provided **value\_error** exception name.

Sql> declare

Z number(2);

begin

Z:=9999999;

dbms\_output.put\_line(Z);

end;

**o/p:** ERROR: ORA-6502: PL/SQL: numeric or value error: number precision too large

sql> declare

Z number(2);

begin

Z:=9999999;

dbms\_output.put\_line(Z);

exception

when value\_error then

dbms\_output.put\_line('Invalid String Lengths');

end;

**o/p:** Invalid String Lengths

Sql> declare

Z number(2):='abcd';

begin

dbms\_output.put\_line(Z);

exception

when value\_error then

dbms\_output.put\_line('Invalid String Lengths');

end;

**o/p:** ERROR: ORA-06502: PL/SQL: numeric or value

error: character to number conversion error

**Exception Propagation:**

In pl/sql exception also raised, declare, executable section, exception section when exception raised executable section those exception are handle either in inner blocks or in declare section where as exception are raised in declare section (or) in exception section those exception must be handle within outer block this is called “Exception Propagation”.

Sql> begin

declare

Z varchar2(3):='abcd';

begin

dbms\_output.put\_line(Z);

end;

exception

when value\_error then

dbms\_output.put\_line('invalid string length handle outer block only');

end;

**o/p:** invalid string length handle through outer block only

Sql> begin

declare

Z varchar2(3):='abcd';

begin

dbms\_output.put\_line(Z);

exception

Inner when value\_error then

Block dbms\_output.put\_line('invalid inner block string length');

end;

exception

Outer when value\_error then

Block dbms\_output.put\_line('invalid string length handle outer block only');

end;

**o/p:** invalid string length handle outer block only

**dup\_val\_on\_index:**

In oracle when we are trying to insert duplicate data into primary key (or) unique key constrains then oracle server returns an **ERROR: ORA-00001: unique constraint (SCOTT.PK\_EMP) violated.** To handle this error we are using **dup\_val\_on\_index** exception name.

Sql> begin

insert into emp(empno)values(7369);

exception

when dup\_val\_on\_index then

dbms\_output.put\_line('Not insert into duplicate data');

end;

**o/p:** Not insert into duplicate data

**2. Userdefined Exception:**

We can also create our own exception name and also raise explicitly this type of exception are also called as “Userdefined Exception”.

Step-1: Declare

Step-2: Raise

Step-3: Handle Exception

**Declare:**

In declare section of the pl/sql block we are creating our own exception name using exception predefined type.

**Syntax:** userdefinedname exception;

Sql> declare

A exception;

**Raise:**

Using raise statement we can also raise userdefined exception explicitly either in executable section (or) in exception section of the pl/sql block.

**Syntax:** raise userdefinedexceptionname;

**Handling:**

We are also handle userdefined exception same like a predefined using exception handle in exception section.

**Syntax:** when userdefinedexceptionname1 then Statement;

When userdefinedexceptionname2 then Statement;

-------------------------------------------------

When others then Statement;

**1. Testing exception propagation using userdefined exception:**

Exception raised in executable section:

In pl/sql when exception raised executable section those exception handle either in inner block (or) in outer block.

Method-1: (Handle using outer block)

Sql> declare

A exception;

begin

begin

raise A;

end;

exception

when A then

dbms\_output.put\_line('Handle using outer block');

end;

**o/p:** Handle using outer block

method-2: (Handle using inner block)

Sql> declare

A exception;

begin

raise A;

exception

when A then

dbms\_output.put\_line('Handle using inner block');

end;

**o/p:** Handle using inner block

**2. Exception raised in exception section:**

In pl/sql when exception are raised on exception section those exceptions are must be handle using outer block only.

Sql> declare

y1 exception;

y2 exception;

begin

begin

raise y1;

exception

when y1 then

dbms\_output.put\_line('y1 is handled');

raise y2;

end;

exception

when y2 then

dbms\_output.put\_line('y2 is handled');

end;

**o/p:** y1 is handled

y2 is handled

**Q) Write pl/sql program raise userdefined exception on Thursday?**

Sql> declare

Z exception;

begin

if to\_char(sysdate,'DY')='THU' then

raise Z;

end if;

exception

when Z then

dbms\_output.put\_line('My exception is raised on Thursday');

end;

**o/p:** My exception is raised on Thursday

**Program:** sql> declare

A exception;

v\_sal number(10);

begin

select sal into v\_sal from emp where ename='KING';

if v\_sal>2000 then

raise A;

else

update emp set sal=sal+100 where ename='KING';

end if;

exception

when A then

dbms\_output.put\_line('Salary is already high');

end;

**o/p:** Salary is already high

P**rogram:** sql> declare

cursor C1 is select \* from emp where deptno=&deptno;

i emp%rowtype;

A exception;

begin

open C1;

fetch C1 into i;

if C1%rowcount=0 then

raise A;

else

while C1%found

loop

dbms\_output.put\_line(i.ename||' '||i.sal||' '||i.deptno);

fetch C1 into i;

end loop;

end if;

close C1;

exception

when A then

dbms\_output.put\_line('U R Deptno does not Exists');

end;

**o/p:** Enter value for deptno: **10**

CLARK 2450 10

KING 5000 10

MILLER 1300 10

SQL> **/**

Enter value for deptno: **50**

U R Deptno does not Exists

**NOTE:**

In oracle we can also raise predefined exception names using raise statement.

Sql> declare

cursor C1 is select \* from emp where job='ABC';

i emp%rowtype;

begin

open C1;

fetch C1 into i;

if C1%rowcount=0 then

raise no\_data\_found;

end if;

dbms\_output.put\_line(i.ename||' '||i.sal);

close C1;

exception

when no\_data\_found then

dbms\_output.put\_line('U R Requested Job Not Available in My Table');

end;

**o/p:** U R Requested Job Not Available in My Table

**3. Unnamed Exception:**

In oracle if we want to handle other then oracle 20 predefined exception name errors then we must use unnamed method this called as “Unnamed Exception”. In this method we are creating our own exception name and associate that exception name with appropriate error number using **exception\_init().** This function accepts two parameters.

**Syntax:** pragma exception\_init(userdefinedexceptionname, error number);

These pragma is used in declare section of the pl/sql block.

Here pragma is a complier directive, whenever we are using this pragma oracle server internally associate error numbers with exception name at compile time.

Sql> declare

A exception;

pragma exception\_init(A,-1400);

begin

insert into emp(empno,ename)values(null,'NARASIMHA');

exception

when A then

dbms\_output.put\_line('Not TO insert null Values');

end;

**o/p:** Not TO insert null Values

Handle on error -2292:

Sql> declare

A exception;

pragma exception\_init(A,-2292);

begin

delete from dept where deptno=10;

exception

when A then

dbms\_output.put\_line('Not To Delete Master Table Record');

end;

**o/p:** Not To Delete Master Table Record

**Q) Write pl/sql program to handle -2291 error number using exception\_init pragma based on emp,dept tables?**

Sql> declare

A exception;

pragma exception\_init(A,-2291);

begin

insert into emp(empno,ename,deptno)values(1,'ABC',90);

exception

when A then

dbms\_output.put\_line('Not To insert Others primary value');

end;

**o/p:** Not To insert Others primary value

**Error Trapping Function:**

In oracle for catching error numbers,error number with error messages,then we are using two functions,these functions are also called as “Error Trapping Functions”. These are

1. sqlcode

2. sqlerrm

These 2 error trapping functions are used in when others then clause in exception handler and also used in our own exception handler (userdefined). In oracle if we want to catch error number at runtime. Then we are using sqlcode function.

Sql> declare

v\_sal number(10);

begin

select sal into v\_sal from emp where deptno=&deptno;

dbms\_output.put\_line(v\_sal);

exception

when others then

dbms\_output.put\_line(sqlcode);

end;

**o/p:** Enter value for deptno**: 10**

**-1422**

SQL**> /**

Enter value for deptno**: 'H'**

**-1722**

SQL**> /**

Enter value for deptno**: 50**

**100**

Sqlcode always returns numbers where as sqlerrm returns error number with error message. Generally if we want to return errors at runtime then we are using sqlcode.

**Sqlcode** **Meaning**

1. No Error

- ve Oracle Error

100 No Data Found

1 Userdefined Exception

Sql> declare

A exception;

begin

raise A;

exception

when A then

dbms\_output.put\_line(sqlcode);

dbms\_output.put\_line(sqlerrm);

end;

**o/p:** 1

User-Defined Exception

**NOTE:**

We can also pass sqlcode value into sqlerrm function.

Sql> begin

dbms\_output.put\_line(sqlcode);

dbms\_output.put\_line(sqlerrm(sqlcode));

dbms\_output.put\_line(sqlerrm(100));

dbms\_output.put\_line(sqlerrm(1));

dbms\_output.put\_line(sqlerrm(-1722));

dbms\_output.put\_line(sqlerrm(-1400));

dbms\_output.put\_line(sqlerrm(-1403));

end;

**o/p:** 0

ORA-0000: normal, successful completion

ORA-01403: no data found

User-Defined Exception

ORA-01722: invalid number

ORA-01400: cannot insert NULL into ()

ORA-01403: no data found

**NOTE:**

Using sqlcode function we can also handle unnamed exception.

Sql> begin

delete from dept where deptno=10;

exception

when others then

if sqlcode=-2292 then

dbms\_output.put\_line('Not To Delete Master Records');

end if;

end;

**o/p:** Not To Delete Master Records

**NOTE:**

Generally we are not allowed sqlcode ,sqlerrm in DML statement. If we want to store sqlcode,sqlerrm return values insert into error lock table then we must declare variable and assigns this function values into variable then only we are using this variables in insert statement.

**Q) Write pl/sql prg to store error number, error number with message of in pl/sql block?**

Sql> create table test(errno number(10),errmsg varchar2(200));

Sql> declare

v\_sal number(10);

v\_errno number(10);

v\_errmsg varchar2(200);

begin

select sal into v\_sal from emp;

exception

when others then

v\_errno:=sqlcode;

v\_errmsg:=sqlerrm;

insert into test values(v\_errno,v\_errmsg);

end;

**o/p:** select \* from test;

**ERRNO** **ERRMSG**

-1422 ORA-01422: exact fetch returns more than requested number of rows

**RAISE\_APPLICATION\_ERROR:**

If we want to display userdefined exception message in more descriptive form then we are using this function. If we want to display userdefined exception message as same as oracle error display format then only we are using **raise\_application\_error()** procedure these applications available in **dbms\_standard**. This procedure used in either in executable section (or) in exception section. This procedure accepts 2 parameters sqlcode, sqlerrm.

**Syntax:** raise\_application\_error(error number, error message);

Sql> declare

A exception;

begin

if to\_char(sysdate,'DY')='SAT' then

raise A;

end if;

exception

when A then

raise\_application\_error(-20435,'Every Saturday My Exception Is Raised');

end;

**o/p:** ERROR:ORA-20435: Every Saturday My Exception Is Raised

**NOTE:** Generally this procedure used in triggers because if the condition is true is raise a message and stops invalid data enter into table based on condition.

Chapter-4

**SUBPROGRAMS**

Subprograms are named pl/sql block which is used to solve particular task. All database systems having 2 types of subprograms.

1. Procedures (may (or) may not return value)

2. Functions (must return a value)

**1. PROCEDURE:** (One time compilation)

Procedure is a named pl/sql block which is used to solve particular task and also procedure may (or) may not returns values. Whenever we are using create or replace keyword infornt of the procedure those procedure are automatically permanently stored in a database. These procedures are also called as **“Stored Procedure”**. Stored procedure is improves the performance and internally one time compilation. In all database systems procedure unites more performance of the application.

Using editor write a procedure to solve particular task

Show error

Execution

Compilation

Using a sqlplus tool to load procedure into database

Sql> desc dbms\_standard;

Every procedure having two parts

1. Procedure Specification

2. Procedure Body

In procedure specifications we are number of the procedure and type of the parameter where has in procedure body we are solving actual task.

**Syntax:** create or replace procedurename(formal parameter)

procedure Is/as

specification ----🡪 Variable declarations,cursors,userdefined exception;

Begin

-----------------------

procedure [exception]

body -----------------------

End [procedurename];

**Syntax:** parametername [mode] datatype

Mode-----------🡪 in, out, in out

To view errors:

Sql> show error;

Executing a procedure:

Method-1: sql> exec procedurename(actual parameter)

Method-2: (using anonyms block)

Sql> begin

Procedurename(actual parameter);

End;

Method-3: sql> call procedurename(actual parameter);

**Q) Write pl/sql stored procedure for passing empno is an parameter then display name of the emp and sal from emp table?**

Sql> create or replace procedure p1(p\_empno number) is

v\_ename varchar2(10);

v\_sal number(10);

begin

select ename,sal into v\_ename,v\_sal from emp where empno=p\_empno;

dbms\_output.put\_line(v\_ename||' '||v\_sal);

end;

Execution:

Method-1: sql> exec p1(7902);

FORD 3000

Method-2: (using anonyms block)

Sql> begin

P1(7839);

End;

**o/p:**  KING 5000

Method-3: sql> call p1(7566);

JONES 2975

In oracle all procedure information stored undered user\_procedures, user\_source data dictionary. If we want to view procedure code then we are using user\_source data dictionary.

Sql> desc user\_procedures;

Sql> desc user\_source;

Sql> select text from user\_source where name='P1';

**Q) Write pl/sql stored procedure for passing deptno is a parameter display emp details from table based on passed deptno?**

Sql> create or replace procedure p2(p\_deptno number) is

i emp%rowtype;

cursor C1 is select \* from emp where deptno=p\_deptno;

begin

open C1;

loop

fetch C1 into i;

exit when c1%notfound;

dbms\_output.put\_line(i.ename||' '||i.sal||' '||i.deptno);

end loop;

end;

**o/p:** SQL> exec p2(10);

CLARK 2450 10

KING 5000 10

MILLER 1300 10

**Q) Write pl/sql stored procedure for passing empno is an parameter delete a record from emp table based on passed empno?**

Sql> create or replace procedure p3(p\_empno number) is

begin

delete from emp where empno=p\_empno;

if sql%found then

dbms\_output.put\_line('U R Record is Deleted');

end if;

if sql%notfound then

dbms\_output.put\_line('U R Record is Not Exits');

end if;

end;

**o/p:**  SQL> exec p3(2222);

U R Record is Not Exits

Sql> create or replace procedure p1 is

begin

dbms\_output.put\_line('First Procedure');

end;

**o/p:** SQL> exec p1;

First Procedure

**Parameter (or) procedure parameters:**

In all database systems parameter are used to passing values into procedure and returns value from the procedure. In---🡪 input

Create procedure

Begin

---------

end;

Jdbc

-------

-------

7902

-------

All procedure having two type of parameter

1. Formal Parameter

2. Actual Parameter

**1. Formal Parameter:** out mode

Formal parameter is defined procedure specification only. Formal parameter specific name of the parameter, mode of the parameter,type of the parameter.

**NOTE;**

In oracle when we are defining formal parameter in procedure,cursor,function. We are not allowed to use datatype size in formal parameter declaration.

**Syntax: (**formal parameter syntax)

Parametername [mode] datatype

**MODE:**

Mode specifies purpose of the parameter. Oracle procedure parameters having three types mode this are

1. in mode

2. out mode

3. in out mode

**1. in mode:**

This mode is used to passing the values into procedure body. In oracle by default parameter mode is IN MODE. In mode behaviors like a **constant** in procedure body that is it’s behind like read only value in procedure body.

**Syntax:** parametername in datatype

**NOTE:**

Through the IN mode we can also pass default values in the IN mode using default (or) **:=** operator.

**Syntax:** parametername in datatype [default (or) := ] actual value

**Q) Write pl/sql stored procedure to insert into a record dept table using IN parameter?**

Sql> create or replace procedure p4(p\_deptno in number,p\_dname in varchar2,p\_loc in varchar2) is

begin

insert into dept values(p\_deptno,p\_dname,p\_loc);

dbms\_output.put\_line('U R Record Inserted Through Procedure');

end;

**o/p:** Sql> exec p4(1,'A','B');

U R Record Inserted Through Procedure

Sql> select \* from dept;

**2. Out mode:**

Out mode is used to return value from the procedure body. Out mode behaviors like a uninitialized **variable** in procedure body here explicitly we must specify OUT keyword.

**Syntax:** parametername out datatype

Sql> create or replace procedure p1(a in number,b out number) is

begin

b:=a\*a;

end;

In a oracle subprograms having OUT (or) IN OUT parameter those subprograms are executed using following 2 methods.

Execution:

Method-1: (using bind variable)

Sql> variable N number

Sql> exec p1(2,:N);

Sql> print n; Sql> print :n;

**N N**

4 4

Method-2: (using anonymous block)

Sql> declare

A number(10);

begin

p1(8,A);

dbms\_output.put\_line(A);

end;

**o/p:** 64

**Q) Write pl/sql stored procedure for passing ename as IN parameter then return salary of the emp using OUT parameter from emp table?**

Sql> procedure p5(p\_ename in varchar2,p\_sal out number) is

begin

select sal into p\_sal from emp where ename=p\_ename;

end;

Execution:

Method-1: (using bind variable)

Sql> variable A number;

Sql> exec p5(‘KING’,:A);

Sql> print A;

**A**

500

Sql> declare

A number(10);

begin

p5('WARD',A);

dbms\_output.put\_line(A);

end;

**o/p:** 1250

**Q) Write pl/sql stored procedure for passing deptno as in parameter return dname,loc suing OUT parameter from dept table?**

Sql> create or replace procedure p6(p\_deptno in number,p\_dname out varchar2,p\_loc varchar2) is

begin

select dname,loc into p\_dname,p\_loc from dept where deptno=p\_deptno;

end;

Execution:

Method-1: sql> variable A varchar2(10);

Sql> variable B varchar2(10);

Sql> exec p6(10,:A,:B);

Sql> print A B;

**A** **B**

ACCOUNTING NEW YORK

**NOTE:**

In all database systems we can also convert group by clause report into procedure IN,OUT parameter.

**Q) Write pl/sql stored procedure current following group by resulting in to IN,OUT parameter in procedure?**

select deptno,count(\*) from emp group by deptno;

**DEPTNO** **COUNT(\*)**

30 6

20 5

10 3

Sql> create or replace procedure p7(p\_deptno in number,p\_count out number) is

begin

select count(\*) into p\_count from emp where deptno=p\_deptno;

end;

Sql> variable A number;

Sql> exec p7(10,:A);

Sql> print A;

**A**

3

**IN Parameter execution method:**

Oracle support three types of execution methods when we are using IN parameter.

1. Positional Notation (ex: sql. Exec p1(1,’a’,’b’))

2. Named Notation (=>) (ex: sql> exec p1(p\_dname=>’a’,p\_loc=>’b’,p\_deptno=>2))

3. Mixed Notation (ex: sql> exec(3,p\_dname=>’a’,p\_loc=>’b’)

**Mixed Notation:**

It is combination of positional, named notation note after positional their can be all named notations but after named their can not be positional.

**NOTE:**

When we are defining default values we are using right to left associability.

Sql> create or replace procedure p7(p\_deptno in number,p\_dname in varchar2,p\_loc in varchar2 default 'HYD') is

Begin

Insert into dept values(p\_deptno,p\_dname,p\_loc);

Dbms\_output.put\_line('U R Record Insert Through Procedure');

end;

**o/p:** Sql> exec p7(5,'GNSR');

U R Record Insert Through Procedure

**Pass by value, pass by Reference:**

Whenever we are using modular programming all language supports 2 types passing particular mechanism this are

1. Pass by Value

2. Pass by Reference

This 2 parameter mechanism specify after we are modify forms formal parameter then actual parameter Effected (or) Not. In pass by value method actual value **does not change** because **copy of the value** is passed into calling program. If we want to affect actual parameter based on formal parameter modification then we are using pass by reference.

Oracle server also support this 2 passing parameters mechanism internally when we are using procedure IN,OUT parameter by default all in parameter internally uses pass by reference where as by default all **OUT parameter** internally uses pass by value method.

When we are returning large amount of data using OUT parameters again copy of the value is created and those value only return into application because OUT parameter internally uses pass by value based on passed value mechanism copy of values are returned. This prove automatically degrades performance of the application. To overcome this problem oracle introduced **nocopy hint** in the OUT parameter.

**Syntax:** parametername out nocopy datatype

Sql> create or replace procedure p1(p\_deptno in number,p\_count out nocopy number) is

Begin

Select count(\*) into p\_count from emp where deptno=p\_deptno;

End;

**o/p:** Sql> variable A number;

Sql> exec p1(10,:A);

Sql> print A;

**A**

3

**3. in out mode:**

This mode is used to pass the values into procedure and return values from the procedure. This mode behaviours like a constant, initialized variable in procedure body. Here also explicitly we must specify IN OUT keyword.

**Syntax:** parametername IN OUT datatype

Sql> create or replace procedure p7(A in out number) is

begin

A:=a\*a;

end;

Sql> variable B number;

Sql> exec :B:=7;

Sql> exec p7(:B);

Sql> print B;

**B**

49

Sql> declare

A number(10):=&A;

begin

p7(A);

dbms\_output.put\_line(A);

end;

**o/p:** Enter value for a: **8**

64

**Q) Write pl/sql stored procedure for passing empno as parameter return sal of the emp using IN OUT parameter?**

Sql> create or replace procedure p8(p\_A in out number) is

begin

select sal into p\_A from emp where empno=p\_A;

end;

**o/p:** Sql> variable A number;

Sql> exec :A:=7839;

Sql> exec p8(:A);

Sql> print A;

**A**

5000

**Autonomous Transaction:**

Autonomous transactions are independent transaction used in procedure, trigger and anonymous block whenever we are calling “Autonomous Procedure”. In another pl/sql block and also blocks having rollback (or) commit those transaction commands are not affected in autonomous procedure. Generally autonomous transactions are used in child procedure.

In oracle if we want make a procedure autonomous then column are using autonomous\_transaction pragma, commit in declare section of the procedure and also we must use commit in procedure body.

**Syntax:** pragma autonomous\_transaction;

Autonomous transaction used in procedure:

**Syntax:** create or replace procedure procedurename(formal parameters) is/as

Pragma autonomous\_transaction;

begin

---------------

Commit;

[Exception]

---------------

End [procedurename];

Sql> create table test(name varchar2(10));

Sql> create or replace procedure p8 is pragma autonomous\_transaction;

begin

insert into test values('INDIA');

commit;

end;

Sql> begin

insert into test values('HYD');

insert into test values('MUMBAI');

p8;

rollback;

end;

Sql> select \* from test;

**NAME**

INDIA

Sql> delete from test;

Sql> select \* from test;

No Row Selected

Without using autonomous transactions in procedure:

Sql> create or replace procedure p9 is

begin

insert into test values('INDIA');

commit;

end;

Sql> begin

insert into test values('HYD');

insert into test values('MUMBAI');

p9;

rollback;

end;

Sql> select \* from test;

**NAME**

HYD

MUMBAI

INDIA

In oracle when a procedure having committed and also when we are calling this procedure in pl/sql block then this procedure commit not only save procedure transaction but also save all above procedure transaction to overcome this problem oracle 8.1.6 introduced autonomous transactions.

Sql> create table test values(sno number(10));

**Main transaction** **session-2**

Sql> insert into test values(1); sql> select \* from test;

Sql> insert into test values(2); no row selected

Sql> select \* from test;

**sno**

1

2

Child transaction:

Sql> declare

Pragma autonomous\_transaction;

Begin

For I in 3..7

Loop

Insert into test values(i);

End loop;

Commit;

End;

Sql> select \* from test;

**Sno**

1

2

3

4

5

6

7

Sql> rollback;

Sql> select \* from test; sql> select \* from test;

**Sno** **sno**

3 3

4 4

5 5

6 6

7 7

**Handled, Unhandled exception in procedure:**

Whenever we are calling inner procedure in to outer procedure then we must handle inner procedure exceptions within procedure exception. Otherwise oracle server executed outer procedure default exception handle.

Inner Procedure:

Sql> create or replace procedure p1(A in number,B in number) is

Begin

dbms\_output.put\_line(a/b);

Exception

When zero\_divide then

Dbms\_output.put\_line('Y can not be zero');

end;

Outer Procedure:

Sql> create or replace procedure p10 is

begin

p1(7,0);

exception

when others then

dbms\_output.put\_line('ANY ERROR');

end;

**o/p:** Sql> exec p10;

Y can not be zero

**Authid current\_user:**

When a procedure having this value another user can not execute that procedure if we are given a privileges also and whenever developer reading data from resource and performing DML operation then only they are using this clause data security point of view. This clause used in procedure specifications.

**Syntax:** create or replace procedure procedurename(formal parameter)

Authid current\_user is/as

--------------------------------

Begin

---------------------------------

[Exception]

----------------------------------

End [procedure];

Given procedure privileges to another user:

**Syntax:** grant execute on procedurename to username1,username2,……;

Sql> create or replace procedure p11(p\_empno number)

authid current\_user is

v\_ename varchar2(10);

v\_sal number(10);

begin

select ename,sal into v\_ename,v\_sal from emp where empno=p\_empno;

dbms\_output.put\_line(v\_ename||' '||v\_sal);

end;

Sql> exec p11(7902);

FORD 3000

Sql> grant execute on p11 to narasimha;

Sql> grant all on emp to narasimha;

Sql> conn narasimha/narasimha;

Sql> exec scott.p11(7902);

Error: Table or View does not exists

We can also drop procedure using drop procedure procedurename.

Sql> drop procedure procedurename;

**2. FUNCTION:**

Function is named pl/sql block which is used to solve particular task and also function must return value. Functions also having 2 pats

1. Function Specification

2. Function Body

In function specification we are specification using name of the function and type of the particular where as in function body we are solving actual task.

**Syntax:** create or replace function functionname(formal parameter)

Return datatype is/as

-----🡪 variable declaration, cursor declaration, user defined exception;

Begin

----------------------

Return expression;

[exception]

-----------------------

End [functionname];

Formal parameter:

**Syntax:** parametername [mode] datatype datatype -----🡪 in ,out, in out

Executing a Function:

Method-1: (using select statement)

When a function having all in parameter (or) function does not have parameters then those type of function are executing through SELECT statement.

**Syntax:** select functionname(actual parameter) from dual;

Method-2: (using Anonymous block)

**Syntax:** begin

Variablename:=Functionname(actual parameter);

End;

**Ex:** sql> create or replace function f1(A varchar2)

return varchar2 is

begin

return a;

end;

**Execution:**

Method-1: (using select statement)

Sql> select f1(‘welcome’) from dual;

Welcome

Method-2: (using anonymous block)

Sql> declare

A varchar2(10);

begin

A:=f1('welcome');

dbms\_output.put\_line(A);

end;

**o/p:** welcome

**Q) Write pl/sql stored function for passing number as a parameter then return either even (or) odd based on parameter?**

Sql> create or replace function f2(A number)

return varchar2 is

begin

if mod(A,2)=0 then

return ('EVEN NUMBER');

else

return ('ODD NUMBER');

end if;

end;

**Execution:**

Method-1: (using select statement)

Sql> select f2(5) from dual;

**F2 (5)**

ODD NUMBER

Method-2: (using anonymous block)

Sql> declare

A varchar2(15);

begin

A:=f2(4);

dbms\_output.put\_line(A);

end;

**o/p: EVEN NUMBER**

Method-3: Sql> variable A varchar2(15);

Sql> begin

:A:=f2(7);

end;

Sql> print A;

**A**

ODD NUMBER

Method-4: Sql> exec dbms\_output.put\_line(f2(8));

EVEN NUMBER

Method-5: Sql> begin

dbms\_output.put\_line(f2(9));

end;

**o/p:** ODD NUMBER

**NOTE:**

We can also use userdefined function in insert statement. In this case function return value match with column datatype.

Sql> create table test(msg varchar2(15));

Sql> insert into test values(f2(3));

Sql> select \* from test;

**MSG**

ODD NUMBER

**DML Statement used in Function:**

**Q) Write pl/sql stored function for passing empno is a parameter deletes that emp record in emp table and also return no.of deleted rows numbers?**

Sql> create or replace function f3(p\_empno number)

return number is

v\_count number(10);

begin

delete from emp where empno=p\_empno;

v\_count:=sql%rowcount;

return v\_count;

end;

**Execution:**

Method-1: sql> select f3(1) from dual;

ERROR: ORA-14551: cannot perform a DML operation inside a query

Method-2: sql> declare

A number(10);

begin

A:=f3(2);

dbms\_output.put\_line(A);

end;

**o/p;** 0

In oracle when a function having DML statement those functions are not allowed to execute using SELECT statement but we are allowed to execute using anonymous block. If we want execute SELECT statement then we must use autonomous transactions in functions i.e here we must use autonomous\_transaction pragma in declare section of the function and also use commit in function body.

**Syntax:** create or replace function functionname(formal parameter)

Return datatype is/as

Pragma autonomous\_transaction;

Begin

-----------------------

Commit;

Return expression;

End [functionname];

Solution:

Sql> create or replace function f4(p\_empno number)

return number is

Pragma autonomous\_transaction;

v\_count number(10);

begin

delete from emp where empno=p\_empno;

v\_count:=sql%rowcount;

commit;

return v\_count;

end;

**o/p:** sql> select f4(1) from dual;

**F4 (1)**

0

**NOTE:**

In oracle we can also use predefined function into userdefined function and also call these user defined function in same (or) in different table.

**Q) Write pl/sql stored function to display a report that is one max(sal) for all employee and no group by function used?**

Sql> create or replace function f5 Return number is

v\_sal number(10);

begin

select max(sal) into v\_sal from emp;

return v\_sal;

end;

Execution: Sql> select ename,sal,f5 from emp;

**ENAME** **SAL** **F5**

SMITH 800 5000

---------- ------- ------

WARD 1250 5000

Sql> select f5 from dual;

**F5**

5000

**Q) Write pl/sql stored function for passing empno as a parameter,calculate tax of the emp from emp table based on following condition?**

1) if annsal>10000 then tax=10% of annsal

2) if annsal>15000 then tax=20% of annsal

3) if annsal>20000 then tax=30% of annsal

Sql> create or replace function f9(p\_empno number)

return number is

v\_sal number(10);

annsal number(10);

A number(10);

begin

select sal into v\_sal from emp where empno=p\_empno;

annsal:=v\_sal\*12;

if annsal>10000 and annsal<=15000 then

A:=annsal\*0.1;

elsif annsal>15000 and annsal<=20000 then

A:=annsal\*0.2;

elsif annsal>20000 then

A:=annsal\*0.3;

else

A:=0;

end if;

return A;

end;

**o/p:** Sql> select f9(7566) from dual;

**F9 (7566)**

10710

**Q) Write pl/sql stored function for passing empno as parameter return gross salary based on following condition?**

Gross:=basic+hra+da-pf;

1. hra ----------🡪 10% of sal
2. da -----------🡪 20% of sal
3. Pf ------------🡪 30% of sal

Sql> create or replace function f10(p\_empno number)

return number is

v\_sal number(10);

hra number(10);

da number(10);

pf number(10);

gross number;

begin

select sal into v\_sal from emp where empno=p\_empno;

hra:=v\_sal\*0.1;

da:=v\_sal\*0.2;

pf:=v\_sal\*0.3;

gross:=v\_sal+hra+da-pf;

return gross;

end;

**o/p:** Sql> select f10(7566) from dual;

**F10 (7566)**

2975

**Q) Write pl/sql stored function for passing empno,date as parameter return no.of year that emp working based passed data using emp table?**

Sql> create or replace function f11(p\_empno number,p\_date date) return number is

A number(10);

Begin

Select months\_between(p\_date,hiredate)/12 into A from emp where empno=p\_empno;

return round(A);

end;

**o/p:** Sql> select f11(7566,sysdate) from dual;

**F11 (7566,SYSDATE)**

34

Sql> select empno,ename,sal,deptno,hiredate,f11(empno,sysdate)||' '|| 'YEARS' "EXPERIANCE" from emp where empno=7566;

**Empno** **Ename** **Sal** **Deptno Hiredate** **Experiance**

7566 JONES 2975 20 02-APR-81 34 YEARS

**Oracle 11g:**

Sql> select empno,ename,sal,hiredate,f11(p\_empno=>empno,p\_date=>sysdate)||' '|| 'YEARS' "EXPERIANCE" from emp where empno=7566; (Named Notation)

Before oracle 11g we are not allowed to named notation I function execution where as in oracle 11g we are allowed to use named notation when a subprogram created using SELECT statement.

**NOTE:**

We are not allowed to use DML statement in function.

**OUT:**

Using out mode we can return multiple value from the function then we can also use out parameter. But in this case all returns must belongs to same data type and also when a function out handling out parameter. Then function we are not allowed to execute using SELECT statement. That why database development not inserted to use out parameter in function.

**Q) Write pl/sql stored function for passing Deptno then return dname,loc using out parameter?**

Sql> create or replace function f12(p\_deptno in number,p\_dname out varchar2,p\_loc out varchar2)

return varchar2 is

begin

select dname,loc into p\_dname,p\_loc from dept where deptno=p\_deptno;

return p\_dname;

end;

**o/p:** sql> variable A varchar2(15);

sql> variable B varchar2(15);

sql> variable C varchar2(15);

sql> begin

:A:=f11(10,:A,:B);

end;

Sql> print A B;

**A** **B**

ACCOUNTING NEW YORK

**Cursor used in Function:** (static cursor)

In oracle we can also develop our own userdefined aggregate function same like a predefined aggregate function. Use those userdefined aggregate functions within group by. If those aggregate functions return multiple value the we must cursor within userdefined functions.

Sql> set wrap off;

Sql> create or replace function f13(p\_deptno number) return varchar2 is

Cursor C1 is select ename from emp where deptno=p\_deptno;

A varchar2(200);

Begin

For I in C1

Loop

A:=A||' '||i.ename;

End loop;

Return A;

End;

**o/p:** sql> select deptno,f13(deptno) from emp group by deptno;

**DEPTNO** **F13 (DEPTNO)**

30 ALLEN WARD MARTIN BLAKE TURNER JAMES

20 SMITH JONES SCOTT ADAMS FORD

10 CLARK KING MILLER

**Q) Write pl/sql stored function for passing empno in IN parameter ename,jobs OUT parameter then return salary of the emp from emp table?**

Sql> create or replace function f18(p\_empno in number,p\_ename out varchar2,p\_job out varchar2) return number is

V\_sal number(10);

Begin

Select ename,job,sal into p\_ename,p\_job,v\_sal from emp where empno=p\_empno;

Return v\_sal;

End;

**Execution:** (using annonyms block)

sql> declare

a number(10);

b varchar2(15);

c varchar2(15);

begin

a:=f18(7566,b,c);

dbms\_output.put\_line(a||' '||b||' '||c);

end;

**o/p:**  2975 JONES MANAGER

**NOTE:**

Oracle 10g introduced wm\_concat() predefined aggregate function which return multiple values group wise this function accepts’ all data type columns in oracle.

**Ex:** sql> select deptno,wm\_concat(ename) from emp group by deptno;

Sql> select job,wm\_concat(hiredate) from emp group by job;

In oracle all functions information stored undered user\_procedure,user\_source data dictionary.

Sql> desc user\_procedures;

Sql> desc user\_source;

Sql> select text from user\_source where name='F13';

We can also drop function using drop function functionname.

Sql> drop function functionname;

**Q) When to use procedure,when to use functions?**

1. When application return multiple values and also require DML statement then we are using procedure. Where as when application return single values then we are using functions.
2. Generally we can perform calculate using user defined function and also call this userdefined function in procedure.
3. In database return values are used select statement then we must use functions.
4. When client application required bulk of data then we must return bulk data using function.

**Procedure** **function**

Select……..into clause (Yes) YES

DML statement (Yes) YES

Expressions (No) YES

Chapter-5

**TRIGGERS**

Trigger is also same as stored procedure and also it will automatically invoked whenever DML operations perform the table (or) view

Oracle having **2** types of trigger

1. Row Level Trigger

2. Statement Level Trigger

In statement level trigger trigger body is executed only one time for DML statement, where as in row level trigger trigger body is executed for each row for DML statement.

**Syntax:** create or replace trigger triggername

Before/after insert/update/delete on tablename

trigger [for each row] ----------🡪 for row level trigger

specification [when condition]

[declare]

---🡪 variable declaration,cursors, userdefined,exceptions;

trigger Begin

body ------------------

End;

**Difference b/w statement level,row level trigger:**

Statement:

Sql> create or replace trigger sl

after update on emp

begin

dbms\_output.put\_line('Statement Level Trigger Fired');

end;

Sql> update emp set sal=sal+100 where deptno=10;

Statement Level Trigger Fired 3 rows updated.

Sql> update emp set sal=sal+100 where deptno=100;

Statement Level Trigger Fired 0 rows updated.

Row Level Trigger:

Sql> create or replace trigger rl

after update on emp

for each row

begin

dbms\_output.put\_line('Row Level Trigger Fired');

end;

Sql> update emp set sal=sal+100 where deptno=10;

Row Level Trigger Fired

Row Level Trigger Fired

Row Level Trigger Fired 3 rows updated.

Sql> update emp set sal=sal+100 where deptno=100;

0 rows updated.

**1. ROW LEVEL TRIGGER:**

In row level trigger trigger body is executed for each row for DML statement. Then we must use **for each row** clause in trigger specification. In oracle whenever we are using row level trigger then DML transaction values are automatically stored in two rollback segment qualifiers. There are

1. OLD

2. NEW

These qualifiers are used in either trigger specification (or) trigger body.

**Syntax :**old**.**columnname

**:**new**.**columnname

**NOTE:**

This qualifier is used in either in trigger specification (or) trigger body. When we are using this qualifier in WHEN clause of the trigger specification then we are not allowed to use [**:**] In front of the qualifier names.

**Syntax:** old.columnname

New.columnname

Update

Delete

Insert

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

**:new**

**:old**

Empno ename job mgr hiredate sal comm deptno

**:old**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7902 | KING | President |  | 12-DEC-81 | 5000 |  | 10 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1  :new.ename | SIMHA |  |  |  |  |  |  |

**:new**

Empno ename job mgr hiredate sal comm deptno

Sql> insert into emp(empno,ename)values(1,’SIMHA’);

**Q) Write pl/sql row level trigger on emp table whenever user inserting salary, that sal should be above 500?**

Sql> create or replace trigger t1

before insert on emp

for each row

begin

if **:**new.sal<5000 then

raise\_application\_error(-20345,'Salary Should be above 5000');

end if; end;

Sql> insert into emp(empno,sal)values(1,2000);

**ERROR:** ORA-20345: Salary Should be above 5000

Sql> insert into emp(empno,sal)values(1,7000);

**Q) Write pl/sql row level trigger whenever user modifying salaries on emp table then automatically display old salary,new salary,salary difference?**

Sql> create or replace trigger t2

after update on emp

for each row

declare

A number(10);

begin

A:=:new.sal-:old.sal;

dbms\_output.put\_line('Old Salary is'||' '||:old.sal);

dbms\_output.put\_line('New Salary is'||' '||:new.sal);

dbms\_output.put\_line('Salary Difference'||' '||A);

end;

Sql> update emp set sal=sal+100 where ename='KING'

Old Salary is 5200

New Salary is 5100

Salary Difference 100

**Q) Write pl/sql row level trigger on emp table whenever user modifying salary always new salary more than the old salary?**

Sql> create or replace trigger t3

before update on emp

for each row

begin

if :new.sal<:old.sal then

raise\_application\_error(-20345,'New Salary More Than The Old Salary');

end if;

end;

**Q) Write pl/sql row level trigger on dept table whenever user modifying deptno on dept table then automatically those modifying affected in emp table?**

Sql> create or replace trigger t4

after update on dept

for each row

begin

update emp set deptno=:new.deptno where deptno=:old.deptno;

end;

Sql> update dept set deptno=1 where deptno=10;

Sql> select \* from emp;

Sql> select \* from dept;

**Q) Write pl/sql row level trigger on emp table each deptno has at most four employees?**

Sql> create or replace trigger t5

before update or insert of deptno on emp

for each row

declare

A number(10);

begin

select count(\*) into A from emp where deptno=:new.deptno group by deptno;

if A>=4 then

raise\_application\_error(-20345,'Too Many Employee');

end if;

end;

Sql> insert into emp(empno,deptno)values(1,10);

Sql> insert into emp(empno,deptno)values(2,10);

**ERROR:** ORA-20345: Too Many Employee

**Row Level Trigger Application In Oracle:**

**Q) Write a pl/sql row level trigger on emp table implement a following business rule?**

Business rule: company does not allow any bonus to ANALYST.

Sql> create or replace trigger t6

before insert on emp

for each row

when(new.job='ANALYST')

begin

if :new.comm is not null then

:new.comm:=null;

end if;

end;

Sql> insert into emp(empno,ename,job,comm)values(1,'SIMHA','ANALYST',3000);

Sql> select \* from emp;

**Q) Write pl/sql row level trigger on emp table which acts like a primary key. i.e not to accept any duplicate data into particular column in a table?**

Sql> create or replace trigger t7

before insert on emp

for each row

declare

A number(10);

begin

select count(\*) into A from emp where empno=:new.empno;

if A>=1 then

raise\_application\_error(-20345,'We Cannot Insert Duplicate Data');

elsif A=0 then

dbms\_output.put\_line('U R Record Is Inserted');

end if;

end;

Sql>insert into emp(empno)values(7788);

**ERROR:** ORA-20345: We Cannot Insert Duplicate Data

**Q) Write pl/sql row level trigger on emp table whenever user deleting data those records stored in another table?**

Sql> create or replace trigger t8

after delete on emp

for each row

begin

insert into test(empno,ename,sal)values(:old.empno,:old.ename,:old.sal);

end;

Sql> delete from emp where sal>3000;

Sql> select \* from test;

Sql> select \* from emp;

**Q) Write pl/sql row level trigger using emp,dept tables implement all delete cascade concept without using on delete cascade clause?**

Sql> create or replace trigger t9

after delete on dept

for each row

begin

delete from emp where deptno=:old.deptno;

end;

Sql> delete from dept where deptno=10;

**Q) Write pl/sql row level trigger on emp table not to delete emp SMITH record in emp table?**

Sql> create or replace trigger t10

after delete on emp

for each row

begin

if :old.ename='SMITH' then

raise\_application\_error(-20345,'We Cannot Delete SMITH Record');

end if;

end;

Sql> delete from emp where ename='SMITH';

**ERROR:** ORA-20345: We Cannot Delete SMITH Record

**AUTO INCREMENT:**

If we want to generate primary key values automatically then we are using ‘Auto Increment’ concepts in all databases. If we want implement auto increment concepts then we are using row level trigger,sequence. i.e create a sequence in sql and use that sequence in pl/sqlrow level trigger.

Sql> create table test(sno number(10) primary key,ename varchar2(15));

Sql> create or replace trigger t11

before insert on test

for each row

begin

select s1.nextval into :new.sno from dual;

end;

Sql> insert into test(ename)values('&ename');

Enter value for ename: SIMHA

Sql> /

Enter value for ename: NARASIMHA

Sql> /

Enter value for ename: REDDY

Sql> select \* from test;

**SNO** **ENAME**

1 SIMHA

2 NARASIMHA

3 REDDY

**NOTE:**

Oracle 11g introduced variable assignment concept when we are using sequence in pl/sql block. i.e without using **dual,select……into** clause and also when we are assigning sequence value into variable.

**Syntax:** begin

Variablename:=sequencename.nextval;

End;

**Oracle 11g:**

Sql> create table test(sno number(20) primary key,name varchar2(15));

Sql> create sequence s1

Start with 1;

Sql> create or replace trigger t12

Before insert on test

For each row

Begin

**:**new**.**sno**:=**s1.nextval;

End;

**Alpha Numeric Data:**

Sql> create table test(sno varchar2(100),name varchar2(100));

Sql> create sequence s1

start with 1;

Sql> create or replace trigger t1

before insert on test

for each row

begin

select 'ABC'||lpad(s1.nextval,10,'0') into :new.sno from dual;

end;

Sql> insert into test(name)values('&name');

Enter value for name: simha

Sql> /

Enter value for name: narasimha

Sql> /

Enter value for name: narasimhareddy

Sql> select \* from test;

**SNO** **NAME**

ABC0000000001 simha

ABC0000000002 narasimha

ABC0000000003 narasimhareddy

**Ex:** sql> create or replace trigger t13

after insert on test

for each row

begin

select s1.nextval into :new.sno from dual;

end;

**ERROR**:ORA-04084: cannot change NEW values for this trigger type

**BEFORE/AFTER:**

In before trigger user inserted transaction values are affected in trigger then only those values are affected in sql engine that why in row level trigger when we modifying assign :new value then must use before trigger. Otherwise oracle server returns on error cannot change new values for this trigger type.

**Q) Write pl/sql row level trigger on emp table when ever user inserting data into ename col those inserted data automatically convert into upper case?**

Sql> create or replace trigger t14

before insert on emp

for each row

begin

:new.ename:=upper(:new.ename);

end;

Sql> insert into emp(empno,ename)values(1,'narasimha');

Sql> select \* from emp;

**AUDITING A COLUMN:**

Whenever we are modifying data in a particular then those transaction values stored in another table is called “Auditing a Column”. In oracle database auditing a column application implement throw row level trigger.

Sql> create table test(oldempno number(10),oldsal number(10),newsal number(10), hiredate date,username varchar2(15));

Sql> create or replace trigger t18

after update of sal on emp

for each row

begin

insert into test values(:old.empno,:old.sal,:new.sal,sysdate,user);

end;

Sql> update emp set sal=sal+100 where deptno=10;

Sql> select \* from test;

**OLDEMPNO** **OLDSAL** **NEWSAL** **HIREDATE** **USERNAME**

7782 2650 2750 03-NOV-14 SCOTT

7839 5000 5100 03-NOV-14 SCOTT

7934 1500 1600 03-NOV-14 SCOTT

**NOTE:**

In oracle we are not allowed to use :new,:old qualifier infornt of the sysdate,user pseudo columns. Whenever we are using sqlloader tool then also we are not allowed to use [:] infornt of the sysdate,user pseudo columns.

**2. Statement Level Trigger:**

In statement level trigger trigger body is executed only once for execution. In statement level trigger we are not allowed to use :new,:old and when clause and for each row clause. Generally statement level trigger used in time component based condition.

**Q) Write pl/sql statement level trigger on emp table not to perform DML operations in sat,sun?**

Sql> create or replace trigger t15

before insert or update or delete on emp

begin

if to\_char(sysdate,'DY')in('SAT','SUN') then

raise\_application\_error(-20123,'We Cannot Perform DML on SAT,SUN DAYS');

end if;

end;

Sql> delete from emp where empno=7902;

**ERROR:** ORA-20123: We Cannot Perform DML on SAT,SUN DAYS

**NOTE:**

We are not allowed to use WHEN clause in statement level trigger. In all database statement level trigger performance very high compare to row level trigger.

Sql> create or replace trigger t15

before insert or update or delete on emp

for each row

when(to\_char(sysdate,'DY')in('SAT','SUN'))

begin

raise\_application\_error(-20123,'We Cannot Perform DML on SAT,SUN DAYS');

end;

**Q) Write pl/sql statement level trigger on emp table not allowed perform DML operation in last day of the month?**

Sql> create or replace trigger t16

before insert or update or delete on emp

begin

if sysdate=last\_day(sysdate) then

raise\_application\_error(-20123,'We Cannot perform DMLs Last Day Of The Months');

end if;

end;

Sql> delete from emp where empno=7902;

**ERROR:** ORA-20123: We Cannot perform DMLs Last Day Of The Months

**TRIGGER EVENTS (or) PRDICATE:**

In all database triggering events are used in either row level (or) in statement level trigger this are inserting,deleting,updating clause. These events are also called “Trigger Predicate” clause. Whenever we are using no.of trigger body and also defined no.of conditions then we are using triggering events these events are used in trigger body these events multiple operation and using multiple tables.

**Syntax:** if inserting then

Statement;

Elsif updating then

Statement;

Elsif deleting then

Statement;

End if;

**Q) Write pl/sql statement level trigger on emp table not to perform DML operations every day using triggering events?**

Sql> create or replace trigger t17

before insert or update or delete on emp

begin

if inserting then

raise\_application\_error(-20123,'We Cannot Perform Insertion');

elsif updating then

raise\_application\_error(-20124,'We Cannot Perform updation');

elsif deleting then

raise\_application\_error(-20125,'We Cannot Perform deletion');

end if;

end;

Sql> delete from emp where empno=7902;

**ERROR:** ORA-20125: We Cannot Perform deletion

Sql> insert into emp(empno)values(1);

**ERROR**: ORA-20123: We Cannot Perform Insertion

Sql> update emp set sal=sal+100 where deptno=10;

**ERROR:** ORA-20124: We Cannot Perform updation

**Q) Write pl/sql trigger program to perform DML operation how many times execute trigger that msg store another table?**

Sql> create table test(msg varchar2(900));

Sql> create or replace trigger t19

after insert or update or delete on emp

declare

A varchar2(900);

begin

if inserting then

A:='Rows Are inserted';

elsif updating then

A:='Rows Are updated';

elsif deleting then

A:='Rows Are deleted';

end if;

insert into test values(A);

end;

Sql> insert into emp(empno)values(1);

Sql> insert into emp(empno)values(2);

Sql> update emp set sal=sal+100 where deptno=10;

Sql> delete from emp where empno in(1,2);

Sql> select \* from test;

**MSG**

Rows Are inserted

Rows Are inserted

Rows Are updated

Rows Are deleted

**Q) Write pl/sql row level trigger on emp table when ever user inserting data those values are stored in another table. Whenever user modifying data those data stored another table whenever user deleting data those data stored in another table?**

Sql> create table test1(empno number(10),ename varchar2(15),salary number(10));

Sql> create table test2(empno number(10),ename varchar2(15),salary number(10));

Sql> create table test3(empno number(10),ename varchar2(15),salary number(10));

Sql> create or replace trigger t20

after insert or update or delete on emp

for each row

begin

if inserting then

insert into test1(empno,ename,salary)values(:new.empno,:new.ename,:new.sal);

elsif updating then

insert into test2(empno,ename,salary)values(:old.empno,:new.ename,:new.sal);

elsif deleting then

insert into test3(empno,ename,salary)values(:old.empno,:old.ename,:old.sal);

end if;

end;

Sql> delete from emp where sal>3000;

Sql> select \* from test3;

**EMPNO** **ENAME** **SALARY**

7839 KING 5000

**Execution order of the trigger:**

1. Before Statement Level Trigger

2. Before Row Level Trigger

3. After Row Level Trigger

4. After Statement Level Trigger

Sql> create table test(sno number(10));

Sql> create or replace trigger t1

after insert on test

for each row

begin

dbms\_output.put\_line('After Row Level Trigger');

end;

Sql> create or replace trigger t2

after insert on test

begin

dbms\_output.put\_line('After Statement Level Trigger');

end;

Sql> create or replace trigger t3

before insert on test

for each row

begin

dbms\_output.put\_line('Before Row Level Trigger');

end;

Sql> create or replace trigger t4

before insert on test

begin

dbms\_output.put\_line('Before Statement Level Trigger');

end;

Sql> insert into test values(10);

Before Statement Level Trigger

Before Row Level Trigger

After Row Level Trigger

After Statement Level Trigger

**Compound trigger:**

Oracle 11g introduced compound trigger. Compound trigger allows different block within a trigger to be executed at different timing points compound trigger also having a global declaration section same like a packages.

**Syntax:** create or replace trigger triggername

For insert/update/delete on tablename

Compound trigger

-----------🡪 global variable declarations;

Before statement is

Begin

------------------

End [before statement];

Before each row is

Begin

------------------

End [before each row];

After each row is

Begin

-------------------

End [after each row];

After statement is

Begin

--------------------

End [after statement];

End;

**Q) Write pl/sql compound trigger to display trigger execution order?**

Sql> create table test(sno number(10));

Sql> create or replace trigger t1

For insert on test

Compound trigger

Before statement is

Begin

Dbms\_output.put\_line(‘Before Statement Level’);

End before statement;

Before each row is

Begin

Dbms\_output.put\_line(‘Before Each Row Level’);

End before each row;

After each row is

Begin

Dbms\_output.put\_line(‘After Each Row Level’);

End after each row;

After statement is

Begin

Dbms\_output.put\_line(‘After Statement Level’);

End after statement;

End;

**Testing:** sql> insert into test value(1);

Before statement level

Before row level

After row level

After statement level

Sql> create table test(sno number(10));

Sql> create or replace trigger t1

After insert on emp

Begin

Dbms\_output.put\_line(‘Trigger1 fired’);

End;

Sql> create or replace trigger t2

After insert on emp

Begin

Dbms\_output.put\_line(‘Trigger2 Fired’);

End;

Sql> create or replace trigger t3

After insert on emp

Begin

Dbms\_output.put\_line(‘Trigger3 Fired’);

End;

Sql> set serveroutput on;

Sql> insert into test values (10);

Trigger3 fired

Trigger2 fired

Trigger 1 fired

**Solution:**

Step-1: first,third triggers are same as above code

Step-3: sql> create or replace trigger t2

After insert on test

Follows t1,t3

Begin

Dbms\_output.put\_line(‘Trigger2 fired’);

End;

**Testing:** sql> insert into test values(10);

Trigger1 fired

Trigger2 fired

Trigger3 fired

**FOLLOWS CLUSE:** (Oracle 11g)

Oracle 11g introduced follows clause in trigger. Whenever we are defining same level of trigger on same table we cannot con troll execution order of the trigger. To overcome this problem oracle 11g introduced follows clause. Which is used follows clause control the explicitly execution order of the trigger follows clause provides generally execution order of the trigger follows clause specifies in trigger in trigger specifies only.

**Syntax:** create or replace trigger triggername

Before/after insert/update/delete on tablename

[for each row]

[when condition]

Follows another triggername1,………….

[declare]

------------------

Begin

------------------

End [triggername];

**Program:**

Sql> create table test(col1 number(10),col2 number(10),col3 date);

Sql> create sequence s1

start with 1234;

Sql> create or replace trigger t1

before insert on test

for each row

begin

select s1.nextval into :new.col1 from dual;

dbms\_output.put\_line('Trigger1 fired');

end;

Sql> create or replace trigger t2

before insert on test

for each row

begin

select reverse(to\_char(:new.col1)) into :new.col2 from dual;

dbms\_output.put\_line('Trigger2 fired');

end;

Sql> set serveroutput on;

Sql> insert into test(col3)values(sysdate);

Trigger2 fired

Trigger1 fired

1 row created.

Sql> select \* from test;

**Col1 col2 col3**

1234 05-Nov-14

**Solution:** (Oracle 11g)

Step-1: same as first trigger above code

Step-2: sql> create or replace trigger t2

before insert on test

for each row

follows t1

begin

select reverse(to\_char(:new.col1)) into :new.col2 from dual;

dbms\_output.put\_line('Trigger2 fired');

end;

**Testing:** sql> insert into test(col3)values(sysdate);

Trigger1 fired

Trigger2 fired

Sql> select \* from test;

**Col1 col2 col3**

1234 4321 05-Nov-14

**Q) Write pl/sql trigger on emp table when ever user declaring record on emp table automatically displayed remaining number of records number in bottom of the delete statement?**

Sql> insert into emp(empno)values(1);

Sql> create or replace trigger t21

after delete on emp

declare

A number(10);

begin

select count(\*) into A from emp;

dbms\_output.put\_line(A);

end;

SQL> delete from emp where empno=1;

14

1 row deleted.

SQL> delete from emp where empno=7902;

13

1 row deleted.

Sql> select \* from emp;

**Ex:** sql> create or replace trigger t22

after delete on emp

for each row

declare

A number(10);

begin

select count(\*) into A from emp;

dbms\_output.put\_line(A);

end;

Sql> delete from emp where empno=7902;

**ERROR**: ORA-04091: table SCOTT.EMP is mutating

**MUTATING ERROR:**

If we are row level trigger based on a table then trigger body can not read data from same table and we can not perform DML operation on same table. If we trying this oracle server return an error: ora-4091: Table is mutating. This error is called “Mutating Error” and also this table is called “Mutating Table” and also this is called “Mutating Trigger”.

Mutating error is only runtime error. Mutating error occurred row level trigger. Mutating error doe not occurs statement level trigger. Generally in all database systems when we are use statement level trigger DML transaction values are automatically commit in to database. That why using trigger body we can read committed data without any problem that why statement level trigger does not return mutating error. Where as in row level trigger DML transaction not committed automatically when we are try to read without committed data using trigger body then database server return mutating error.

In oracle we are using autonomous transaction for avoiding mutating errors. Autonomous transaction automatically avoids mutating error. But this transaction return previous results.

**Syntax:** sql> create or replace trigger triggername

Before/after insert/update/delete on emp

For each row

Declare

Pragma autonomous\_transaction;

--------------------------------------

Begin

--------------------------------------

Commit;

End;

**Program:** sql> create or replace trigger t1

after delete on emp

for each row

declare

pragma autonomous\_transaction;

A number(10);

begin

select count(\*) into A from emp;

commit;

dbms\_output.put\_line(A);

end;

Sql> set serveroutput on;

Sql> delete from emp where empno=7902;

14

Sql> delete from emp where empno=7566;

14

**Calling a Procedure into Trigger:**

In oracle using call statement we can also call procedure into trigger.

**Syntax:** create or replace trigger triggername

Before/after insert/delete/update on tablename

Call procedurename

Sql> create or replace procedure p1 is

v\_sal number(10);

begin

delete from test;

select sum(sal) into v\_sal from emp;

insert into test values(v\_sal);

end;

Sql> create or replace trigger t1

after insert or update or delete on emp

call p1

Sql> update emp set sal=sal+100;

14 rows updated.

Sql> select \* from test;

**TOTALSALARY**

30825

**WHEN CONDITION:**

We can also specify logical condition in trigger specification through WHEN clause. But WHEN clause are used in row level trigger only when ever we are using WHEN clause not allowed to use [:] infornt of the qualifier name in WHEN clause. In WHEN clause we are not allowed to use pl/sql expression i.e in WHEN clause we must use sql Boolean expression when condition always return value either true (or) false when condition true then only trigger body is executed.

**NOTE:**

In WHEN clause we are not allowed to use sub query.

Sql> create or replace trigger t23

after insert on emp

for each row

when(new.empno=1)

begin

dbms\_output.put\_line('When empno=1 then only body is executed');

end;

Sql> insert into emp(empno)values(1);

When empno=1 then only body is executed

1 row created.

Sql> insert into emp(empno)values(2);

1 row created.

**Q) Write pl/sql row level trigger on emp table when ever user inserting job as CLERK then only system date is stored into another table using WHEN clause?**

Sql> create table test(col date);

Sql> create or replace trigger t24

after insert on emp

for each row

when(new.job='CLERK')

begin

insert into test values(sysdate);

end;

Sql> insert into emp(empno,job) values(1,'CLERK');

Sql> select \* from test;

**COL**

05-NOV-14

**System trigger:**

In all database system trigger created database administrator. System trigger database administrator create trigger on database level and schema level. These types of trigger are called “System Trigger”. In schema level trigger database administrator uses DDL commands in trigger specifications. That why trigger are also called “DDL Trigger”.

**Syntax:**  create or replace trigger triggername

Before/after create/alter/drop/truncate/rename on scott.schema

Declare

-----------------------

Begin

-----------------------

End;

**Ex:** sql> create or replace trigger t25

After create on database

Begin

Dbms\_output.put\_line(‘Some User Creating Database Object’);

End;

Sql> create table test(sno number(10));

Some User Creating Database Object

**Q) Write pl/sql trigger on scott.schema not to drop emp table?**

Sql> create or replace trigger t1

Before drop on scott.schema

Begin

If ora\_dict\_name=’EMP’ and ora\_dict\_obj\_type=’TABLE’ then

Raise\_application\_error(-20123,’We Cannot Drop EMP Table’);

End if;

End;

Sql> drop table emp;

Ora-20123: We Cannot Drop EMP Table

Oracle support having 12 type trigger of trigger based on statement level,row level,before,after,DML statement and also oracle support instead of trigger on views,system trigger on database.

Statement level trigger -------------🡪 before/after --------🡪 insert/update/delete

6 <= 2 \* 3

Row level trigger -------------🡪 before/after --------🡪 insert/update/delete

6 <= 2 \* 3

--------🡪 instance trigger and system trigger

**Enable /Disable:**

Oracle 11g introduced enable,disable clause within trigger specification it self.

**Syntax:** create or replace trigger triggername

Before/after insert/delete/update on tablename

[for each row]

[when condition]

[follows anothertriggername]

[enable/disable]

[declare]

-------------

Begin

--------------

End;

**Enable/disable single trigger:**

**Syntax:** alter trigger triggername enable/disable;

**Enable/disable:**

**Syntax:** alter table tablename enable/disable all trigger;

Sql> alter table emp disable all triggers;

All information stored undered user\_triggers data dictionary;

Sql> desc user\_triggers;

We can also drop trigger triggername.

Sql> drop trigger triggername;

Chapter-6

**PACKAGES**

Package is an database object which encapsulate global variable, constant, types, procedure, functions, cursors into single unit.

Generally package **does not accepts** parameters and also not be nested and also can not be invoked directly.

Generally packages are used to **improve performance** of the application. Because when ever we are calling package subprograms first time then automatically total packages loaded into RAM memory area. When ever user calling subsequent subprogram calls then oracle server calling those subprogram from RAM memory this process automatically reduces disk I/O. packages having 2 parts

1. Package Specification

2. Package Body

In package specification we are declaring the objects where as in package body we are implementing those objects. By default package specifications are public and package body is private.

**Package Specification Index:**

**Syntax:** create or replace package packagename is/as

---🡪 Global Variable, Declaration, Constant Declarations;

---🡪 Cursor Declarations;

---🡪 Times Declarations;

Public

---🡪 Procedure Declarations;

---🡪 Function Declarations;

End;

**Package Body:**

**Syntax:** create or replace package body packagename is/as

-------🡪 Procedure Implementations;

Private

-------🡪 Function Implementations;

End;

**Executing packaged subprograms:**

Executing packaged procedure:

Method-1:

**Syntax:** sql> exec packagename.procedurename(actual parameters);

Method-2: (using anonymous block)

**Syntax:** sql> begin

Packagename.procedurename(actual parameter);

End;

**Executing packaged functions:**

Method-1: (using select statement)

**Syntax:** sql> select packagename, functionname(actual parameter) from dual;

Method-2: (using anonymous block)

**Syntax:** sql> begin

Variablename:=packagename.functionname(actual parameters);

End;

Sql> create or replace package pk1 is

procedure p1;

procedure p2;

end;

Sql> create or replace package body pk1 is

procedure p1 is

begin

dbms\_output.put\_line('First Procedure');

end p1;

procedure p2 is

begin

dbms\_output.put\_line('Second procedure');

end p2;

end;

Sql> exec pk1.p1;

First Procedure

Sql> exec pk1.p2;

Second procedure

**Global Variable:**

In oracle global variable are defined in package specification only.

Sql> create or replace package pk2 is

A number(10):=700;

procedure p1;

function f1(B number) return number;

end;

Sql> create or replace package body pk2 is

procedure p1 is

B number(10);

begin

B:=A/2;

dbms\_output.put\_line(B);

end p1;

function f1(B number) return number is

begin

return A\*B;

end f1;

end;

Sql> exec pk2.p1;

350

Sql> select pk2.f1(8) from dual;

**PK2.F1 (8)**

5600

**State of the Global Variable:**

If we want to maintain state of the global variable is state of the globalized cursor then we must use serially\_reusable pragma in package.

**Syntax:** pragma serially\_reusable;

Sql> create or replace package pk3 is

G number(10):=10;

pragma serially\_reusable;

end;

Sql> begin

pk3.G:=40;

end;

Sql> begin

dbms\_output.put\_line(pk3.G);

end;

**o/p:** 10

**State of the cursor:**

Sql> create or replace package pk4 is

Cursor C1 is select \* from emp;

Pragma serially\_reusable;

End;

Sql> begin

open pk4.C1;

end;

**Overloading Procedure:**

Overloading refocus to same name can be used for different perform. We can also implement overloading in oracle through packages this procedure having same name and difference type are difference number of arguments.

Sql> create or replace package pk5 is

procedure p1(A number,B number);

procedure p1(X number,Y number);

end;

Sql> create or replace package body pk5 is

procedure p1(A number,B number) is

Z number(10);

begin

Z:=A+B;

dbms\_output.put\_line(Z);

end p1;

procedure p1(X number,Y number) is

Z number(10);

begin

Z:=X-Y;

dbms\_output.put\_line(Z);

end p1;

end;

Sql> exec pk5.p1(A=>8,B=>4);

12

Sql> exec pk5.p1(X=>6,Y=>9);

-3

**Forward Declaration:**

Declaring subprogram in packages body is called **‘Forward Declaration’**. That is whenever we are calling procedure into another procedure then only **‘Forward Declaration’**. That is whenever we are calling local procedure first we must implement local procedure before calling otherwise oracle on error to overcome this problems we must use forward declaration.

Sql> create or replace package pk6 is

procedure p1;

end;

Sql> create or replace package body pk6 is

procedure p2;

procedure p1 is

begin

p2;

end p1;

procedure p2 is

begin

dbms\_output.put\_line('Local Procedure');

end p2;

end;

Sql> exec pk6.p1;

Local Procedure

**TYPES USED IN PACKAGES:**

In oracle we can also create our own userdefined type using type keyword. In oracle we are create in two step process. First we creating type appropriate syntax then only we creating variable that type.

Pl/sql having following type:

1. pl/sql record

2. Index by table (or) pl/sql table (or) associative array

Composite datatype

Collections

3. Nested table

4. Varray

5. Ref cursor

**Index by table:**

Index by table is an userdefined type, which is used to store multiple data items in single unit. Index by table is an “Unbound” table. Here indexes are either integers (or) characters. These integers are positive, negative numbers. Basically index by table key\_value pairs. That that is here value fields actual data and key fields index values. This key behaviors like a primary key. This key stores either +ve (or) –ve. It does not accept duplicate index.

Key value

Key value pair

Primary key

|  |
| --- |
| Smith |
| Allen |
| Scott |
| Clerk |
| King |

|  |
| --- |
| 10 |
| 20 |
| 30 |
| 40 |
| 50 |

Generally index by table ar used to improves performance of the application. Because those tables are stored in RAM memory area that’s way these tables are also called as “Memory Table”. To improve performance of the application oracle provided binary\_integer datatype for key field in index by table. If we want to process index by table data then we are using following collection methods. These are exits,first,last,prior,next,delete(index),delete (index1, indexn),delete,count.

This is a userdefined type so we are creating in two step process, first we are creating type then only we are creating a variable of that type.

**Syntax:** 1**.** type typename is table of datatype(size) index by binary\_integer;

2. variablename typename;

Sql> declare

type t1 is table of number(10) index by binary\_integer;

v\_t t1;

begin

v\_t(1):=10;

v\_t(2):=20;

v\_t(3):=30;

v\_t(4):=40;

dbms\_output.put\_line(v\_t(1));

dbms\_output.put\_line(v\_t.first);

dbms\_output.put\_line(v\_t.last);

dbms\_output.put\_line(v\_t.prior(3));

dbms\_output.put\_line(v\_t.next(2));

dbms\_output.put\_line(v\_t.count);

v\_t.delete;

dbms\_output.put\_line(v\_t.count);

end;

**o/p:** 10 value

Key

|  |
| --- |
| 10 |
| 20 |
| 30 |
| 40 |

|  |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |

1

4

2

3

V\_t

4

0 binary\_integer

**Q) Write pl/sql program to transfer all employees names from emp table and storing into index by table and also display content from index by table?**

Sql> declare

type t1 is table of varchar2(10) index by binary\_integer;

v\_t t1;

cursor C1 is select ename from emp;

n number(10):=1;

begin

open C1;

loop

fetch C1 into v\_t(n);

exit when C1%notfound;

n:=n+1;

end loop;

close C1;

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i));

end loop;

end;

**o/p: ename**

Smith

-------

Miller

When resource table having large amount of data and also if we are transferring data using cursor then oracle server automatically degrades performance of application to overcome this problem oracle 8i introduced bulk collect clause. When we are using bulk collect clause oracle server improve performance of the application. Because bulk collect clause internally works based on columns when we are using bulk collect clause oracle server selects columns at a time and transfers that data into collections.

**Syntax:** select \* bulk collect into collectionvarname from tablename where condition;

Sql> declare

type t1 is table of varchar2(10) index by binary\_integer;

v\_t t1;

begin

select ename bulk collect into v\_t from emp;

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i));

end loop;

end;

**o/p: ename**

Smith

-------

Miller

**Q) Write pl/sql program next 10 date into index by table and display content from index by table?**

Sql> declare

type t1 is table of date index by binary\_integer;

v\_t t1;

begin

for i in 1..5

loop

v\_t(i):=sysdate+i;

end loop;

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i));

end loop;

end;

**o/p:** 11-NOV-14

12-NOV-14

13-NOV-14

14-NOV-14

15-NOV-14

**Q) Write pl/sql program to transfer all emp’s joining date from emp table into index by table and also display content from index by table?**

Sql> declare

type t1 is table of date index by binary\_integer;

v\_t t1;

begin

select hiredate bulk collect into v\_t from emp;

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i));

end loop;

end;

**pl/sql record:**

This is an userdefined type which is used to represent different datatype into single unit. It is also same as structure in ‘C’ language. This is a userdefined so we are creating in two step process first we are creating type then only we are creating a variable of type.

**Syntax:** 1) type typename is record(attribute1 datatype(size),attribute2 datatype(size),.....);

2) Variablename typename;

Sql> declare

type t1 is record(a1 number(10),a2 varchar2(15),a3 number(10));

v\_t t1;

begin

v\_t.a1:=100;

v\_t.a2:='NARASIMHA';

v\_t.a3:=5000;

dbms\_output.put\_line(v\_t.a1||' '||v\_t.a2||' '||v\_t.a3);

end;

**o/p:** 100 NARASIMHA 5000

Sql> create or replace package pk4 is

type t1 is record(a1 number(10),a2 varchar2(20),a3 number(10));

procedure p1;

end;

Sql> create or replace package body pk4 is

procedure p1 is

v\_t t1;

begin

select empno,ename,sal into v\_t from emp where ename='KING';

dbms\_output.put\_line(v\_t.a1||' '||v\_t.a2||' '||v\_t.a3);

end p1;

end;

Sql> exec pk4.p1;

7839 KING 5000

**NOTE:**

Index by table having following collection method this are exists,first,last,prior,next, count, delete(index),delete(index1,indexn),delete.

value

|  |
| --- |
| Nara |
| Simha |
| Reddy |
| Abc |

|  |
| --- |
| a |
| b |
| c |
| d |

X

Varchar2(10)

V\_t

Varchar2(20)

Sql> declare

type t1 is table of varchar2(15) index by varchar2(15);

A:=’b’; A:=v\_t.next(A);

**SIMHA**

**REDDY**

**ABC**

v\_t t1;

A varchar2(15);

begin

v\_t('a'):='NARA';

v\_t('b'):='SIMHA';

A:='a'; A:=v\_t.prior(A);

**NARA**

v\_t('c'):='REDDY';

A:='d'; A:=v\_t.prior(A);

ABC

REDDY

SIMHA

NARA

v\_t('d'):='ABC';

A:='a';

loop

dbms\_output.put\_line(v\_t(A));

A:=v\_t.next(A);

exit when A is null;

end loop;

end;

**o/p:** NARA

SIMHA

REDDY

ABC

**NOTE-1:**

In index by table we can also use key field as character datatype in this case if we want displayed data we are not allowed to use for loop because for loop index variable internally user integer datatype. In oracle database datatype in cursors searching records through forward directions only. If we want to travel a record backend direction also then we go for index by table.

**Using record datatype in value field:**

Key

Value

|  |
| --- |
|  |
|  |
|  |
|  |

|  |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |

%row type

X

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7902 | Smith | Clerk | 7566 | 12-dec-82 | 2900 | 1400 | 30 |

V\_t

V\_t

Empno ename job mgr hiredate sal comm deptno

Binary\_integer

**Ex:** Sql> declare

type t1 is table of emp%rowtype index by binary\_integer;

v\_t t1;

begin

select \* bulk collect into v\_t from emp;

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i).ename||' '||v\_t(i).sal||' '||v\_t(i).hiredate);

end loop;

end;

**o/p:** SMITH 800 17-DEC-80

----------- ----- --------------

MILLER 1500 23-JAN-82

**Return result sets:**

If we want to return large amount of data from database server into client application then we are using following two methods in oracle.

Method-1: using ref cursor

Method-2: using index by table

Whenever we are implementing this type of application 1st we must develop a database server application which returns bulk of data and also execute these applications using client application.

**Method-1: (**using index by table)

If we want to return bulk of data from oracle server into client application then we are using functions which returns large of data.

**Database server application:**

**Q) Write pl/sql program which returns emp details from emp table into client application using an index by table?**

Sql> create or replace package pk6 is

type t1 is table of emp%rowtype index by binary\_integer;

function f1 return t1;

end;

Sql> create or replace package body pk6 is

function f1 return t1 is

v\_t t1;

begin

select \* bulk collect into v\_t from emp;

return v\_t;

end f1;

end;

execution: (using pl/sql client)

Sql> declare

A pk6.t1;

begin

A:=pk6.f1;

for i in A.first..A.last

loop

dbms\_output.put\_line(A(i).ename||' '||A(i).sal||' '||A(i).hiredate);

end loop;

end;

**o/p:** SMITH 800 17-DEC-80

----------- ----- --------------

MILLER 1500 23-JAN-82

**Exists Collection method:**

Exists collection method is used in index by table,nested table,Varray. Exists collection method also returns Boolean value either true (or) false. Generally exists collection method is used to test whether requested data is available in collection (or) not.

**Syntax:** collectionvariablename.exists(indexvalue)

**Ex:** Sql> declare

type t1 is table of number(10) index by binary\_integer;

v\_t t1;

A boolean;

begin

v\_t(1):=10;

v\_t(2):=20;

v\_t(3):=30;

v\_t(4):=40;

A:=v\_t.exists(3);

if A=true then

dbms\_output.put\_line('U R Requested Index Exists With Having an Element'||' '||v\_t(3));

else

dbms\_output.put\_line('U R Requested Index does not exists');

end if;

dbms\_output.put\_line(v\_t(1));

dbms\_output.put\_line(v\_t.prior(3));

dbms\_output.put\_line(v\_t.next(3));

dbms\_output.put\_line(v\_t.last);

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i));

end loop;

end;

**o/p:** U R Requested Index Exists With Having an Element 30

10

2

4

4

10

20

30

40

**Ex:** Sql> declare

type t1 is table of varchar2(10) index by binary\_integer;

v\_t t1;

begin

select ename bulk collect into v\_t from emp;

v\_t.delete(3);

|  |
| --- |
| Smith |
| Allen |
|  |
| Jones |
| Scott |

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i)); gaps

end loop;

end;

**o/p:** SMITH

ALLEN

**ERROR:** ORA-01403: no data found

Whenever index by table or nested table having gaps ands also when we are try to display these collection data then oracle server returns an error: ora-1403: No Data Found. To overcome this problem this problem we must use ‘EXISTS’ collection method.

**Solution:**

Sql> declare

type t1 is table of varchar2(10) index by binary\_integer;

v\_t t1;

begin

select ename bulk collect into v\_t from emp;

v\_t.delete(3);

for i in v\_t.first..v\_t.last

loop

if v\_t.exists(i) then

dbms\_output.put\_line(v\_t(i));

end if;

end loop;

end;

**o/p:** Smith

--------

Miller

**Nested table,varray:**

Oracle 8.0 introduced nested table,varrays. These are userdefined type,which is used to store no.of data items in single unit.

Before we are storing data into nested table,Varray then we must use constructors, constructor name is also same as type name.

**Nested Table:**

Oracle 8.0 introduced nested tables this is an user defined type, which is used to store no.of dataitems in single unit. Nested values,Varray are started with 1. Nested table is also in unbound table, these indexes are consecutive. Generally we are not allowed to store index by table permanently into database. To overcome this problem oracle 8.0 introduced extension of the index by table called “Nested Table” which is used to store permanently into database using SQL.

Generally in index by table we can not add (or) remove indexes, where as in nested by table we can add (or) remove indexes using extend trim collection methods. This is an userdefined type so we are creating in two step process 1st we are creating type then only we are creating variable of that type.

**Syntax: 1)** type typename is table of datatype (size);

2) variablename typename:=typename();

Sql> declare

type t1 is table of number(10) index by binary\_integer;

v\_t t1;

begin

v\_t(500):=90;

dbms\_output.put\_line(v\_t(500));

end;

**o/p:** 90

Index by table are basically sparse. i.e we are not allocating memory explicitly.

**Ex:** Sql> declare

type t1 is table of number(10);

v\_t t1:=t1();

begin

v\_t(500):=90;

dbms\_output.put\_line(v\_t(500));

end;

**o/p: ERROR:** ORA-06533: Subscript beyond count

in nested table we must reserved indexes explicitly by using extend collection method.

**Ex:** Sql> declare

type t1 is table of number(10);

v\_t t1:=t1();

begin

v\_t.extend(500);

v\_t(500):=90;

dbms\_output.put\_line(v\_t(500));

end;

**o/p:** 90

**Ex:** Sql> declare

type t1 is table of number(10);

v\_t t1:=t1(10,20,30,40);

begin

dbms\_output.put\_line(v\_t.first);

dbms\_output.put\_line(v\_t.last);

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i));

end loop;

end;

**o/p:** 1

4

10

20

30

40

**Ex:** Sql> declare

type t1 is table of number(10);

v\_t1 t1;

v\_t2 t1:=t1();

begin

if v\_t1 is null then

dbms\_output.put\_line('v\_t1 is null');

else

dbms\_output.put\_line('v\_t1 is not null');

end if;

if v\_t2 is null then

dbms\_output.put\_line('v\_t2 is null');

else

dbms\_output.put\_line('v\_t2 is not null');

end if;

end;

**o/p:** v\_t1 is null

v\_t2 is not null

1. Type t1 is table of number(10);

V\_t1 t1; null

V\_t1

1. Type t1 is table of number(10);

V\_t2 t1:=t1(); -------------------------------

V\_t1

1. Declare

Type t1 is table of number(10);

Begin

V\_t.extend;

V\_t(1):=10;

Dbms\_output.put\_line(v\_t(1));

End; 10 --------------------

1

V\_t

**Ex:** Sql> declare

type t1 is table of number(10);

v\_t t1:=t1(10,20,30,40);

begin

dbms\_output.put\_line(v\_t.first);

**o/p:** 1

4

2

3

5

10

20

30

40

50

0

dbms\_output.put\_line(v\_t.last);

dbms\_output.put\_line(v\_t.prior(3));

dbms\_output.put\_line(v\_t.next(2));

v\_t.extend;

v\_t(5):=50;

v\_t.extend(2,4);

v\_t.trim(2);

dbms\_output.put\_line(v\_t.count);

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i));

end loop;

v\_t.delete;

dbms\_output.put\_line(v\_t.count);

end;

10 20 30 40 50 --------------------

V\_t

1 2 3 4 5 --------------------

**Q) Write pl/sql program to transfer all emp names from emp table into nested table and also contents from nested table?**

Sql>

**o/p:** SMITH

----------

MILLER

**Varray:**

Oracle 8.0 introduced Varray. This is an userdefined datatype which is used to store number of data items in a single unit maximum limit is 2GB data. Before we are storing data into Varray we must initialize using constructor. This is an userdefined type so we are creating two step process.

**Syntax:** 1) type typename is Varray(maxsize) of datatype(size);

2) variablename:=typename();

**Ex:** Sql> declare

type t1 is varray(10) of varchar2(10);

v\_t t1:=t1('a','b','c','d');

A boolean;

begin

dbms\_output.put\_line(v\_t.limit);

dbms\_output.put\_line(v\_t.count);

dbms\_output.put\_line(v\_t.first);

dbms\_output.put\_line(v\_t.last);

dbms\_output.put\_line(v\_t.prior(4));

dbms\_output.put\_line(v\_t.next(2));

A:=v\_t.exists(3);

if A=true then

dbms\_output.put\_line('index 3 exists with having an element'||' '||v\_t(3));

else

dbms\_output.put\_line('u r index does not exists');

end if;

v\_t.extend(2);

v\_t(5):='e';

v\_t(6):='f';

v\_t.extend(4,5);

v\_t.trim(2);

dbms\_output.put\_line(v\_t.count);

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t.(i));

end loop;

v\_t.delete;

dbms\_output.put\_line(v\_t.count);

end;

**o/p:**

**NOTE:**

I varray we are not allowed to delete particular index (or) range of index using delete collection on method. But we can delete all the elements using delete collection method.

**Q) Write pl/sql program to transfer first 100 emp’s from emp table and storing into varray and also display content from varray method?**

Sql> declare

type t1 is varray (10) of varchar2(10);

v\_t t1:=t1();

begin

select ename bulk collect into v\_t from emp where rownum<=10;

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i));

end loop;

end;

**o/p:** SMITH

-----------

TURNER

Sql> declare

type t1 is table of varchar2(10) index by binary\_integer;

v\_t t1;

begin

select ename bulk collect into v\_t from emp;

v\_t.delete(3);

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i));

end loop;

end;

**o/p:** SMITH

ALLEN

**ERROR**: ORA-01403: no data found

**NOTE:**

Whenever collection having gaps then if we are try to total data from collection then oracle server returns an **error: ora-1403: No Data Found.** To overcome this problem we must exists collection methods.

Sql> declare

type t1 is table of varchar2(10) index by binary\_integer;

v\_t t1;

begin

select ename bulk collect into v\_t from emp;

for i in v\_t.first..v\_t.last

loop

if v\_t.exists(i) then

dbms\_output.put\_line(v\_t(i));

end if;

end loop;

end;

**Without bulk bind:**

**Oracle server**

Pl/sql runtime engine

Pl/sql block

Begin

For i in c1

Loop

Update emp set sal=sal\*0.1

Where empno=i.empno;

End loop;

End;

Procedure statement executor

SQL Engine

**Bulk bind:**

**Oracle server**

Pl/sql runtime engine

Pl/sql block

Begin

For in in c1

Loop

Update emp set sal=sal\*0.1

Where empno=v\_t(i);

End loop;

End;

Procedure statement executor

SQL engine

Bulk bind is a two step process that is before we are using actual bulk bind through FORALL statement we must fetch the date from resource into collection by using bulk collect clause.

Step-1: fetching data from resource into collection using bulk collect.

Step-2: using FORALL statement process all data at a time using SQL engine (actual bulk bind).

**Step-1**:

1. Select …………into clause
2. Cursor ……….fetch clause
3. DML ……………returning ………into clause.

**1. Bulk collect used in select…….into clause:**

**Syntax:** select \* bulk collect into collectionvarname from tablename where condition;

**Ex:** sql> declare

type t1 is table of emp%rowtype index by binary\_integer;

v\_t t1;

begin

select \* bulk collect into v\_t from emp;

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i).ename);

end loop;

end;

**o/p:**  SMITH

----------

MILLER

**2. Bulk collect used in cursor……fetch statement:**

**Syntax:** fetch cursorname bulk collect into collectionname [limit anynumber]

Fetch c1 bulk collect into v\_t,v\_t1; Cursor c1 select ename,job from emp;

111111 Open c1;

Oracle database

Emp

|  |  |  |  |
| --- | --- | --- | --- |
| Empno | Ename | Job | Sal |
|  |  |  |  |
|  |  |  |  |

|  |  |
| --- | --- |
| Ename | Job |
| ----------  ----------  ----------  ----------  ----------  ---------- | ----------  ----------  ----------  ----------  ----------  ---------- |

V\_t t1

111111

V\_t1

**Ex:** sql> declare

type t1 is table of varchar2(10)

index by binary\_integer;

v\_t t1;

v\_t1 t1;

cursor c1 is select ename,job from emp;

begin

open c1;

fetch c1 bulk collect into v\_t,v\_t1;

close c1;

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i)||' '||v\_t1(i));

end loop;

end;

**o/p:** SMITH CLERK

---------- ----------

FORD ANALYST

**Calculate elapsed time in pl/sql block:**

Bulk collect clause always improves performance of the application because bulk collect works based on column wise selection. If we want to calculate elapsed time then oracle provided dbms\_utility.get\_time method from utility bulk. This method always returns number datatype.

**Syntax:** variablename:=dbms\_utility.get\_time;

Fetch c1 bulk collect into v\_t,v\_t1; Cursor c1 select owner,object\_name from all\_objects;

111111 Open c1;

Oracle database

Emp

|  |  |  |  |
| --- | --- | --- | --- |
| Empno | Ename | Job | Sal |
|  |  |  |  |
|  |  |  |  |

|  |  |
| --- | --- |
| Owner | Object\_name |
| ----------  ----------  ----------  ----------  ----------  ---------- | ----------------  ----------------  ----------------  ----------------  ----------------  ---------------- |

V\_t t1

111111

V\_t1

Sql> desc all\_objects;

Sql> select count(\*) from all\_objects;

**COUNT (\*)**

40727

**Ex:** Sql> declare

type t1 is table of all\_objects.object\_name%type index by binary\_integer;

v\_t t1;

v\_t1 t1;

cursor c1 is select owner,object\_name from all\_objects;

z1 number(10);

z2 number(10);

begin

z1:=dbms\_utility.get\_time;

for i in c1

loop

null;

end loop;

z2:=dbms\_utility.get\_time;

dbms\_output.put\_line('elapsed time for normal fetch'||' '||(z2-z1)||' '||'hsecs');

z1:=dbms\_utility.get\_time;

open c1;

fetch c1 bulk collect into v\_t,v\_t1;

close c1;

z2:=dbms\_utility.get\_time;

dbms\_output.put\_line('elapsed time for bulk fetch'||' '||(z2-z1)||' '||'hsecs');

end;

**o/p:** elapsed time for normal fetch 84 hsecs

elapsed time for bulk fetch 76 hsecs

**3. Bulk collect used in DML…….returning ……..into clauses:**

Returning into clauses are used in DML statement only when we are processing large amount of data we are can also use bulk clause in returning into clauses through collections.

**Ex:** sql> variable A varchar2(10);

Sql> update emp set sal=sal-100 where ename='KING' returning job into :A;

Sql> print a;

**A**

PRESIDENT

Sql> declare

type t1 is table of number(10) index by binary\_integer;

v\_t t1;

begin

update emp set sal=sal-100 where job='CLERK' returning sal bulk collect into v\_t;

dbms\_output.put\_line('affected no.of clerks on'||' '||sql%rowcount);

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i));

end loop;

end;

**o/p:** affected no.of clerks on 4

700

1000

850

1400

**Step-2**:

If we want improves performance of the application we must reduces context switches between SQL,PL/SQL engine using FORALL statement that is once data available in collection. We are processing collection data using SQL engine.

**Syntax:** forall indexvarname in collectionname.first..collectionname.last

DML statement where colname=collectionname(indexvarname);

**Ex:** sql> declare

type t1 is varray(10) of number(10);

v\_t t1:=t1(10,20,30,40,50);

begin

forall i in v\_t.first..v\_t.last

update emp set sal=sal+100 where deptno=v\_t(i);

end;

sql> select \* from emp;

**Q) Write pl/sql to retrieve all empno’s from emp table and storing into index by table using bulk collect clause and also modify salary of the emp at a time using FORALL statement (actual bulk bind)?**

Sql> declare

type t1 is table of number(10)

index by binary\_integer;

v\_t t1;

begin

select empno bulk collect into v\_t from emp;

forall i in v\_t.first..v\_t.last

update emp set sal=sal+100 where empno=v\_t(i);

end;

Sql> select \* from emp;

**EX:** sql> declare

type t1 is table of emp.empno%type

index by binary\_integer;

v\_t t1;

begin

select empno bulk collect into v\_t from emp;

v\_t.delete(3);

forall i in v\_t.first..v\_t.last

update emp set sal=sal+100 where empno=v\_t(i);

end;

**o/p: error:** ORA-22160: element at index [3] does not exist

Index by table (or) nested table having gaps we are not allowed to bulk bind process to overcome this problem oracle 10g introduced indices of clause in bulk bind process.

**Syntax:** forall indexvarname in indices of collectionvarname

DML statement where colname=collectionvarname(index variablename);

**Before oracle 10g** **oracle 10g (indices of)**

First

Empno

First

Empno

|  |
| --- |
| V\_t(i)=7369 |
| V\_t(i)=7566 |
| V\_t(i)=7839 |
| V\_t(i)=7866 |

|  |
| --- |
| 7369 |
|  |
| 7566 |
|  |
| 7839 |
|  |
| 7866  Last |

|  |
| --- |
| 7369 |
| Update all rows based on array  Update all rows based on array limiting |
| 7902 |
|  |
| 7788  Last |

**solution:** (using indices of)

Sql> declare

type t1 is table of emp.empno%type

index by binary\_integer;

v\_t t1;

begin

select empno bulk collect into v\_t from emp;

v\_t.delete(3);

forall i in indices of v\_t

update emp set sal=sal+100 where empno=v\_t(i);

end;

sql> select \* from emp;

**sql%bulk\_rowcount:**

if we want to view affected number of rows in each process after bulk bind then we are using sql%bulk\_rowcount attribute.

Sql> declare

type t1 is varray(10) of number(10);

v\_t t1:=t1(10,20,30,40,50);

begin

forall i in v\_t.first..v\_t.last

update emp set sal=sal+100 where deptno=v\_t(i);

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line('affected number of rows in deptno'||' '||v\_t(i)||' '||'is'||' '||sql%bulk\_rowcount(i));

end loop;

end;

**o/p:** affected number of rows in deptno 10 is 3

affected number of rows in deptno 20 is 4

affected number of rows in deptno 30 is 7

affected number of rows in deptno 40 is 0

affected number of rows in deptno 50 is 0

**bulk insert:**

sql> declare

type t1 is table of number(10)

index by binary\_integer;

v\_t t1;

begin

select empno bulk collect into v\_t from emp;

forall i in v\_t.first..v\_t.last

insert into target values(v\_t(i));

end;

sql> select \* from target;

**bulk delete:**

sql> declare

type t1 is varray(10) of number(10);

v\_t t1:=t1(10,20,30,40,50);

begin

forall i in v\_t.first..v\_t.last

delete from emp where deptno=v\_t(i);

end;

sql> select \* from emp;

**dbms\_utility package:**

Using this package we are calculate elapsed time through get\_time method. These methods always return number datatype.

**Syntax:** variablename:=dbms\_utility.get\_time;

This package internally having index by table this package having two methods.

1. comma\_to\_table

2. table\_to\_comma

**1. comma\_to\_table:**

This method is used to convert comma separated into index by table.

**Syntax:** dbms\_utility.comma\_to\_table(stringname,binary\_integer variablename, index by table variable);

**2. table\_to\_comma:**

Using this method is used convert index by table value into comma separated string.

**Syntax:** dbms\_utility.table\_to\_comma (index by table,binary\_integer varname,stringname);

Before we are using these methods we must from dbms\_utility package in declare section of the pl/sql block.

**Syntax:** index by table varname dbms\_utility.uncl\_array;

**Q) Write pl/sql program to convert comma separate string into index by table using dbms\_utility package and also display context from index by table?**

Sql> declare

v\_t dbms\_utility.uncl\_array;

A binary\_integer;

str varchar(200);

begin

str:='a,b,c,d,e,f';

dbms\_utility.comma\_to\_table(str,A,v\_t);

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i));

end loop;

end;

**o/p:** a

b

c

d

e

f

**Q) Write pl/sql program to convert all dname into dept table comma separated string using dbms\_utility package?**

Sql> declare

v\_t dbms\_utility.uncl\_array;

A binary\_integer;

str varchar(200);

begin

select dname bulk collect into v\_t from dept;

dbms\_utility.table\_to\_comma(v\_t,A,str);

dbms\_output.put\_line(str);

end;

**o/p:** ACCOUNTING,RESEARCH,SALES,OPERATIONS

**REF CURSOR:**

Oracle 7.2 introduced ref cursor. Ref cursor is a userdefined which is used to process multiple record and also **RECORD BY RECORD** process. Generally in static cursors database servers executes only one select statement at a time for a single active set area where as in ref cursor database server execute **number of select statement dynamically** for a single active set area. That why this cursor are also called as “Dynamic Cursor”.

Generally we are not allowed to pass static cursor as parameter to the subprogram. To overcome this problem ANSI/ISO sql introduced ref cursor which is used to cursor. Because basically ref cursor userdefined type. Generally static cursor does not return values into client application where as return values into client application(**.**net,java,php,…………..).

Oracle having two types of ref cursor

1. strong ref cursor

2. week ref cursor

Strong ref cursor is return type and week ref cursor is does not return type. This is an userdefined type so we are creating in two step process first we creating type then only a variable of that type. That’s why this cursor is also called as “Cursor Variable”.

**Syntax:** 1. Type typename is ref cursor return recordtype datatype;

Variablename typename;

2. Type typename is ref cursor;

Variablename typename;

In ref cursor we must specifies select statement through open……… for statement only this statement are used in executable section of the pl/sql block.

**Syntax:** open refcursorvarname for select statements;

Sql> declare

type t1 is ref cursor;

v\_t t1;

i emp%rowtype;

begin

open v\_t for select \* from emp where sal>2000;

loop

fetch v\_t into i;

exit when v\_t%notfound;

dbms\_output.put\_line(i.ename||' '||i.sal);

end loop;

end;

**o/p:** JONES 2975

BLAKE 2850

CLARK 2650

SCOTT 3000

KING 5000

FORD 3000

**Q) Write pl/sql program using ref cursor when ever user enter deptno 10th then display 10th deptno from emp table where as when ever user enter deptno 20th display 20th deptno is from dept table?**

Sql> declare

type t1 is ref cursor;

v\_t t1;

i emp%rowtype;

j dept%rowtype;

v\_deptno number(10):=&deptno;

begin

if v\_deptno=20 then

open v\_t for select \* from emp where deptno=10;

loop

fetch v\_t into i;

exit when v\_t%notfound;

dbms\_output.put\_line(i.ename||' '||i.sal||' '||i.deptno);

end loop;

elsif v\_deptno=10 then

open v\_t for select \* from dept where deptno=20;

loop

fetch v\_t into j;

exit when v\_t%notfound;

dbms\_output.put\_line(j.deptno||' '||j.dname||' '||j.loc);

end loop;

end if;

close v\_t;

end;

**o/p:** Enter value for deptno: 10

20 RESEARCH DALLAS

Sql> /

Enter value for deptno: 20

CLARK 2650 10

KING 5000 10

MILLER 1500 10

**Package ref cursor as parameter to the subprogram:**

Sql> create or replace package pk1 is

Type t1 is ref cursor return emp%rowtype;

Type t2 is ref cursor return dept%rowtype;

Procedure p1(v\_t1 out t1);

Procedure p2(v\_t2 out t2);

End;

Sql> create or replace package body pk1 is

Procedure p1(v\_t1 out t1) is

Begin

Begin

Open v\_t1 for select \* from emp

End p1;

Procedure p2(v\_t2 out t2) is

Begin

Open v\_t2 for select \* from dept

End p2;

End;

**Execution:** sql> variable A ref cursor;

Sql> variable B ref cursor;

Sql> exec pk1.p1(:A);

Sql> exec pk2.p2(:B);

Sql> print A B;

**NOTE:**

If oracle we are not allowed to ref cursor directly in package.

**Sys\_refcursor:**

Oracle 9i introduced sys\_refcursor predefined type is place of week cursor.

**Syntax:** refcursorname sys\_refcursor;

Sql> declare

v\_t sys\_refcursor;

i emp%rowtype;

begin

open v\_t for select \* from emp where rownum<=3;

loop

fetch v\_t into i;

exit when v\_t%notfound;

dbms\_output.put\_line(i.ename||' '||i.sal);

end loop;

end;

**o/p:** SMITH 800

ALLEN 1600

WARD 1250

**Return result sets using ref cursor:**

Using ref cursor also we can return large amount of data from oracle database into client application in this case we must develop database server application using function.

Sql> create or replace function f1(A varchar2)

return sys\_refcursor is

v\_t sys\_refcursor;

begin

open v\_t for A;

return v\_t;

end;

Sql> select f1('select \* from emp where sal>=3000') from dual;

**F1('SELECT\*FROMEMPWH**

CURSOR STATEMENT: 1

**EMPNO** **ENAME** **JOB** **MGR** **HIREDATE** **SAL** **COMM** **DEPTNO**

7788 SCOTT ANALYST 7566 19-APR-87 3000 20

7839 KING PRESIDENT 17-NOV-81 5000 10

7902 FORD ANALYST 7566 03-DEC-81 3000 20

**Utl\_file package:**

Oracle 7.3 introduced utl\_file package. This package is used to using a data into operating system file and also record a data os file. If we want using data into file then we are putf() method from utl\_file package. If we want to read data from file then we are using get\_time() from utl\_file method. Before we are using utl\_file package we must create an alias directory related to directory using following directory.

**Syntax:** create or replace directory directoryname as ‘path’;

Before we are creating alias directory database administrator use create any directory privileges.

**Syntax:** grant create any directory to username1,username2,………..;

Sql> conn sys as sysdba;

Enter password: SYS

Sql> grant create any directory to scott;

Sql> conn scott/tiger

Sql> create or replace directory ABC as 'c:\';

Before we are permanently read/write operation we are using read,write alias directory using syntax.

**Syntax:** grant read,write on directoryname to username1,username2,………..;

**Ex:** sql> conn sys as sysdba;

Enter password: SYS

Sql> conn system/tiger

Sql> grant read,write on directory ABC to scott;

**Writing data into an os file:**

Step-1: before we are opening the file we must declare the point variable in declare section of the pl/sql block using file type from utl\_file package.

**Syntax:** filepointvarname utl\_file.file\_type;

Step-2: before we are storing data we must open the file fopen from utl\_file package the method accepts three parameters index return file\_type.

**Syntax:** filepointvarname:=utl\_file.fopen(‘alias directoryname’,’filename’,’mode’);

Mode-write(w), read(r),append(a)

Step-3: to write data into a file then we are using putf() procedure from utl\_file packages.

**Syntax:** utl\_file.putf(filepointvarname,’actual data’);

Step-4: after writing data into file then we must close the file using ‘fclose’ procedure from utl\_file package.

**Syntax:** utl\_file.fclose(filepointvariablename);

**Q) Write a pl/sql program to retrieve all employee names from emp table and storing into an external file using utl\_file package?**

Sql> declare

fp utl\_file.file\_type;

begin

fp:=utl\_file.fopen('ABC','GNSR.txt','w');

utl\_file.putf(fp,'Happy New Year 2015 And Happy Sankranthi');

utl\_file.fclose(fp);

end;

Generally when we are writing a table data into a file then oracle server stores columns data into horizontal data mannar in os file,to overcome this problem if we want to store our own format then we are using %s access specifier,\n in the second parameters of the putf procedure.

**Q) Write pl/sql program to retrieve all employee name from emp table and storing into an external file using utl\_file package?**

Sql> declare

fp utl\_file.file\_type;

cursor c1 is select ename from emp;

begin

fp:=utl\_file.fopen('ABC','file1.txt','w');

for i in c1

loop

utl\_file.putf(fp,'%s\n',i.ename);

end loop;

utl\_file.fclose(fp);

end;

**Reading data from an os file:**

To read data from file then we are using get\_file procedure from utl\_file package. Before we are using these procedure we must use read mode(r) in fopen().

**Syntax:** utl\_file.get\_line(filepointervariablename,buffervariablename);

**Q) Write pl/sql prgm read data from gnsr.txt using utl\_file package and display the data?**

Sql> declare

fp utl\_file.file\_type;

A varchar2(200);

begin

fp:=utl\_file.fopen('ABC','GNSR.txt','r');

utl\_file.get\_line(fp,A);

dbms\_output.put\_line('Data from file'||' '||A);

utl\_file.fclose(fp);

end;

**Q) Write pl/sql program using utl\_file package read all ename from emp.txt file and display those data?**

Sql> declare

fp utl\_file.file\_type;

A varchar2(200);

begin

fp:=utl\_file.fopen('ABC','file1.txt','r');

loop

utl\_file.get\_line(fp,A);

dbms\_output.put\_line(A);

end loop;

utl\_file.fclose(fp);

exception

when no\_data\_found then

null;

end;

**o/p:** SMITH

----------

MILLER

**NOTE:**

In oracle when we are reading data from external file using utl\_file package. Then oracle server returns an error ORA-1403: NO DATA FOUND when end of file reached to handle this we must use NO\_DATA\_FOUND exception name.

Chapter-7

**SQL LOADER**

Sqlloader is an utility program which is used to transfer data from flat file into oracle database this tool is also called as “BULK LOADER”. Always sqlloader execute control file this file extension is **.**ctl. Sqlloader is very high performance.

Always sqlloader executes controls. Based on the type of flat file,we are creating control file and submit to the sqlloader file,then only sqlloader transfer data from flat file into oracle database. During these process sqlloader automatically creates log file,this file stores all other file information and also store loaded,rejected number of record numbers and also this file stores oracle errors with error messages.

Control file(**.ctl)**

**--------------------------------------------------------------------------------------------------------**

Sqlloader.exe

Target table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Stores rejected record

Stores rejected record

Discord file

Log file

Bad file

**.**bad **.**log **.**dsc

Oracle database

**Based on some reasons:**

Some records are rejected during loading process these rejected records are automatically stored in bad files,discard files. Bad files stores rejected records based on datatype mismatch,business rule violation. Where as discard file stores rejected records based on when clause condition within control file

**Flat file:**

Flat file is on structure file which contains number of records all systems having 2 types of flat files.

1. Variable record flat file

2. Fixed record flat file

**1. Variable record flat file:**

A flat file which contains delimeters is called “Variable record flat file”.

**Ex:** 101,ABC,3000

102,DEF,4000

103,GHI,8000

104,JKL,9000

**2. Fixed record flat file:**

A flat file which does not have delimeters is called “Fixed record flat file”.

**Ex:** 101 ABC 2000

102 DEF 3000

103 GHI 8000

104 JKL 9000

Sqlloader always executes control file that why based on the type of flat file we are creating control file and submit to the sqlloader then only sqlloader transfer data from flat file into oracle database.

**Control file(.ctl):**

**1. creating a control file for variable record flat file:**

Always control file execution start with ‘load data’ clause. After load data clause we must specify path of the flat file using ‘infile’ clause.

**Syntax:** load data

Infile ‘path of flat file’

**NOTE:**

We can also use flat file data with in control file itself in this case we must use ‘\*’. In place of flat file path of the ‘infile’ clause and also we must specify begin data clause in the above of the flat file data. Some time control file also having flat file.

**Ex:** load data

Infile \*

------------

Begindata

101,abc,3000

102,def,4000

103,ghi,8000

After infile clause we are using ‘infile table tablename’ clause to store into oracle database. Before ‘infile table tablename’ clause we are using either insert (or) truncate (or) append (or) replace clauses.

**NOTE:**

By default clause is insert. If target table is an empty table then we are using insert. After ‘into table tablename’ clause we are specifying clauses based on the flat file data these are

1. Fields terminated by

2. Optionally enclosed by

3. Trailing nullcols

After specify these clauses we must use target table column with in parenthesis ( () ).

**Control file(.**ctl**):**

**Syntax:** load data

Infile ‘path of flat file’

Bad file ‘path of bad file’

Discord file ‘path of discord file’

Insert/truncate/replace/append

Into table tablename

Fields terminated by ‘delimetername’

Optionally enclosed by ‘delimetername’

Trailing nullcols(col1,col2,col3,…………)

-------🡪 After that save file **filename.ctl**

**Ex:** first we are writing a flat file some data like below (notepad).

101,abc,3000

102,def,4000

103,ghi,8000

---------🡪 After that file save file **filename.txt**

**Invoking sqlloader:**

Start -------🡪 Run --------🡪 cmd ------🡪 d:\>sqlldr

User id=scott/tiger

Sql> create table target(empno number(10),ename varchar2(15),sal number(10));

**Creating control file:**

Load data

Infile ‘d:\file1.txt’

Insert into table target

Field terminated by ‘,’

(empno,ename,sal)

----🡪 file save and **save as type** to change to test document to **all files** and also change the **filename.ctl**.

**Execution:** d:\>sqlldr user id=scott/tiger

Control=d:\simha.ctl

After the above process go to sql prompt to check the data store (or) not.

Sql> select \* from target;

During this execution process sqlloader automatically creates that long file as same name as control file. This long file shows another information loaded,registered and also stores oracle error numbers,errormsgs.

(or)

Load data

Infile \*

Insert into table target

Fields terminated by ‘,’

(empno,ename,sal)

Begindata

101,abc,3000

102,def,4000

103,ghi,8000

**Constant,filler clauses used in control file:**

If we want to store default values into oracle database through sqlloader then we are using constant within control files.

**Syntax:** columnname constant ‘actual value’

If flat file having less number of fields and target table require more number of fields then only we are using constant does not. If flat file having more number of columns and also target table required less number of columns then only we are using filler clause.

**Syntax:** anyname filler

101,abc

102,def

103,ghi

Sql> create table target(empno number(10),loc varchar2(10));

**Control file:**

Load data

Infile ‘d:\file1.txt’

Insert into table target fields terminated by ‘,’

(empno,ename filler,loc constant ‘AP’)

**Execution:** d:\>sqlldr userid=scott/tiger

Control=d:\simha.ctl

Sql> select \* from target;

**Empno** **loc**

101 AP

102 AP

103 AP

**Bad file:**

Bad file stores rejected records those file extension is bad file (**.**bad). If we are not specify bad file clause also sqlloader automatically creates bad file as same name as flat file name.

1. Datatype mismatch

2. Business rule violate

**1. Datatype mismatch:**

**Ex:** 101,abc,5000

‘102’,xyz,3000

‘103’,def,6000

104,pqr,9000

Sql> create table target(empno number(10),ename varchar2(15),sal number(10));

**Control file(**simha.ctl**):**

Load data

Infile ‘d:\file1.txt’

Insert into table target

Fields terminated by ‘,’

(empno,ename,sal)

**Bad file:**

‘102’,xyz,3000

‘103’,def,6000

**2. Business rule violate:**

If we want to maintain proper data into oracle database when defined using check constraint this is called “Data Integrity”.

**Ex:** 101,abc,5000

102,xyz,3000

103,def,6000

104,pqr,2000

Sql> create table target(empno number(10),ename varchar2(15),sal number(10) check(sal>4000);

**Control file(**simha.ctl**):**

Load data

Infile ‘d:\file1.txt’

Insert into table target

Fields terminated by ‘,’

(empno,ename,sal)

**Bad file:**

102,xyz,3000

104,pqr,2000

**NOTE:**

When ever flat file contains null values then sqlloader rejected those null value record and also stores records bad file to overcome this problem we are using trailing nullcol’s clause in control file.

**Flat file:** 101,abc,3000

102,xyz

103,efg

104,stu,8000

Sql> create table target(empno number(10),ename varchar2(15),sal number(10));

**Control file:**

Load data

Infile ‘d:\file1.txt’

Insert into table target

Fields terminated by ‘,’

Trailing nullcols

(empno,ename,sal)

**Recnum clause used in control file:**

Recnum clause automatically assign’s numbers to loaded,rejected scripted number of records.

**Ex:** 101,abc

102,xyz

103,def

104,sdg

Sql> create table target(empno number(10),ename varchar2(15),rno number(10));

**Control file:**

Load data

Infile ‘d:\file1.txt’

Insert into table target

Fields terminated by ‘,’

(empno,ename,rno recnum)

Sql> select \* from target;

**Empno** **ename** **rno**

101 abc 1

104 sdg 4

**Date used in control file:**

Method-1: using date datatype

Method-2: using to\_date() conversion function

**Method-1:**

**Syntax:** colname date “flat file date format”

**Flat file:** 101,abc,05112014

102,def,02122014

103,pqr,21042014

Sql> create table target(empno number(10),ename varchar2(15),hiredate date);

**Control file:** load data

Infile ‘d:\file1.txt’

Insert into table target

Fields terminated by ‘,’

(empno,ename,hiredate date “DD/MM/YYYY”)

**Function used in control file:**

We can also use oracle predefined,userdefined functions within control file in this case we must use “within functions functionality” and also we must use [:] infornt of the column in function functionality.

**Syntax:** colname “functionname(:colname)”

**Flat file:**  101,abc,M

102,def,F

103,pqr,M

104,xyz,F

Sql> create table target(empno number(10),ename varchar2(15),gender varchar2(10));

**Control file:** load data

Infile ‘d:\file1.txt’

Insert into table target

Fields terminated by ‘,’

(empno,ename,gender “decode(:gender,’M’,’male’,’F’,’female’)”)

Sql> select \* from target;

**Empno Ename Gender**

101 abc male

102 def female

103 pqr male

104 xyz female

**Method-2:** (using to\_date function)

**Control file:** load data

Infile ‘d:\file1.txt’

Insert into table target

Fields terminated by ‘,’

(empno,ename,hiredate “to\_date(:hiredate,’DD/MM/YYYY’)”)

**Discard file:**

Discard file is also store r based on **when** clause condition file. WHEN clause used after “into table tablename” clause. Discard file are store rejected records and also called as “discard trigger” this file extension is (**.**dsc).

**Syntax:** when condition

Generally bad file created automatically but where as we must specify discard file explicitly within control file using discard file clause.

**Syntax:** discard file ‘path of the discard file’

**Flat file:** 101,abc,10

102,def,20

103,pqr,10

104,xyz,30

105,jkl,10

Sql> create table target(empno number(10),ename varchar2(15),deptno number(10));

**Control file:** load data

Infile ‘d:\file1.txt’

Discardfile ‘d:\abc.dsc’

Insert into table target

When deptno=’10’

Fields terminated by ’,’

(empno,ename,deptno)

**Discard file:** 102,def,20

104,xyz,30

**NOTE-1:**

**When** clause condition values must be specify within single cotes (‘ ‘).

**NOTE-2:**

In **when** clause we are not allowed to use other then =,<> relational operators only.

**NOTE-3:**

In **when** clause we are allowed to use logical operator ‘OR’. Then we are allowed logical operator ‘AND’.

**Sequences used in Control file:**

We can also use sequences in control file. In this case we must specify functionality within double cotes (“ “).

**Syntax:** colname “sequencename.nextval”

**Flat file:** 101

102

103

104

Sql> create table target(empno number(10));

Sql> create sequence s1

Start with 1;

**Control file:** load data

Infile ‘d:\file1.txt’

Insert into table target

Fields terminated by ‘,’

(empno.”s1.nextval”)

**Creating a Control file for fixed record flat file:**

A flat file which does not have delimeters is called “fixed record flat file”. When we are using fixed record flat file then we must use **position** clause within control file and also along with **position** clause. We must specify sqlloader datatypes. We must specify starting and ending position at the every field using [:] operator.

Sqlloader having 3 datatypes

1. integer external

2. decimal external

3. char

**Syntax:** colname position(starting position**:**ending position) sqlloader datatype

**NOTE:**

When we are using functions (or) expressions in ctl file then we are not allowed to use sqlloader datatype along with **position** clause.

**Syntax:** colname position(starting position**:**ending position) “functionname(:colname)”

**Flat file:** 101abc2000

102def3000

103xyz7000

104pqr9000

Sql> create table target(empno number(10),ename varchar2(15),sal number(10));

**Control file:** load data

Infile ‘c:\file1.txt’

Insert into table target

(empno position(01:03) integer external,

Ename position(04:09) char,

Sal position(10:15) integer external)

**NOTE:**

In fixed record flat file we can also use trailing nullcols clause and also we are optionally enclosed by clauses. But we are not allowed use fields terminated by clause.

**Data manipulation:** date **simha.ctl**

Number(10)

**Target table** number() varchar() load data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sno | Value | Time | Col1 | Col2 | Col3 |
|  |  |  |  |  |  |

Infile \*

Insert into table target

(sno “s1.nextval”,value constant ‘50’,

Time “to\_char(sysdate,’HH:MI:SS’)”,

Col1 position(01:09) “:col1/100”,

Col2 position(10:14) “upper(:col2)”,

Create sequence s1 HH:MI:SS col3 position(15:20) “to\_date(:col3,’DDMMYY’)”)

Start with 1; begindata

Col1/100 to\_date() 100000000 AAAAAAA 051215

Upper() 200000000 BBBBBBBB 090714

Using sqlloader we can also transfer number of flat file data into single target table. In this case we are using number of infile clause within control file. Using sqlloader we can also transfer single flat file into number of target tables in this case we are using into table tablename clauses. But if resource having different database data and also resource contains combination flat file at we are not allowed sqlloader. To overcome this problems explicitly tools using this are informatica,MSBI,Data stage.

**LOBS (Large Objects):**

Oracle 8.0 introduced large objects(LOBS). Lobs are predefined type which stores large amount of data. In oracle if we want store more 2000bytes of alpha numeric data then we are using varchar2 datatype. This datatypes stores 4000bytes if we want store more than 4000bytes of alphabet numeric data than we are using long datatype. Its store upto 2GB data.

**Syntax:** colname long

In all database there can be only one long and also we are not allowed to create primary key on that column to overcome this problem oracle 8.0 introduced ‘clob’ datatype.

**Syntax:** colname clob

Sql> create table test(col1 long);

Sql> create table test2(col1 long,col2 long);

**Error:** ORA-01754: a table may contain only one column of type LONG

Sql> create table test3(col1 long primary key);

**ERROR**: ORA-02269: key column cannot be of LONG datatype

We want to store binary date then we are using ‘RAW’ datatype this datatype store upto 2000bytes.

**Syntax:** colname raw(size)

If we want to store more than 2000bytes of binary datatype then we are using ‘long raw’ datatype.

**Syntax:** colname long raw

But there can be only one long raw to overcome this problem oracle 8.0 introduced blob.

**Syntax:** colname blob

Sql> create table test(col1 raw(200));

Sql> desc test;

**Name** **Null?** **Type**

COL1 RAW(200)

Sql> create table test1(col1 long raw);

Sql> desc test1;

**Name** **Null?** **Type**

COL1 LONG RAW

All database systems having two types of large objects

1. internal large objects

2. external large objects

**1. internal large objects:**

Internal large objects are stored within tablespace. Internal large objects are store within database oracle having two types of internal large objects.

1. clob(character large objects) ------------🡪 colname clob

2. blob(binary large objects) ---------------🡪 colname blob

**2. external large objects:**

External large objects are stored in outside of the database. That is this objects are stored in OS file extension large objects is an bfile datatype.

**Syntax:** colname bfile

**Difference b/w long,lobs datatype:**

**Long** **lobs**

1. Can contain upto 2GB data. 1. Can contain 4GB data.

2. Table contain only one long datatype column. 2. Table can contain more than one lob col.

3. Subquery cannot select a long datatype col. 3. Subquery can select lob datatype col.

**LOB Locator:**

When we are defining internal large objects then inserted data not store directly within table columns in place of this one a locator is create that locator only points to where actual data stored in data segment. This pointer is called ‘LOB Locator’ and also where storing data place is called ‘lob value’.

|  |  |  |
| --- | --- | --- |
| Sno | Clob | Bfile |
| 1 | Lob locator |  |

Lob value

Gnsr.jpg

**Empty\_clob (or) Empty\_blob():**

This two function are used in SQL DML. These function are used to initialize lob locator to empty locator insert,update statement.

**NOTE:**

Before we start writing data into lob using either OCI(oracle call interface) are using dbms\_lob package then we must initialize lob column into empty locator using empty\_clob, empty\_blob.

**Difference b/w null,empty\_clob() function:**

Sql> create table test(sno number(10),col2 clob);

Sql> insert into test values(1,’abc’);

Sql> insert into test values(2,null);

Sql> insert into test values(3,empty\_clob());

Sql> select \* from test;

**SNO** **COL2**

1 abc

2

3

Sql> select \* from test where col2 is null;

**SNO** **COL2**

2

Sql> select \* from test where col2 is not null;

**SNO** **COL2**

1 abc

3

**Storing large amount of data into clob column (or) storing an image into blob column using dbms\_lob package:**

**Step-1:** Before we are storing large amount of data or image into lob columns then we must create alias directory related to physically directory using following syntax.

**Syntax:** create or replace directoryname as ‘PATH’;

Before we are creating alias directory dba must give create any privileges.

**Syntax:** grant create any privilege any directory to username;

Sql> conn sys as sysdba;

Enter password: SYS

Sql> grant create any directory to scott;

Sql> conn scott/tiger;

Sql> create or replace directory XYZ as ‘D:\’;

**Step-2:** create table in database using lob column.

Ex: create table test(sno number(10),col1 clob);

**Step-3:** Develop a pl/sql block to data (or) image into lob column using dbms\_lob package.

**Develop pl/sql block:**

**Step-1:** in declare section of the pl/sql block we are declared in lob,bfile variables using following syntax.

**Syntax:** declare

Varname1 clob;

Varname1 bfile;

**Step-2:** before we are using dbms\_lob package, we must initialize lob locator into empty locator using empty\_clob() and also we must store locator value into upper appropriate variable using returning into clause.

**Syntax:** insert into tablename(clobname)values(empty\_clob()) returning clob colname into variablename;

This stmt is used in executable section of the pl/sql block.

**Step-3:** we must specify actual file along with alias directory using bfilename function. This function accepts two parameters and returns bfile datatype.

**Syntax:** bfile\_variablename**:**=bfilename(‘alias directoryname’,’filename’);

**Step-4:** before we are loading data into lob column we must open the file using file open procedure from ‘dbms\_lob’ package.

**Syntax:** dbms\_lob.fileopen(bfilevarname);

**Step-5:** using load from file procedure we can load data (or) image into lob column. This procedure accepts 3 parameters. These are

1. Lob variablename

2. bfile variablename

3. length of bfile

**NOTE:**

To return length of the bfile we must use get length function from dbms\_lob package.

**Syntax:** dbms\_lob.getlength(bfilevarname);

**Step-6:** we must close the file using file close procedure after processing.

**Syntax:** dbms\_lob.fileclose(bfilevarname);

Sql> create table test(sno number(10),col1 clob);

Sql> declare

v\_clob clob;

v\_bfile bfile;

Begin

Insert into test values(1,empty\_clob()) returning col1 into v\_clob;

v\_bfile:=bfilename('ABC','GNSR.txt');

dbms\_lob.fileopen(v\_bfile);

dbms\_lob.loadfromfile(v\_clob,v\_bfile,dbms\_lob.getlength(v\_bfile));

dbms\_lob.fileclose(v\_bfile);

End;

Sql> select \* from test;

**SNO** **COL1**

1 Happy New Year 2015 And Happy Sankranthi

**NOTE:**

We can also store large amount of data,image using bfile datatype in this case we must using bfile name function in insert statement but we are not allowed to display this data in sql\*plus environment.

Ex: sql> create table test(sno number(10),col1 bfile);

Sql> insert into test values(1,bfilename('ABC','GNSR.txt'));

1 row created.

Sql> select \* from test;

**Error:** Column or attribute type can not be displayed by SQL\*Plus.

Chapter-8

**NESTED BLOCK**

Block within another block is called “NESTED BLOCK”. Generally child block are also called as “NESTED BLOCK”. In nested block we can also use same variable name in parent, child block. Generally by default child block executable statement always access child block variables only. If we want to access parent block variable then we must use parent block name within executable statement of the child block infornt of variable using ( **• )** operator. Before this process we must specify parent block name within (<< >>). To use exception propagation.

**Ex:** <<parent>>

Declare

V\_empno emp.empno%type:=&empno1;

Begin

Declare

V\_empno emp.empno%type:=&empno2;

V\_ename emp.ename%type;

Begin

Select ename into v\_ename from emp where empno=parent.v\_empno;

Dbms\_output.put\_line(v\_ename);

End;

End;

**o/p:** Enter value for empno1: **7902**

Enter value for empno2: **7566**

**FORD**

When pl/sql blocks contain number of select……..into clauses and also if we want to handle exception propagation then we must use nested block.

Sql> declare

V\_empno1 emp.empno%type:=&empno1;

V\_ename emp.ename%type;

V\_empno2 emp.empno%type:=&empno2;

Begin

Begin

Select ename into v\_ename from emp where empno=v\_empno1;

Dbms\_output.put\_line('My First Employee'||' '||v\_ename);

exception

when no\_data\_found then

dbms\_output.put\_line('U R Employee Does Not Exists With EMPNO'||' '||v\_empno1);

end;

begin

select ename into v\_ename from emp where empno=v\_empno2;

dbms\_output.put\_line('My Second Employee'||' '||v\_ename);

exception

when no\_data\_found then

dbms\_output.put\_line('U R Employee Does Not Exists With EMPNO'||' '||v\_empno2);

end;

end;

**o/p:** Enter value for empno1: **7902**

Enter value for empno2: **7566**

My First Employee FORD

My Second Employee JONES

**NOTE:**

In pl/sql exception propagation we must use nested blocks.

**Exception Propagation:**

In pl/sql exception also raised in declare section, execution section, executable section. When exception are raised in executable those exception raised either inner block (or) outer block where as when exception are raised in declare section (or) in exception section those exception are must be handle using outer block only to use must handle.

**1. Exception raised in executable section:**

Method-1: handled using inner block

**Ex:** declare

A exception;

Begin

Raise A;

Exception

When A then

Dbms\_output.put\_line ('handled inner block');

End;

Method-2: handled using outer block

**Ex:** declare

A exception;

Begin

Begin

Raise A;

End;

Exception

When A then

Dbms\_output.put\_line('handled outer block');

End;

**2. Exception raised in exception section:** (Handled by using outer block)

**Ex:** declare

k1 exception;

k2 exception;

begin

begin

raise k1;

exception

when k1 then

dbms\_output.put\_line('k1 handled');

raise k2;

end;

exception

when k2 then

dbms\_output.put\_line('k2 handled using outer blocks only');

end;

**o/p:** k1 handled

k2 handled using outer blocks only

**3. Exception raised in declare section:**

**Ex:** begin

declare

A number(2):=99999;

begin

dbms\_output.put\_line(A);

end;

exception

when others then

dbms\_output.put\_line('Not To Store More Than Datatype Size');

end;

**o/p:** Not To Store More Than Datatype Size

**Local procedure, local function:**

Local subprograms are named pl/sql block. Which is used to solve particular task. Local subprogram are not create (or) replace. This subprogram are not store permanently. This local subprogram are used in anonymous block, stored procedure. Local program must defined in bottom of the declare section of the pl/sql and call this subprogram in immediately executable section.

**Syntax:** declare

--🡪 variable declaration, constant declarations

--🡪 type declarations

--🡪cursor declarations

--🡪 procedure procedurename(formal parameter) is/as

----------------------------------------------------

----------------------------------------------------

Begin

---------------------------------------------------

[exception]

End[procedurename];

--🡪function functionname(formal parameter)returning datatype is/as

---------------------------------------

Begin

Return expression;

End[functionname];

Begin

--🡪 procedurename(actual parameter);

--🡪 variable:=functionname(actual parameter);

End;

**Ex:** declare

Procedure p1 is

Begin

Dbms\_output.put\_line('Actual Parameter');

end p1;

begin

p1;

end;

**o/p:** Actual Parameter

**Ex:** create or replace procedure p2 is

procedure p1 is

begin

dbms\_output.put\_line('Actual Parameter');

end p1;

begin

p1;

end;

Sql> exec p2;

**o/p:** Actual Parameter

**DYNAMIC SQL**

Oracle **7.1** introduced dynamic sql. In dynamic sql statements are executed dynamically. Generally in pl/sql we are not allowed to use DCL,DDL commands. If we want to use these statement then we must use dynamic sql constructs within pl/sql block. In dynamic sql we must specify sql statements within single quotes [‘ ‘] and also execute these statements within pl/sql block through **execute immediate** clause must be used in executable section of the pl/sql block.

**Syntax:** begin

Execute immediate ‘sql statement’;

End;

**Example:** begin

execute immediate 'create table test1(sno number(10))';

end;

**Q) Write pl/sql program to create a Role?**

Sql> begin

execute immediate 'create role AB'; [ role create only administrator]

end;

**o/p: ERROR:** ORA-01031: insufficient privileges

**Solution:** sql> conn sys as sysdba Enter password: SYS

Sql> begin

execute immediate 'create role AB';

end;

**O/P:** PL/SQL procedure successfully completed.

**Passing values into dynamic sql statement:**

In dynamic sql we are not allowed to pass actual data directly sql statement. In these case we must use place holders in place of actual data and also pass actual data into place holders through ‘USING’ clause. Place holder are represent “**:”.** Place holder used in either insert, update, delete (or) in sql statement.

**Q) Write dynamic sql program to immediate record into dept table?**

Sql> declare

v\_deptno number(10):=&deptno;

v\_dname varchar2(15):='&dname';

v\_loc varchar2(15):='&loc';

begin

execute immediate 'insert into dept values(:1,:2,:3)' using v\_deptno, v\_dname,v\_loc;

end;

**O/P:** Enter value for deptno**: 60**

Enter value for dname**: narasimha**

Enter value for loc**: y s r**

Sql> select \* from dept;

**Retrieve data from dynamic sql statement:**

Through **into** clause we are retrieving data from dynamic sql statement.

**Q) Write dynamic sql program to display maximum salary from emp table?**

Sql> declare

A number(10);

begin

execute immediate 'select max(sal) from emp' into A;

dbms\_output.put\_line(A);

end;

**O/P: A**

5000

**NOTE:**

When ever we are using ‘USING’, ‘INTO’ clauses in a single dynamic sql statement. There always ‘INTO’ clause processed ‘USING’ clause.

**Q) Write dynamic sql program to retrieve all employee names on emp table on storing into index by table and also display content from table?**

Sql> declare

type t1 is table of varchar2(15)

index by binary\_integer;

v\_t t1;

begin

execute immediate 'select ename from emp' **bulk collect** into v\_t;

for i in v\_t.first..v\_t.last

loop

dbms\_output.put\_line(v\_t(i));

end loop;

end;

**O/P:** SMITH

-----------

MILLER

**Q) Write dynamic sql program for passing deptno=10 retrieve dname,loc from dept table?**

Sql> declare

v\_dname varchar2(15);

v\_loc varchar2(15);

begin

execute immediate 'select dname,loc from dept where deptno=:1' into v\_dname,v\_loc using 10;

dbms\_output.put\_line(v\_dname||' '||v\_loc);

end;

**O/P:** ACCOUNTING NEW YORK

Chapter-9

**PRE-DEFINED PACKAGE**

If we want communicate oracle database procedure to the OS process then we are using dbms\_pipe package. Generally this package also used debugging oracle message in pl/sql debugger. If we want to communicate b/w two session for a single instance then we must use dbms\_pipe package. Generally this package used in inter process communication application. This package is used to message passing in b/w two session for single instance through pipes. This pipes are available in SGA this package having following methods.

**1. DBMS\_PIPE PACKAGE:**

Dbms\_pipe package used to send and receiving messages in b/w two sessions for a single instance in oracle. If we want to develop inter process communicate application then we must use dbms\_pipe package. This package having following procedures and function these are

1. pack\_message()

Procedures

2. unpack\_message()

3. send\_message()

Functions

4. receive\_message()

**SGA**

|  |
| --- |
| -------------------- pipe --------------------- |

Session-1 session-2

**-------------------------------------------------------------------------------------------------------------------------------------------------**

**-----------------------------------------------------------------------------------------------------------------------------**

Pack\_message() unpack\_message()

Send\_message(pipename) receive\_message(pipename)

We are collecting the data using pack\_message() procedure and send\_message data through the send\_message function, send\_message receive\_message always accepts pipe name and return **status code.**

**Syntax:** dbms\_pipe.pack\_message(data);

Statuscodevarname:=dbms\_pipe.send\_message(‘pipename’);

Dbms\_pipe.unpack\_message(varname);

Statuscodevarname:=dbms\_pipe.receive\_message(‘pipename’);

This package is available in **sys** user only before we are work with this package then sys clause must given either execute (or) all privileges.

**Syntax:** grant execute on dbms\_pipe to username;

**(or)**

Grant all on dbms\_pipe to username;

Sql> conn sys as sysdba

Enter password: SYS

Sql> grant all on dbms\_pipe to scott;

Sql> conn scott/tiger;

Sql> create or replace package pk is

procedure p1(p\_text varchar2);

procedure p2;

end;

Sql>create or replace package body pk is

procedure p1(p\_text varchar2) is

v\_status number(10);

begin

dbms\_pipe.pack\_message(p\_text);

v\_status:=dbms\_pipe.send\_message('pipe');

if v\_status<>0 then

dbms\_output.put\_line('Any Network Error');

end if;

end p1;

procedure p2 is

v\_status number(10);

v\_text varchar2(10);

begin

v\_status:=dbms\_pipe.receive\_message('pipe');

if v\_status<>0 then

dbms\_output.put\_line('Any Network Error');

end if;

dbms\_pipe.unpack\_message(v\_text);

dbms\_output.put\_line(v\_text);

end p2;

end;

**Execution:**

Session-1: sql> exec pk.p1(‘hi’);

Session-2: sql> set serveroutput on;

Sql> exec pk.p2;

Sql> create or replace procedure p2(v\_text out varchar2,p\_pipe in varchar2) is

v\_status number(10);

begin

v\_status:=dbms\_pipe.receive\_message(v\_pipe);

if v\_status<>0 then

dbms\_output.put\_line('Any Network Error');

end if;

dbms\_pipe.unpack\_message(v\_text);

end;

Sql> conn scott/tiger;

Sql> exec p1(‘hi’);

Sql> conn simha/simha;

Sql> exec p2(:G);

**Hi**

**2. DBMS\_FLASHBACK PACKAGE:**

Oracle 9i introduced flashback queries for retrieving accidental data after commit transaction also flashback queries allows content of the table can be query with reference to a specific point of time we can also having handle flashback query using ‘DBMS\_FLASHBACK PACKAGE’. Using **enable\_at\_time** procedure we can retrieve accidental data after commit also.

**Syntax:** dbms\_flashback.enable\_at\_time(time);

**Ex:** Sql> create table test3(col date);

Sql> insert into test3 values(sysdate);

Sql> commit;

Sql> delete from emp1 where rownum<=5;

Sql> commit;

Sql> declare

v\_col date;

begin

select col into v\_col from test3;

dbms\_flashback.enable\_at\_time(col);

end;

Sql> select \* from emp1;

**3. DBMS\_JOB PACKAGE:**

In oracle can also execute procedure (or) DML transactions in a specific interval of a time through dbms\_job package. This package is used by database administrator only. This package having predefined procedure through that procedure we are controlling job Q. If we want to submit job into database then we want use submit procedure from dbms\_job package.

**Syntax:** dbms\_job.submit(jobvarname,’DML (or) Procedurename;’,particulartime, ’interval’);

After submitting job we must use ‘commit’.

**Ex:** sql> create table test(col date);

Sql> declare

v\_job number;

begin

dbms\_job.submit(v\_job,'insert into test values(sysdate);', to\_date('25-JAN-2015 10:23:00','DD-MON-YYYY HH24:MI:SS'),null);

commit;

end;

**O/P:** Sql> select \* from test;

**COL**

25-JAN-15

Chapter-10

**MUTATING ERROR**

When a row level trigger based on table then trigger body cannot read data from same table and also we cannot perform DML operation on the same table. If we are trying this oracle server returns **“ERROR: ORA-4091: Table Is Mutating”** this error is called “MUTATING ERROR”. This error is runtime. Statement level triggers does not return mutating error. Because in all database in statement level triggers DML transaction values are automatically committed in database. If we are try to read committed data using trigger body then database server does not returns an errors. Where as in row level trigger DML’s transaction values are not committed automatically. If we are try to read without committing data using trigger body always database servers returns ‘Mutating Error’.

**Ex:** varchar2(10) **table-1**  number(10) varchar2(10) **table-2** date

|  |  |
| --- | --- |
| **Col1** | **Col2** |
| 40 | 20-JAN-2015 |

|  |  |
| --- | --- |
| **Col1** | **Col2** |
| SIMHA | 40 |

Sql> create table test1(col varchar2(10),col2 number(10));

Sql> create table test2(col number(10),col2 date);

Sql> create or replace trigger tr1

after insert on test1

for each row

declare

A number(10);

begin

select col2 into A from test1 where col2=:new.col2;

insert into test2 values(A,sysdate);

end;

Sql> insert into test1 values('SIMHA',50);

**ERROR:** ORA-04091: table SCOTT.TEST1 is mutating, trigger/function may not see it

Wherever we are using statement level triggers DML transaction values are automatically committed in database. We can also read committed data using trigger body without any problems that’s way statement level trigger does not return mutating error. Where as in **row level trigger** DML transaction are not committed automatically. When we are try to read not committed data using trigger body then oracle server returns an mutating error. To avoiding a mutating error oracle provided following steps through package global variable.

Step-1: create an package global variable.

Step-2: develop an after row level trigger.

Step-3: develop an after statement level trigger.

**Avoiding:**

Sql> create or replace trigger tr1

after insert on test1

for each row

declare

A number(10);

begin

select col2 into A from test1 where col2=:new.col2;

insert into test2 values(A,sysdate);

end;

**Solution:** sql> create or replace package pk is

A number(10);

end;

Sql> create or replace trigger t1

after insert on test1

for each row

begin

pk.A:=:new.col2;

end;

Sql> create or replace trigger t2

after insert on test1

begin

insert into test2 values(pk.A,sysdate);

end;

Sql> insert into test1 values('SIMHA',50);

Sql> select \* from test1;

**COL** **COL2**

SIMHA 50

Sql> select \* from test2;

**COL** **COL2**

50 25-JAN-15

**NOTE:**

In oracle we can also avoid mutating error using oracle 11g compound trigger.

**COMPOUND TRIGGER:**

Oracle 11g introduced compound trigger. Compound trigger allows different blocks with in a blocks with in a trigger to the execute at different timing points. Compound trigger also a global declaration section of the same package.

Sql> create or replace trigger triggername

For insert/update/delete on tablename

Compound trigger

[global declaration section]

----🡪 declaring,variables,contents;

----🡪 type declarations

----🡪 procedure procedurename(formal parameters) is

Begin

--------------------

End[procedurename];

----🡪 function functionname(formal parameters);

Return datatype is

Begin

--------------------

Return expression;

End[functionname];

Before statement is

Begin

--------------------------------

End[before statement];

Before each row is

Begin

----------------------------

End[before each row];

After each row is

Begin

------------------------------

End[after each row];

After statement is

Begin

------------------------------

End[after statement];

End;

**Avoiding mutating error using compound trigger (oracle 11g):**

Sql> create or replace trigger t1

for insert on test

compound trigger

before statement is

begin

dbms\_output.put\_line('Before Statement Level');

end before statement;

after statement is

begin

dbms\_output.put\_line('After Statement Level');

end after statement;

before each row is

begin

dbms\_output.put\_line('Before Row Level');

end before each row;

after each row is

begin

dbms\_output.put\_line('After Row Level');

end after each row;

end;

**O/P:** sql> insert into test values(10);

Before statement level

Before row level

After row level

After statement level

**Q) Write pl/sql compound trigger program on emp table when ever user deleting date display remaining number of records numbers in bottom of the delete statement?**

Sql> create or replace trigger t1

For delete on emp

Compound trigger

V\_count number(10);

After statement is

Begin

Select count(\*) into v\_count from emp;

Dbms\_output.put\_line(v\_count);

End after statement;

End;

**O/P:** Sql> delete from emp where empno=7566;

**13**

**Member Function,Member Procedure:**

Pl/sql also supports objects technology-member subprogram having a name and also used to solve particular task. Member subprograms are only defined in object type.

**Object type:**

Object type is an user defined type which contains collection of different datatypes and also member procedures,member functions. These member subprograms are operate this member data object having two parts.

1. type specification

2. type body

In object specification we are declaring member data. Member subprogram where as in object body we are implementation member programs

**Type specification:**

**Syntax:** create or replace type typename as object

(attributename1 datatype(size), attributename2 datatype(size),…………..

Member procedure procedurename,member function functionname return datatype);

**Type body:**

**Syntax:** create or replace type body typename

As member procedure procedurename is

Begin

----------------------------------------

End[procedurename];

Member function functionname

Return datatype is

Begin

----------------------------------

Return expression;

End[functionname];

End;

**Instantiating an object:**

Once we are creating an object type then we must instantiate this object in declare section of the pl/sql block. After instancing an object we must assign values to the attributes using constructor. Here constructor name as same as object typename,these values must be assign in executable section of the pl/sql block.

**Syntax:** varname objecttypename;

varname:=objectname(values1,values2,……………………);

Sql> create or replace type student as object(sno number(10),sname varchar2(15));

/

Sql> declare

A student;

begin

A:=student(101,'NARASIMHA');

dbms\_output.put\_line('My Student Id'||' '||A.sno);

dbms\_output.put\_line('My Student Name Is'||' '||A.sname);

end;

**O/P:** My Student Id **101**

My Student Name Is **NARASIMHA**

Sql> create or replace type circle as object(R number,member procedure p1,member function f1 return number);

/

Sql> create or replace type body circle as

member procedure p1 is

begin

null;

end p1;

member function f1

return number is

begin

return 2\*3.14\*R;

end f1;

end;

**Instantiating an object:**

Sql> declare

D circle;

begin

D:=circle(8);

dbms\_output.put\_line(D.f1);

end;

**O/P:** 50.24

**Where current of ,for update clause used in cursors** (or) **delete,update statement are used in cursor** (or) **cursor locking mechanism:**

**Q) Write pl/sql cursor program modify salaries of the employee based on following conditions using emp1 table?**

**1) if job=’CLERK’ then increment sal--------🡪 100**

**2) if job=’SALESMAN’ then decrement sal -----------🡪 200**

Sql> declare

cursor c1 is select \* from emp;

i emp%rowtype;

begin

open c1;

loop

fetch c1 into i;

exit when c1%notfound;

if i.job='CLERK' then

update emp set sal=i.sal+100 where empno=i.empno;

elsif i.job='SALESMAN' then

update emp set sal=i.sal-200 where empno=i.empno;

end if;

end loop;

close c1;

end;

Sql> select \* from emp;

Generally in all database systems whenever we are using update,delete statement then database server automatically provides default locking mechanism. If we want establishes to locks before update,before delete statement then only we are using explicit locking mechanism through cursors.

**FOR UPDATE:**

If we want to establishes locks then we must use **for update** clause in cursor select statement. When ever we are using **for update** clause also database servers does not establishes locks when we are defining the cursors. i.e whenever we are opening the cursor then only database server establishes exclusive locks.

**Syntax:** cursor cursorname is select \* from tablename where condition for update;

**WHERE CURRENT OF:**

WHERE CURRENT OF clause is used to identifying by a record uniquely. Because this clause internally **rowid**. Where current of clause is used in update,delete statement only. Whenever we are using where current of clause then we must use for update clause in cursor select statement.

**Syntax:** update tablename set colname=newvalue where current of cursorname;

Delete from tablename where current of cursorname;

**NOTE:**

After processing we must **release the locks** using **commit** with in pl/sql block. Where current of clause is used to update,delete most recently fetched row from the cursor.

**Q) Write pl/sql cursor program modify salaries of the CLERK’s from emp table using cursor locking mechanism?**

Sql> declare

cursor c1 is select \* from emp for update;

i emp%rowtype;

begin

open c1;

loop

fetch c1 into i;

exit when c1%notfound;

if i.job='CLERK' then

update emp set sal=i.sal+100 where current of c1;

end if;

end loop;

commit;

close c1;

end;

Sql> select \* from emp;

Chapter-11

**ORACLE 11G FEATURES**

🡪 Oracle 11g introduced **READ-ONLY** tables through alter command, in these table we cannot perform DML operations.

**Syntax:** Alter table tablename read only;

Alter table tablename read write;

🡪 Oracle 11g introduced simple-integer datatype in pl/sql. This datatype perform is very high compare to simple\_integer datatype.

**Syntax:** variablename simple\_integer:=value;

🡪 simple\_integer datatype internally having not null clause that’s way we must assign the value at the time of variable declaration.

**Ex:** declare

A simple\_integer:=50;

Begin

Dbms\_output.put\_line(A);

End;

**O/P: 50**

🡪 Oracle 11g introduced virtual column to store expressions directly database.

🡪 Oracle 11g introduced continues statement in pl/sql loops, it is also same as ‘C’ language continue statement.

**Syntax:** continue;

Sql> begin

For I in 1..10

Loop

If i=5 then

Continue;

End if;

Dbms\_output.put\_line(i);

End loop;

End;

🡪 Oracle 11g introduced **pivot()** function which convert rows as columns pivot() function always display aggregate[group] values into tabular format.

Sql> select job,sum(decode(deptno,10,sal)) “Deptno 10”,

Sum(decode(deptno,20,sal)) “Deptno 20”,

Sum(decode(deptno,30,sal)) “Deptno 30”

From emp group by job;

**O/P: JOB Deptno 10 Deptno 20 Deptno 30**

CLERK 1700 2300 1150

SALESMAN 4800

PRESIDENT 5000

MANAGER 2650 2975 2850

ANALYST 6000

**Syntax:** select \* from

(select col1,col2,……….from tablename)

Pivot(aggrigatefunationname(colname) for colname

in(value1,value2,value3,……………………………));

Sql> select \* from (select job,deptno,sal from emp) pivot(sum(sal) for deptno in (10 as deptno 10,20 as deptno 20,30 as deptno 30));

🡪 Oracle 11g introduced compound trigger follows clause in triggers.

🡪 Oracle 11g introduced enable,diable keyword in trigger specification.

🡪 Oracle 11g introduced variable assignment concepts when we are using sequences pl/sql block. i.e Here we are not allowed to use dual table.

🡪 Oracle 11g introduced named,mixed notations when a subprogram executed using select statement.