

ORACLE

PL\SQL

30 Aug
Jul

MONDAY

JAN 12

2

002-364 WK-01

PL/SQL

PL/SQL is a procedural language extension for SQL.

It is the combination of procedural, data manipulation language.

Oracle 6.0 introduced PL/SQL.

To view version of pl/sql :

```
sql> Select * from v$version;
```

Basically pl/sql is a block structured :

Declare [optional]

variable declarations,
cursors,
undefined exceptions.

Begin [mandatory]

DML, TCL

select --- into ---
Conditional,
control statements ;

Exception [optional]

Handling
exceptions

End ; [mandatory]

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

TUESDAY

JAN 12

3

003-363 WK-01

There are two types of blocks supported by pl/sql :

1. Anonymous blocks
2. Named blocks

1. Anonymous Blocks

These blocks does not have a name & also not stored in database & also we are allowed to call these blocks in another blocks or in client application.

ex. Declare

— —

— .. —

Begin

— .. —

— .. —

end ;

2. Named Blocks

These blocks having a name & also automatically stored in database.

These blocks used by all types of programmers in all applications, these are procedures,

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

WEDNESDAY

JAN-19

4

004-362 WK-01

functions, triggers, packages, etc.

PL/SQL Datatypes, Variables

I. It supports all sql datatypes (scalar datatypes)

+

boolean datatypes

II. lobs (clob, blob, bfile)

III. composite datatypes.

IV. ref objects

V. non-pl/sql variables (or) bind variables
(or) host variables.

Variable

Variable is used to store a single value into memory location.

Syntax for declaring a variable :

Variablename datatype (size);

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

THURSDAY

JAN 12

5

005-361 WK-01

Generally we are declaring variables in declare section of the pl/sql block.

ex. declare

a number (10);
b varchar2 (10);

storing a value into variable :-

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

Syntax : `variablename := value;`

ex. : `a := 60 ;`

Display message (or) variable value

`dbms_output.put_line ('Message');`

Packagename Procedurename

`dbms_output.put_line (variablename);`

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

FRIDAY

JAN 12

6

ex. sql > set serveroutput on;

sql > begin

dbms_output.put_line ('Kadam');

end ;

/

(Output) Kadam

ex. declare

a number (10);

begin

a := 60 ;

dbms_output.put_line (a);

end ;

/

(Output) 60

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

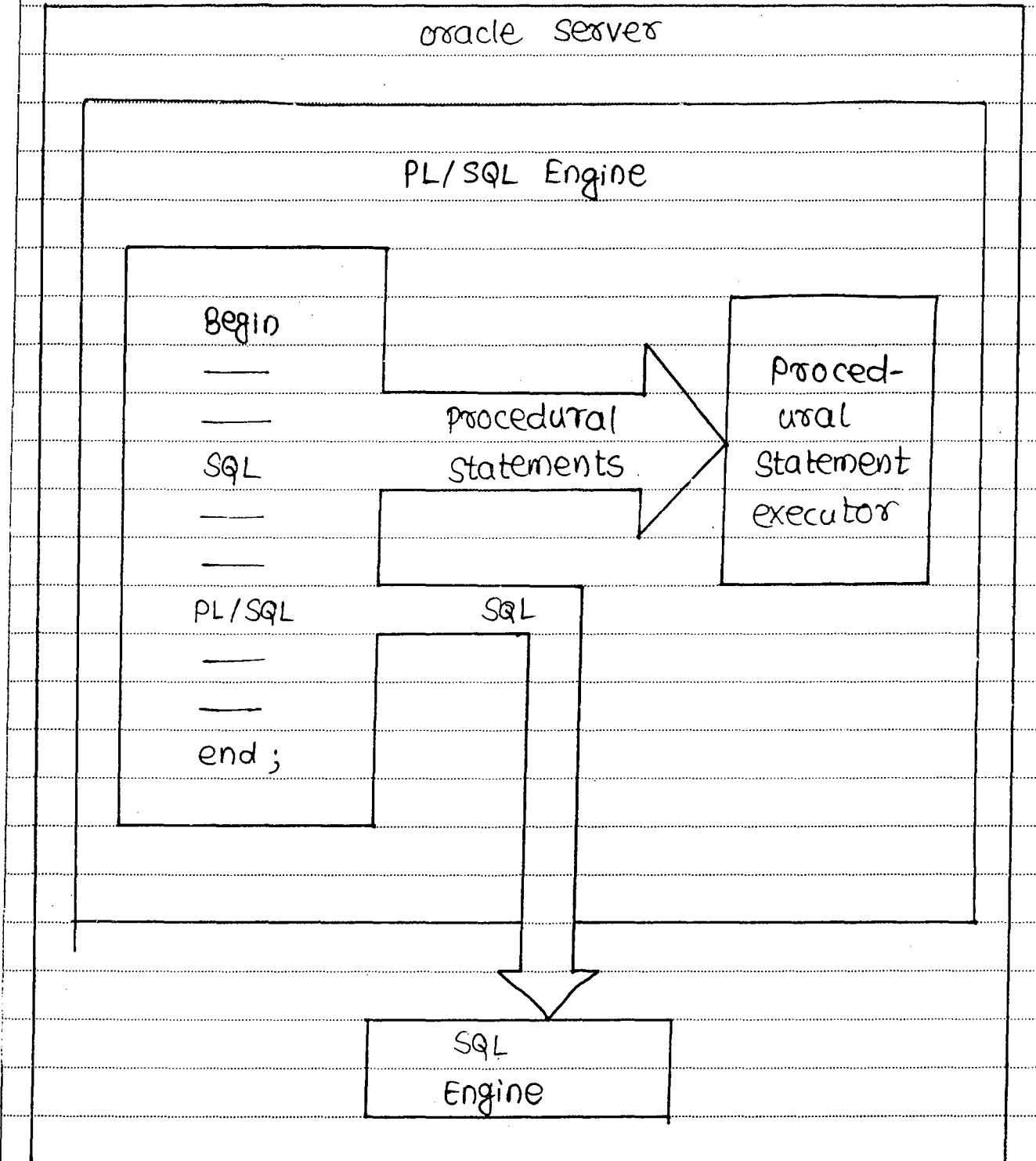
SATURDAY

JAN 12

7

007-359 WK-01

Fig. PL/SQL Architecture



January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Select --- into --- clause

This clause is used to data from table & storing into pl/sql variables.

Select --- into --- clause always returns single record or value at a time.

Syntax: Select col1, col2, ... into var1, var2,
... from tablename where condition;

This clause is used in executable section of pl/sql program.

? Write a pl/sql program for user entered empno ; display name of the employee & salary from emp table.

```
soln declare
v_ename varchar2(10);
v_sal number(10);
begin
select ename, sal into v_ename, v_sal
from emp where empno = &no;
dbms_output.put_line (v_ename || ' ' || v_sal);
end;
/
```

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

MONDAY

JAN 12

9

(output) Enter value for no : 7902

009-357 WK-02

FORD 3600

- 31 [Aug] Jul

Note

Whenever we are using not null, constant clauses in variable declaration we must assign the value at the time of declaration only.

ex. declare

a number(10) not null := 60 ;

b constant number(10) := 5 ;

begin

dbms_output.put_line (a);

dbms_output.put_line (b);

end ;

/

(output)

60

5

Note

We can also use default keyword in place of assignment operator, in declare section of the pl/sql block.

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

JAN 12

10

010-356 WK-02

```

ex. declare
      a number (10) default 30;
begin
  dbms_output.put_line (a);
end;
/

```

(output) 30

FEB

? Write a pl/sql program to display maximum salary from emp table.

Solⁿ

```

declare
  v-sal number (10);
begin
  select max (sal) into v-sal from
  emp;
  dbms_output.put_line (v-sal);
end;
/

```

(output) 6100

Note

We are not allow to use group functions, decode conversion functions in pl/sql expressions.

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Fr	Sa	Su	Mo	Tu	We	♦	♦	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

WEDNESDAY

JAN 12

11

But we are using number, character, date functions in pl/sql expressions.

011-355 WK-02

ex. declare

a number (10);

b number (10);

c number (10);

begin

a := 40;

b := 30;

c := greatest (a, b);

dbms_output.put_line (c);

end;

/

(Output) 40

Variable Attributes

Variable attributes are used in place of datatypes in variable declaration or procedure parameter declaration in pl/sql block.

There are two types of variable attributes supported by pl/sql.

1. Column level attributes

2. Row level attributes

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

12

012-354 WK-02

I. Column Level Attributes

In this method we are defining attributes for individual columns.

This attribute is represented using %type.

Syntax: VariableName tablename.Columnname%type;

Whenever we are using this attribute PL/SQL runtime engine allocates memory for the variables corresponding to the columns in database tables.

```
ex. declare
      v_ename    emp.ename %type ;
      v_sal      emp.sal %type ;
      begin
      select ename, sal into v_ename, v_sal
      from emp
      where empno = &no ;
      dbms_output.put_line ('v_ename || ''||'
                           v_sal );
      end ;
      /
```

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

FRIDAY

JAN '12

13

2. Row Level Attribute

013-353 WK-02

In this method a single variable represents all different datatypes in entire row in a table that's why this variable is also called as record type variable.

This variable is represented using %rowtype.
It is also same as structure in C language.

Syntax : Variablename tablename %.rowtype;

ex. Declare

i emp%.rowtype;

i.empno

empno	ename	job	MGR	hiredate	Comm	sal	deptno
7309	SMITH	CLERK	

i

ex. declare

i emp%.rowtype;

begin

select ename, sal, hiredate, deptno

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

JAN 12

14

014-352 WK-02

into i.ename, i.sal, i.hiredate,
 i.deptno from emp

where empno = &no;

dbms_output.put_line (i.ename || ' ' || i.sal ||
 ' ' || i.hiredate || ' ' || i.deptno);

end ;

/

FEB

ex.

declare

i emp%rowtype;

begin

select * into i from emp where

empno = &no;

dbms_output.put_line (i.ename || ' ' || i.sal ||
 ' ' || i.hiredate || ' ' || i.deptno);

end;

/

MAR

APR

Conditional Statements (if)

I. if

II. if else

III. elsif

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Sa	♦	♦						
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

SUNDAY

JAN 12

15

015-351 WK-02

I. if Syntax:

```
if Condition then  
    Statements;  
end if;
```

II. if else syntax:

```
if Condition then  
    Statements;  
else  
    Statements;  
end if;
```

III. elseif To check more number of conditions we are using elseif.

```
Syntax : if condition 1 then  
    Statements;  
elseif Condition 2 then  
    Statements;  
elseif Condition 3 then  
    Statements;  
-----  
else  
    Statements;  
end if;
```

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

MONDAY

- 01 Aug -

JAN 12

16

016-350 WK-03

ex. declare

v_deptno number (10);

begin

select deptno into v_deptno from emp

where deptno = & no;

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 v_deptno = 40

dbms_output.put_line ('others');

end if;

end;

/

(output) enter value for no : 40

others

enter value for no: 90

error

Note

If a pl/sql block contains select into clause & also if requested data not available in a table Oracle server returns an error ora-1403 : no data found

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

TUESDAY

JAN 12

17

Note

017-349 WK-03

In the above program if we are using emp table in place of dept table & also if requesting data oracle server returns an error because whenever select --- into --- clause try to return more than one value oracle server returns an error

ora-1422 : exact fetch returns more than requested number of rows.

Note

When a pl/sql block contains dml statements & also if requested data not available in a table oracle server does not return any error.

ex. sql > begin
 delete from emp where ename='java';
 end;
 /

(Output) PL/SQL procedure successfully completed.

To handle these types of blocks we are using implicit cursor attributes.

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

WEDNESDAY

JAN-12

18

018-348 WK-03

Control Statements (loops)

There are three types of loops supported by pl/sql.

1. Simple loop
2. While loop
3. For loop

1. Simple loop

Here body of the statement is executed repeatedly.

```
Syntax : loop
          statements;
      end loop;
```

This loop is also called as infinite loop.

```
ex. begin
      loop
          dbms_output.put_line ('Welcome');
      end loop;
  end;
```

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Fr	Sa	Su	Mo	Tu	We	Fr	Sa	Su
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

THURSDAY

JAN 19

19

019-347 WK-03

To exit from infinite loop we are using following two methods

(i) Method 1

Syntax: exit when true condition;

ex. declare

n number (10) := 1 ;

begin

loop

dbms_output.put_line (n);

exit when n >= 10 ;

n := n + 1 ;

end loop ;

end ;

/

(ii) Method 2 (using if)

Syntax: if condition then

exit ;

end if ;

ex. declare

n number (10) := 1 ;

begin

loop

dbms_output.put_line (n);

if n >= 10 then

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

FRIDAY

JAN 12

20

020-346 WK-03

```

exit ;
end if ;
n := n + 1 ;
end loop ;
end ;
/

```

2. While loop

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

Syntax: While (condition)

```

loop
statements ;
end loop ;

```

ex. declare

n number (10) := 1 ;

begin

while (n >= 10)

loop

dbms_output.put_line (n);

n := n + 1 ;

end loop ;

end ;

/

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

SATURDAY

JAN 12

21

3. For loop

021-345 WK-03

Syntax: for indexvariablename in
lowerbound .. upperbound

loop

statements ;

end loop ;

ex. declare

n number (10);

begin

for n in 1..10

loop

dbms_output.put_line (n);

end loop ;

end ;

/

ex. begin

for n in 1..10

loop

dbms_output.put_line (n);

end loop ;

end ;

/

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

JAN 12

22

022-344 WK-03

```

ex. begin
    for n in reverse 1..10
        loop
            dbms_output.put_line(n);
        end loop;
    end;
/

```

Bind Variable -

These variables are session variables, these variables are created at host environment that's why these variables are also called as host variables.

These variables are non-pl/sql, but these variables are used in pl/sql to execute procedures having out parameters.

Step 1 Creating a bind variable.

Syntax: sql> variableName datatype;

Step 2 Using bind variable.

Syntax: :variableName;

Step 3 Display value from bind variable.

Syntax: sql> print variableName;

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Fr	Sa	Su	Mo	Tu	We	Fr	Sa	Su
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

MONDAY

JAN '12

23

023-343 WK-C4

ex. sql> variable g number;

```
sql> declare  
      a number(10) := 500;  
begin  
  :g := a/2;  
end;
```

/

Sql> print g;

(output) G

250

- 02 Aug -

CURSOR

Cursor is a private SQL memory area which is used to process multiple records & also this is a record by record process.

There are two types of static cursors supported by oracle.

1. Implicit Cursor
2. Explicit Cursor

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

JAN 22

24

024-342 WK-04

1. Implicit Cursors

SQL statements returns a single record is called implicit cursors.

Implicit cursors are simple pl/sql programs which contains select into clause or dml statements.

When pl/sql block contains select... into clause Oracle server creates a memory area and returns a single record.

This memory area is also called as sql area.

When a pl/sql block contains dml statements & also sql area returns multiple records.

In this case these all records are processed at a time this is also called as implicit cursor memory area or context area.

ex. declare

```
v_ename varchar2(10);
```

```
v_sal number(10);
```

begin

```
Select ename, sal into v_ename, v_sal
```

```
from emp where empno = &no;
```

```
dbms_output.put_line (v_ename||'/'||v_sal);
```

end;

/

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Fr	Sa	Su	Mo	Tu	We	♦	♦	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

WEDNESDAY

JAN 12

25

025-341 WK-04

(Output) Enter value for no : 7902

FORD 3000

2. Explicit Cursors

for SQL statement returns a multiple records is called explicit cursors.

Explicit cursor memory area is also called as active set area.

In this explicit cursors we are storing multiple records & also these records are controlled by database developers explicitly but database developers can not control implicit cursor memory area record.

Explicit Cursor Lifecycle

- i. Declare
- ii. Open
- iii. Fetch
- iv. Close

i. Declare In declare section of the pl/sql block we are defining cursors using following syntax:

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

JAN 12

cursor cursorname is select *
from tablename where condition;

26

026-340 WK-04

ii. Open Whenever we are opening the cursor then only oracle server fetch data from table into cursor memory area, because whenever we are opening the cursor then only cursor select statements are executed.

Syntax: open cursorname;

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

Note

When we are opening the cursor implicitly cursor pointer points to first record in the cursor.

iii. Fetch (Fetching From Cursor) Using fetch statement we are fetching data from cursor into pl/sql variables.

Syntax: Fetch cursorname into variable1,
variable2, ... ;

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Fr	Sa	Su	Mo	Tu	We	Fr	Sa	Su
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

FRIDAY

JAN 12

27

027-339 WK-04

iv. Close When we are closing the cursor all resources allocated from cursor memory area automatically released.

Syntax: close cursorname ;

ex. declare

```
cursor c1 is select ename, sal from emp;
v-ename varchar2(10);
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);
close c1 ;
end ;
/
```

(Output) SMITH 2000
 ALLEN 900

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

JAN 12

Explicit Cursor Attributes

28

028-338 WK-04

Every explicit cursor having following four attributes :

- 1) %notfound
- 2) %found
- 3) %isopen
- 4) %rowcount

All these cursor attributes using along with cursorname only.

Syntax: cursorname.%attributename

Except %rowcount all other cursor attribute records boolean value returns either true or false whereas %rowcount returns number datatype.

1) %notfound This attribute returns true when cursor does not have data after fetching records from cursor.

? Display all the records from emp table using explicit cursor lifecycle ?

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Fr	Sa	Su	Mo	Tu	We	♦	♦	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

SUNDAY

JAN-12

29

029-337 WK-04

SQ1D declare
cursor c1 is select ename, sal
from emp;
v-ename varchar2(10);
v-sal number(10);
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;
/

-03 Aug-

? Write a pl/sql cursor program to display first five highest salary employees from emp table using %rowcount attribute.

SQ1D declare
cursor c1 is select sal from emp order
by sal desc;
v-sal number(10);
begin

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

MONDAY

JAN 30

30

030-336 WK-05

```

open c1;
loop
fetch c1 into v-sal;
dbms_output.put_line (v-sal);
exit when c1%rowcount=5;
end loop;
close c1;
end;
/

```

? Write a pl/sql cursor program to display even number of rows from emp table using %rowcount attribute.

Sol) declare

```

cursor c1 is select * from emp;
; emp%rowtype;
begin
open c1;
loop
fetch c1 into i;
exit when c1%notfound;
if mod (c1%rowcount,2)=0 then
dbms_output.put_line (i.ename ||' || i.sal);
end if ;
end loop;
close c1;
end;
/

```

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

TUESDAY

JAN 12

31

031-335 WK-05

Whenever we are using cursor automatically pl/sql runtime engine creates four variables.

These variables stores a single value at a time these variables are identified through cursor attribute.

Cx.notfound	C1x.found	C1x.isopen	C1x.rowcount
true	false	true	2

%rowCount

This cursor attribute always returns number datatype i.e. it stores number of record's number fetched from the cursor.

ex: declare

cursor c1 is select * from emp;

 i emp%rowtype;

begin

 open c1;

 fetch c1 into i;

 dbms_output.put_line (i.ename || '|| i.sal);

 fetch c1 into i;

 dbms_output.put_line (i.ename || '|| i.sal);

 dbms_output.put_line ('no. of records
 fetched from cursor is ' || Cx.RowCount);

January	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

WK

ACTION PLAN

FEBRUARY '12

Mon	Tue	Wed	Thu	Fri	Sat	Sun
5			1	2	3	4
6	6	7	8	9	10	11
7	13	14	15	16	17	18
8	20	21	22	23	24	25
9	27	28	29			

FEB

3
A
P
R

卷之三

OBJECTIVES THIS MONTH

A

2012

JANUARY '12	M	T	W	T	F	S	S	M	T	W	T	F	S	S	
									1	2	3	4	5	6	7
9	10	11	12	13	14	15	16	17	18	19	20	21	22		
23	24	25	26	27	28	29	30	31							

WEDNESDAY

PDB 12

1

Close C1;
end;

032-334 WK-05

/

(output) SMITH 1800

ALLEN 700

no. of records fetched from cursor
is 2

ex. 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 > 2000 then
      dbms_output.put_line (i.sal || '||'highsal');
    end if;
  end loop;
  close c1;
end;
```

/

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

THURSDAY

FEB 12

2

033-333 WK-05

(output) SCOTT highsal
 KING highsal
 ADAM highsal
 FORD highsal
 MILLER highsal

Note

These explicit cursors are also used to transfer data from one oracle table to another oracle table.

sql> Create table target (name varchar2(10),
 sal number(10));

sql> declare
 cursor c1 is select * from emp
 where sal > 2000;
 i emp%rowtype;
 begin
 open c1;
 loop
 fetch c1 into i;
 exit when c1%notfound;
 insert into target values (i.empno, i.sal);
 end loop;
 close c1;
 end;
 /

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

FRIDAY

3

Sql> select * from target;

034-332 WK-05

Eliminating Explicit Cursor Lifecycle

Using cursor for loops we are eliminating explicit cursor lifecycle i.e when we are using cursor for loops internally pl/sql runtime engine uses open, fetch, close statements.

Cursor for loop

Syntax: for indexvariable in cursorname
loop
 statements;
end loop;

This is also called as shortcut method of the cursor. This loop is used in executable section of the pl/sql block.

Note

Here cursor for loop index variable internally behaves like a "record type" variable" (i.e %rowtype)

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

SATURDAY

NBS

4

035-331 WK-05

```

ex. declare
cursor c1 is select * from
emp;
begin
for i in c1
loop
dbms_output.put_line (i.ename||' '||i.sal);
end loop;
end;
/

```

Note

We can also eliminate declare section of the cursor using for loop in this case we are using select statements in place of cursorname in cursor for loop.

```

ex: begin
for i in (select * from emp )
loop
dbms_output.put_line (i.ename||' '||i.sal);
end loop;
end;
/

```

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

FEB 12

- 16 Aug -

5

Parametrized Cursors

036-330 WK-05)

we can also

pass parameters to the cursors same like a subprograms in parameters.

These type of cursors are also called as parametrized cursors.

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

Syntax: Open cursorname (actual parameters);

ex. 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.deptno);

end loop ;

close C1 ;

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

MONDAY

FEB 12

6

037-329 WK-06

Note

Whenever we are passing formal parameter to the cursor, procedure, function then we can not specify datatype size in formal parameter declaration.

? Write a pl/sql parametrized cursor program to display employees working as managers or analyst from emp table & also produce this report statically.

```

    begin
        declare
            cursor c1 (p-job varchar2) is select
                * from emp where job = p-job ;
            i emp%rowtype;
        begin
            open c1 ('MANAGER');
            dbms_output.put_line ('Employee working as
                managers');
            loop
                fetch c1 into i ;
                exit when c1%notfound ;
                dbms_output.put_line (i.ename);
            end loop;
            close c1 ;
        end;
    
```

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

FEB 12

7

038-328 WK-06

```
Open C1 ('ANALYST');
dbms_output.put_line ('Employee
working as analyst');
loop
fetch C1 into i;
exit when C1%notfound;
dbms_output.put_line (i.ename);
end loop;
close C1;
end;
/
```

Note

Before we are reopening the cursor
we must close the cursor properly otherwise
oracle server returns an error
cursor already open

Note

If we are try to close the cursor
without opening the cursor oracle server
returns an error

ORA-1001 : invalid cursor

ex. declare

```
cursor C1 (p_deptno number) is
select * from emp where
deptno= p_deptno;
```

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

WEDNESDAY

FEB 12

8

039-327 WK-06

```

begin
for i in c1(10)
loop
dbms_output.put_line ( i.ename || '|| i.deptno);
end loop;
end;
/

```

Note

We can also pass default values using either default or " := " operator.

Syntax : parametername datatype default
actualvalue

Syntax : parametername datatype := actualvalue

ex. declare

```

cursor c1 ( p-deptno number default 20)
is select * from emp where
deptno = p-deptno ;

```

begin

for i in c1()

loop

```
dbms_output.put_line ( i.ename || '|| i.deptno);
```

end loop;

end ;

/

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

FEB 12

9

040-326 WK-06.

? Write a pl/sql program to retrieve different deptno from dept

table using a cursor & also pass these deptno.'s into another parameterized cursor to retrieve employee details for the corresponding deptno.

Soln 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 ('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;

/

- 17 Aug -

? Write a pl/sql cursor program to modify salaries of employees from emp table based

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

FRIDAY

FEB 19

10

041-325 WK-06

on following conditions :

i) If job = 'CLERK' then increment
Sal 100

ii) IF job = 'SALESMAN' then decrement sal 200

iii) IF job = 'ANALYST' then increment sal 100

SOL declare

cursor c1 is select * from emp;

; emp%rowtype;

begin

open c1;

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 ;

elsif i.job = 'ANALYST' then

update emp set sal = i.sal + 100 where

empno = i.empno ;

end if ;

end loop;

close c1 ;

end ;

/

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

FEB 12

11

Where current of, for update clauses.
used in cursors

042-324 WK-06

(or)

Update, Delete statements used in cursors

Generally when we are using update, delete statements automatically locks are established. If you want to establish locks before update, delete statements then we are using cursor locking mechanism in all database systems.

In this case we must specify for update clause in cursor definition.

Syntax: Cursor Cursorname is Select * from tablename where condition for update [of columnname] [Nowait];

If you are specifying for update clause also Oracle server does not establish the lock i.e whenever we are opening the cursor then only oracle server internally uses exclusive locks.

After processing we must release the locks using commit.

where current of this clause is used in update, delete statements used in cursor.

Syntax: update tablename set columnname =

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

SUNDAY

FBB 12

12

043-323 WK-06

newvalue where current of
cursorname ;

Syntax: Delete from tablename where current
of cursorname;

where current of clause uniquely identifying
a record in each process because where
current of clause internally uses rowid.

Note

Whenever we are using where current of
clause we must use for update clause.

Note

where current of clause used to update
or delete latestly fetched row from the
cursor.

ex. declare

cursor C1 is select * from emp

for update;

i emp%rowtype;

begin

open C1;

loop

fetch C1 into i;

exit when C1%notfound;

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

MONDAY

FEB '12

13

044-322 WK-07

```
if i.job = 'CLERK' then
    update emp set sal=i.sal + 100
    where current of c1;
end if;
end loop;
Commit;
close c1;
end;
/
```

? Write a pl/sql cursor program using cursor locking mechanism raise 5% salary who are working under King as manager from emp table.

```
SOLn
declare
    cursor c1 (p-mgr number) is select
        sal 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 * 1.05
        where current of c1;
    end loop;
```

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

TUESDAY

FEB 12

14

045-321 WK-07

Commit ;

end ;

/

Implicit Cursor Attributes

When a pl/sql block contains pure DML statements & also contains select into clause oracle server internally creates an memory area, this memory area is also called as context area or sql area or implicit cursor.

This sql area allocates four variables.

These variables are identified using attributes i.e. implicit cursor having following four attributes

1. sql%notfound
2. sql%found
3. sql%isopen
4. sql%rowcount

Here sql%isopen always returns false whereas sql%notfound, sql%found returns boolean value either true or false & also sql%rowcount returns no. datatype.

ex. begin

delete from emp where ename='abcd';

if sql%found then

dbms_output.put_line ('your record deleted');

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

WEDNESDAY

FEB 15

15

046-320 WK-07

```
end if;  
if sql%notfound then  
dbms_output.put_line ('your record  
does not exist');  
end if;  
end;  
/
```

ex. begin

```
update emp set sal=sal + 100 where  
job = 'CLERK';  
dbms_output.put_line (' affected number  
of clerks are:' || sql%rowcount);  
end;  
/
```

(Output) → affected number of clerks are: 4

- 18 Aug -

Exception

Exception is an error occurred during runtime.

Whenever runtime error occurs use an appropriate exception name in exception handler under exception section.

There are three types of exceptions supported by oracle

1. Predefined exceptions

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

THURSDAY

MONDAY 12

2. Userdefined exceptions
3. Unnamed exceptions

16

047-319 WK-07

1. Predefined Exceptions

Oracle defined 20 predefined exceptions names for regularly occurred errors.

Whenever these errors occur use an appropriate exceptionname in exception handler.

Syntax : when predefinedexceptionname1 then
statements ;

when predefinedexceptionname2 then
statements ;

when Others then
statements ;

Predefined exceptions

- i) no_data_found
- ii) too_many_rows
- iii) zero_divide
- iv) invalid_cursor
- v) cursor_already_open
- vi) invalid_number
- vii) value_error

MAR

APR



March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

FRIDAY

FEB 12

17

i) no-data-found

048-318 WK-07

When a pl/sql block contains select --- into clause & also if requested data not available in a table oracle server returns an error ora-1403: no data found.

To handle this error we are using no_data_found exceptionname.

ex. declare

v_ename varchar2(10);

v_sal number(10);

begin

select ename, sal into v_ename, v_sal
from emp where empno = &no;
dbms_output.put_line (v_ename || ' ' || v_sal);
exception

when no-data-found then

dbms_output.put_line ('employee does not exist');
end;

/

(Output) ⇒ Enter value for no: 7902

FORD 4000

⇒ /

Enter value for no: 9999

employee does not exist.

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

SATURDAY

18-02-12

18

049-317 WK-07

ii) too_many_rows

When a select --- into clause try to return more than one record or more than one value then oracle server returns an error ora-1422 : exact fetch returns more than requested number of rows.

To handle this error we are using too_many_rows exceptionname.

ex: 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 to return more
rows');
end;
```

/

iii) zero_divide

ora-1476: divisor is equal to zero

ex: declare

a number(10);

MAR

APR



March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

BBB-12

19

050-316 WK-07

```
b    number(10);
c    number(10);
begin
  a:=5;
  b:=0;
  c:=a/b;
  dbms_output.put_line(c);
exception
  when zero_divide then
    dbms_output.put_line('b cannot be zero');
  end;
/
```

iv) invalid_CURSOR

Whenever we are performing invalid operations on the cursor oracle server returns an error i.e. if you are try to close the cursor without opening the cursor oracle server returns an error ora-1001 : invalid cursor.

To handle this cursor we are using invalid_cursor exceptionname.

Ex: declare

```
cursor c1 is select * from emp;
  i emp%rowtype;
begin
loop
```

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

MONDAY

FEB 19

20

051-315 WK-08

```

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;
/

```

v> cursor_already-open

When we are try to reopen the cursor without closing the cursor oracle server returns an error ora-06511: cursor already open To handle this error we are using cursor_already-open exceptionname.

ex: declare

cursor c1 is select * from emp;

i emp%rowtype;

begin

open c1;

loop

fetch c1 into i;

3 MAR

APR

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

FEB 12

21

052-314 WK-08

```
exit when c1%notfound;
dbms_output.put_line ('i.enamell'||i.sal);
end loop;
open c1
exception
when cursor_already_open then
dbms_output.put_line ('before reope
we must close the cursor');
end;
/
```

invalid_number, value_error

Whenever we are try to convert string type
to number type oracle server return errors
invalid_number, value error.

v) invalid_number

When a pl/sql block contains sql statements
& also if you are try to convert string
type to number type oracle server returns
an error, ora-1722 : invalid number.

To handle this error we are using invalid_error
exceptionname.

ex: begin

```
insert into emp (empno,sal) values (1,'abc');
```

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

WEDNESDAY

PDB 12

22

053-313 WK-08

exception

when invalid_number then

dbms_output.put_line ('insert proper
data only');

end;

/

vii) value_error

when a pl/sql block contains pl/sql statements
& also if you are try to convert string type to
number type oracle server returns an error,
ORA-6502 : numeric or value error : character
to number conversion error.

To handle this error we are using value_error
exceptionname.

ex: declare

z number(10);

begin

z = '&x' + '&y';

dbms_output.put_line (z);

exception

when value_error then

dbms_output.put_line ('enter proper data
only');

end;

/

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

FEB 12

23

19 Aug

Note

054-312 WK-08

whenever we are try to store large amount of data than the specified datatype size in variable declaration Oracle Server returns an error ora-6502 : numeric or value error : character string buffer too small. To handle this error we are using value_error exceptionname.

ex: declare

z varchar2(3);

begin

z := 'abcd';

dbms_output.put_line (z);

exception

when value_error then

dbms_output.put_line ('invalid string length');

end;

/

Exception propagation

Exceptions also raised in either declare section or executable section or in exception section.

IF exceptions are raised in executable section those exceptions are handled using either inner blocks or an outer block.

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

FRIDAY

1 FEB 12

24

055-311 WK-08

Whereas if exceptions are raised in declare section or in exception section those exceptions are handled using outer blocks only.

```

ex: begin
declare
z varchar2(3) = 'abcd';
begin
dbms_output.put_line(z);
exception
when value_error then
dbms_output.put_line('invalid string length');
end;
exception
when value_error then
dbms_output.put_line('invalid string length
handled using outer block');
end;
/

```

2. Userdefined Exceptions

We can also create our own exception names & also raise whenever necessary.

These types of exceptions are called user defined exception.

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

FEB 12

25

056-310 WK-08

- Step I. declare
- Step II. raise
- Step III. handling exception

Step I. declare

In declare section of the pl/sql program we are defining our own exceptionname using exception type.

Syntax: userdefinedexceptionname exception ;

ex: declare
a exception;

Step II. raise

Whenever necessary raise user defined exception either in executable section or in exception section, in this case we are using raise keyword.

Syntax: raise userdefinedexceptionname ;

In this case oracle server only raise that exception.

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

SUNDAY

RECORDED

26

step III. handling exception

we can also handle userdefined exceptions as same as predefined exception using predefined handler.

057-309 WK-08

syntax: when userdefinedexceptionname1 then statements;

when userdefinedexceptionname2 then statements;

when others then statements;

ex: declare

z exception;

begin

if to_char (sysdate, 'DY') = 'SUN' then
raise z;

end if;

exception

when z then

dbms_output.put_line ('my exception raised
today');

end;

/

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

MONDAY

FEB-12

27

058-308 WK-09

ex: declare
v-sal number (10);
z exception;
begin
select sal into v-sal from emp
where empno = 7902;
if v-sal > 2000 then
raise z;
else
update emp set sal = sal + 100
where empno = 7902;
end if;
exception
when z then
dbms_output.put_line ('salary
already high');
end;

1

Note

Generally user defined exceptions are used to transfer control to the number of block without checking each individual condition.

Raising Predefined Exceptions

We can also raise predefined exceptions

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦

TUESDAY

NPB 12

28

059-307 WK-09

using raise statement.

Syntax: raise predefinedexceptionname;

ex: declare

```

cursor C1 is select * from emp where
job = 'SW';
i emp%rowtype;
begin
open C1;
fetch C1 into i;
if C1%rowcount = 0 then
raise no_data_found;
end if;
close C1;
exception
when no_data_found then
dbms_output.put_line ('your job not
available');
end;
/

```

-21 Aug-

Note

Generally user defined exceptions are also used to test exception propagation.

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

WEDNESDAY

FEB 12

29

060-306 WK-09

Exceptions raised in executable section

Exceptions are raised in executable section handled using innerblocks or outerblocks.

Using innerblock

```
ex: declare
    z exception;
begin
    raise z;
exception
    when z then
        dbms_output.put_line ('handled using
        inner block');
end;
/
```

(output) handled using inner block.

Using outerblock

```
ex: declare
    z exception;
begin
    begin
```

February	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	♦	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	♦	♦



```

raise z;
end;
exception
when z then
dbms_output.put_line ('handled using outer
block');
end;
/

```

Exceptions raised in declare section

Exceptions raised in declare section must be handled using outerblocks only.

```

ex: begin
declare
z varchar2 (3)= 'abcd';
begin
dbms_output.put_line ('handled using outer z
block');
end;
exception
when value_error then
dbms_output.put_line ('handled using
outer blocks');
end;
/

```

Notes



Exception raised in exception section

When exceptions are raised in exception section must be handled using outerblock only.

Ex. declare

z1 exception;

z2 exceptions;

begin

begin

raise z1;

exception

when z1 then

dbms_output.put_line ('z1 handled');

raise z2;

end;

exception

when z2 then

dbms_output.put_line ('z2 handled using
outer block only');

end;

/

ACTION PLAN

MARCH '12

WK	ACTION PLAN						MARCH '12
▼	Mon	Tue	Wed	Thu	Fri	Sat	Sun
9				1	2	3	4
10	5	6	7	8	9	10	11
11	12	13	14	15	16	17	18
12	19	20	21	22	23	24	25
13	26	27	28	29	30	31	

OBJECTIVES THIS MONTH

DATE

DESCRIPTION

REMARKS

THURSDAY

MAR-12

1

Error Trapping Functions

061-305 WK-09

There are two error trapping functions supported by oracle.

- i) sqlcode
- ii) sqlerrm

sqlcode returns numbers whereas sqlerrm returns error number with error message.

Sqlcode return values	meaning
0	no errors
negative (-)	oracle errors
100	no data found
1	user defined exception

Ex: declare

v_sal number (10);

begin

select sal into v_sal from emp;

dbms_output.put_line (sqlcode);

dbms_output.put_line (sqlerrm);

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

FRIDAY

MAR/12

2

062-304 WK-09

end;
/

(Output) \Rightarrow 1
user-Defined exception

raise_application_error()

If you want to display user defined exception messages in more descriptive form then we are using raise_application_error() i.e. this function is used to display user defined exception message as same as oracle error displayed format.

This function is used in either executable section or in exception section.

Syntax: raise-application_error (error number,
message); -20000 to -20999
 upto 512 characters

ex: declare
v-sal number(10);
z exception;
begin
select sal into v-sal from emp where
empno = 7902;
if v-sal > 2000 then

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

MAR 12

3

063-303 WK-09

raise z;

else

update emp set sal=sal+100

where emp = 7902;

end if;

exception

when z then

raise_application_error (-20752, 'salary
already high');

end;

/

(Output) ORA-20752: salary already high

Note

Generally this function is used in triggers because whenever condition is true it raise a message & also it stops invalid data entry according to condition whereas dbms_output.put_line displays plain text message but it does not stops invalid data entry.

- 22 Aug -

3. Unnamed Exceptions

If you want to handle other than oracle 20 predefined errors we are using unnamed method.

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

MAR-12

4

064-302 WK-09

because oracle define exception names for regularly occurred errors.

other than 20 they are not defining exception names.

In this case we are providing exception names & also associate this exceptionname with appropriate error no. using exception_init function.

Syntax: pragma Exception_init (userdefinedexceptionname, errornumber);

Here pragma is a compiler directive i.e. at the time of compilation only pl/sql runtime engine associate error number with exception name.

This fun is used in declare section of the pl/sql block.

ex: declare

z exception;

pragma exception_init (z , -1400);

begin

insert into emp (ename) values ('Murali');

exception

when z then

dbms_output.put_line ('not to insert nulls');

end;

/

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

MONDAY

MAR.12

5

065-301 WK-10

ex: declare

z exception;

pragma exception_init (z, -2292);

begin

delete from dept where deptno = 10;

exception

when z then

dbms_output.put_line ('not to delete
master record');

end;

/

? Write a pl/sql program handle -2291 error
using unnamed method from emp, dept
tables.

Soln

declare

z exception;

pragma exception_init (z, -2291);

begin

insert into emp (empno, deptno) values (1, 60);

exception

when z then

dbms_output.put_line ('not to insert other
than primary values');

end;

/

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

MAR 12

6

066-300 WK-10

Subprograms are named pl/sql blocks which is used to solve some particular task.

There are two types of subprograms supported by oracle :

- i. Procedures
- ii. Functions

Procedures may or may not return a value.

Functions must return a single value.

i. Procedures

Procedures are named pl/sql blocks which is used to solve some particular task & also procedures may or may not return values.

Whenever we are using create or replace keyword in front of procedure those procedures automatically permantly stored in database.

These type of procedures are also called as stored procedures.

Generally procedures are used to improve performance of the application because procedures internally having one time compilation i.e one time compilation program units automatically improves performance i.e whenever we are loading a procedure into database automati-

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

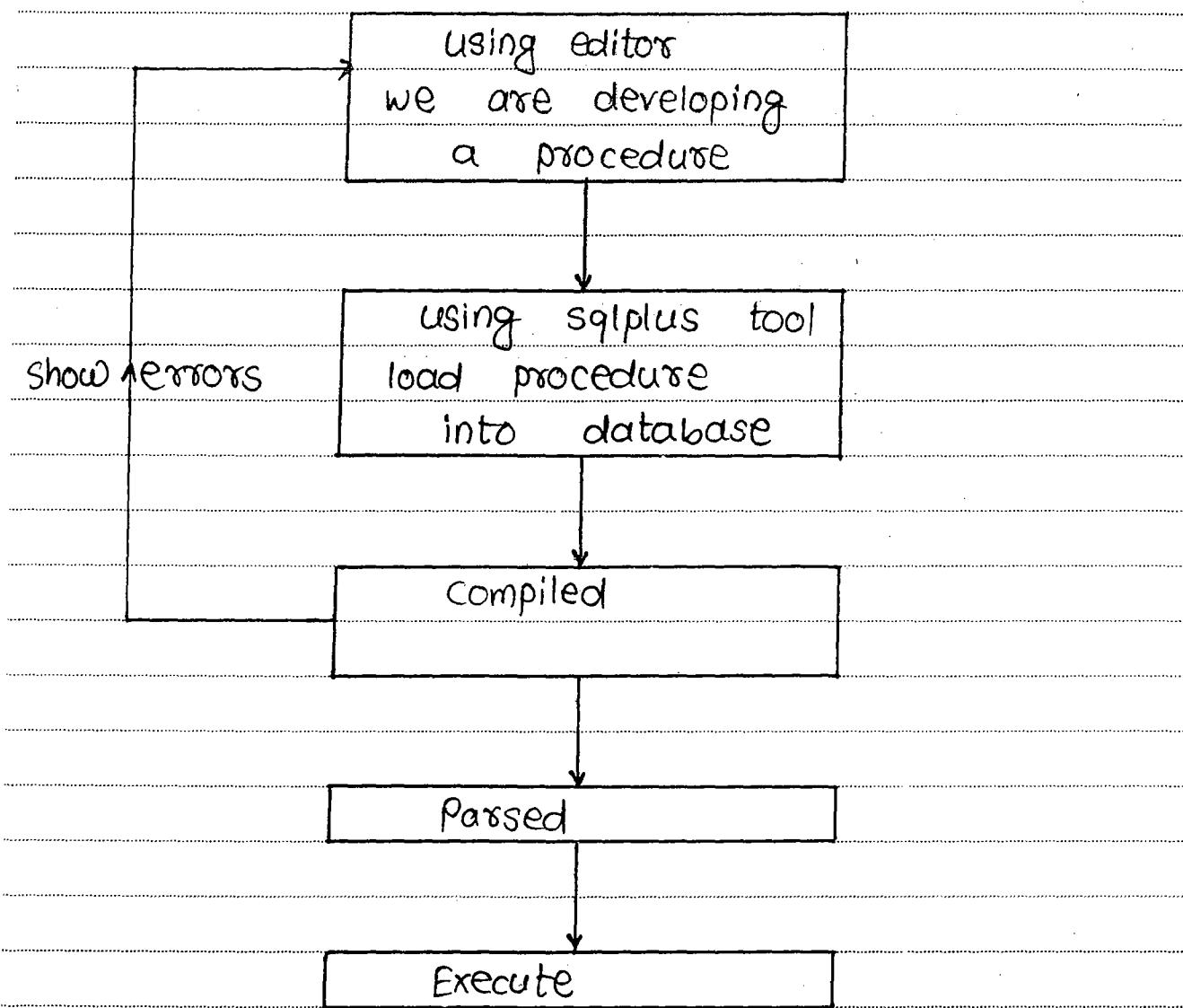
WEDNESDAY

MAR 12

7

Cally that procedure compiled.

067-299 WK-10



Every procedure having two parts :

- 1) procedure specification
- 2) procedure body

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

MAR 12

8

068-298 WK-10

In procedure specification we are specifying name of the procedure, type of the parameters.

Whereas in procedure body we are solving actual task.

Syntax: {create [or replace] procedure

procedure specification }
procedure specification {
procedure { IS / AS parametername [mode] datatype
variable declaration, cursors, | in
userdefined exceptions ; | out
procedure begin | in out
body { ----- [Exception]

end, [procedurename] ; }

executing a procedure

Method 1 :

sql> exec procedurename (actual parameters);

Method 2 : (using anonymous blocks)

sql> begin
procedurename (actual parameters);
end;
/

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

FRIDAY

MAR.12

9

Method 3:

sql> call procedurename (actual
parameters);

069-297 WK-10

? Write a pl/sql stored procedure for
passing empno as a parameter
display name of employee & his salary.

-23 Aug -

Solⁿ 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 4600

method 2 : using anonymous block

```
sql> begin  
      p1 (7566);  
end;
```

/

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

MAR 12

JONES 1064

10

method 3: sql> call p1(7902);
FORD 4600

070-296 WK-10

All stored procedures information stored under user_source data dictionary.

ex: sql> desc user_source;

sql> select text from user_source where NAME = 'P1';

? write a pl/sql stored procedure for passing deptno as a parameter to display employee details corresponding to that deptno.

Soln Create or replace procedure p1(p-deptno number)
is

cursor c1 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.sal||' '||i.deptno);

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SUNDAY

MAR 12

11

071-295 WK-10

end loop;

close c1;

end;

/

method 1:

sql> exec p1(10);

method 2:

begin

p1(20);

end;

/

Parameters used in procedures

parameters are used to pass the values into procedure & also return values from the procedure.

In this case we must use two types of parameters :

- 1) Formal parameters
- 2) Actual parameters

1) Formal parameters

Formal parameters are defined in procedure specification.

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

MONDAY

MAR 12

12

072-294 WK-11

In formal parameter we are defining parameter name & mode of the parameter.

There are three types of modes supported by oracle.

- i) in mode
- ii) out mode
- iii) in out mode

Syntax: parametername [mode] datatype

+---+
| in |
+---+
| out |
+---+
| in out |

Note

Whenever we are using procedures, funs, cursors we are not allow to use datatype size in formal parameter declaration.

i) in mode

By default procedure parameters having in mode.

in mode is used to pass the values into procedure body.

This mode behaves like a "constant" in procedure body. Through the in mode we can also pass default values using default or := operators.

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

MAR 12

13

073-293 WK-11

? write a pl/sql stored procedure
to insert a record into dept
table using in parameter.

Soln Create or replace procedure p1 (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 ('record inserted
  through procedure');
end;
/
```

There are three types of execution methods
supported by in parameter.

- (I) Positional notations
- (II) Named notations
- (III) mixed notations

(I) Positional parameter notations

ex: Sql> exec p1 (1, 'a', 'b');

(II) Named notations (\Rightarrow)

ex: Sql> exec p1 (p-dname \Rightarrow 'x', p-loc \Rightarrow 'y',
p-deptno \Rightarrow 2);

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

WEDNESDAY

MAR'12

14

074-292 WK-11

(III) Mixed notation

It is the combination of positional, named notation.

After positional their can be all named notations but after named their can not be positional.

ex: sql> exec p1 (3, p-loc=>'y', p-dname=>'xe');
record inserted through procedure

ii) Out mode

This mode is used to return values from procedure body.

Out mode internally behaves like a uninitialized variable in procedure body.

Here we must specify out keyword explicitly

Syntax: parametername out datatype

ex: create or replace procedure p1 (a in number, b out number)

is

begin

b = a*a;

end;

/

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

MAR 12

-24 Aug-

15

075-291 WK-11

In oracle if a subprogram contains out, in out parameters those subprograms are executed using following two methods

Method 1: using bind variable

Method 2: using anonymous blocks

Bind variable

These variables are session variables.

These variables are created at host environment that's why these variables are also called as host variables.

Generally these variables are used to pass the values betⁿ database server & client appⁿ.

Basically these variables are not a pl/sql variables, but we can also use these variables in pl/sql to execute subprograms having out parameter.

Step 1 : Creating a bind variable

Syntax: Sql> variable variablename datatype;

Step 2: Using a bind variable

Syntax: :variablename

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

FRIDAY

MAR 12

16

076-290 WK-11

Step 3: display value from bind variable

Syntax: sql> print bindvariablename;

ex:sql>Variable g number;

sql> declare

a number(10) :=500;

begin

:g := a/2

end;

/

sql> print g;

G

250

Method 1: Using bind variable

sql> variable z number;

sql> exec p1 (4, :z);

sql> print z;

Z

16

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SATURDAY

MAR.12.

17

method 2: using anonymous block

077-289 WK-11

ex: declare

x number(10);

begin

p1(6, x)

dbms_output.put_line(x);

end;

/

36

? Write a pl/sql stored procedure for passing employee name as in parameter return salary of that employee using out parameter from emp table.

SOLN create or replace procedure p1 (p-ename in varchar2 , p-sal out number)
is
begin
select sal into p-sal from emp where
name = p-name ;
end;

Execution :

method 1: using bind variable

Sql> Variable z number;

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

MAR 12

18

078-288 WK-11

Sql> exec p1 ('KING', :z);

Sql> print z;

Z

6800

Execution :

method 2: using anonymous block

```
declare  
x number (10);
```

```
begin
```

```
p1 ('SMITH', x);
```

```
dbms_output.put_line(x);
```

```
end;
```

```
/
```

2800

? Write a pl/sql stored procedure for passing deptno as a parameter return number of deptno's as out parameter from emp table.

Solⁿ Create or replace procedure p1 (p-deptno in number, p-a out number)

is

begin

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

MONDAY

MAR-12

19

079-287 WK-12

Select count (*) into p-a from emp
where deptno = p-deptno;
end;
/

Execution:

method 1 : using bind variable

sql > variable & number;
sql > exec p1(10, :&);

sql > print &;

R

3

iii) in out mode

This mode is used to pass the values into subprogram & return values from subprogram.
ie this mode internally behaves like a constant, initialized variable.

Here also explicitly we must specify in out keyword.

Syntax: parametername in out datatype

ex: create or replace procedure p1 (a in out number)

is

begin

a:=a*a;

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

MAR 12

20

080-286 WK-12

end;
/

Execution:

method 1: using bind variable

sql> variable z number;

sql> exec :z := 8;

sql> exec p1 (:z);

sql> print z;

z
—
64

Execution:

method 2: Using anonymous block

declare
x number(10) := &x;
begin
p1(x);
dbms_output.put_line (x);
end;
/

(output) enter value for x = 7

49

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

WEDNESDAY

MAR 12

21

081-285 WK-12

-25 Aug -

Create or replace procedure p1 (p-z
in out number)
is

begin
Select sal into p-z from emp where
Sal = p-z ;
end ;
/

Execution :

Sql> variable z number;
Sql> exec :z := 7902;

Sql> exec p1 (:z);

Sql> print z ;

z
4600

Pass by value, Pass by reference

Whenever we are using modular programming
two types of passing parameter mechanism
Supported by all languages

1. Pass by value
2. Pass by Reference

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

22

082-284 WK-12

These two parameter mechanism specifies when we are modifying formal parameters, actual parameters are affected or not.

In pass by value method actual value does not change in [calling] ^{main} program, Because copy of the values are passed into called program.

If we want to change actual parameters according to formal parameter modification we are using pass by reference method.

Oracle also support these two passing parameter mechanisms when we are using sub-program parameters & also by default all in parameters are passed by reference whereas all out parameters are passed by value.

Whenever we are returning large amount of data using out parameter again copy of the value is created because in databases out parameters are Pass by value. To overcome this problem oracle introduce no "NOCOPY" hint in the out parameter.

Syntax: parametername out nocopy datatype

ex: Create or replace procedure p1(p-ename in varchar2 , p-sal out , number)
nocopy
is

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

FRIDAY

MAR 12

23

083-283 WK-12

begin

select sal into p-sal from emp

where ename=p-ename;

end;

/

Autonomous Transactions

Autonomous transactions are independent transactions used in either procedures or in triggers.

Generally autonomous transactions are used in child procedures, these procedures are not affected from the main transactions when we are using commit or rollback.

If we want to convert a procedure into autonomous we must use "autonomous_transaction pragma , commit".

Syntax: pragma autonomous_transaction ;

This pragma used in declare section of the procedure.

using autonomous transaction

ex:

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

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

MAR 12

24

084-282 WK-12

```
sql> Create or replace procedure p1  
is  
pragma autonomous_transaction;  
begin  
insert into test values ('india');  
commit;  
end;  
/
```

```
sql> begin  
insert into test values ('hyd');  
insert into test values ('mumbai');  
p1;  
rollback;  
end;  
/
```

```
sql> Select * from test;
```

Name

india

without autonomous transaction

ex:

```
sql> delete from test;
```

```
sql> Create or replace procedure p1  
is
```

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SUNDAY

MAR-12

25

085-281 WK-12

```
begin
insert into test values ('india');
commit;
end;
/
```

```
Sql> begin
      insert into test values ('hyd');
      insert into test values ('mumbai');
      p1;
      rollback;
end;
/
```

```
Sql> select * from test;
```

NAME

hyd
mumbai
india

Note

When a procedure contains commit pl/sql runtime engine not only commit procedure transaction but also commit above of the procedure transaction. To overcome this problem oracle 8i introduce autonomous transaction.

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

- 26 Aug

MONDAY

MAR 12

AUTHID CURRENT_USER

26

When we are reading data from table & 086-280 WK-13
also provides security for the data, database developer using authid current_user clause in procedure.

When we are using this clause another user can not execute procedure if database administrator giving privileges also.

giving procedure privilization

Syntax: grant execute on procedurename to
user1, user2, ... ;

Syntax: create or replace procedure procedurename
(formal parameters)

authid current-user

is/as

→ variable declarations, cursors,
userdefined exceptions;

begin

[exception]

end;

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

MAR 12

27

087-279 WK-13

ex: sql> conn scott/tiger;

sql> create or replace procedure p1 (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> grant execute on p1 to murali;

sql> conn murali/murali;

sql> set serveroutput on;

sql> exec scott.p1(7902);

error:- table or view does not exist.

Handled or unhandled exception in procedure

Whenever we are calling inner procedures into

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

WEDNESDAY

MAR 12

28

088-278 WK-13

outer procedures we must implement
inner procedures exception handler

otherwise pl/sql runtime engine execute

outer procedure default procedure handler.

Inner procedure

ex: create or replace procedure p1 (x in number,
y in number)

is

begin

dbms_output.put_line (x/y);

exception

when zero_divide then

dbms_output.put_line ('y can not be zero');

end;

/

Outer procedure

ex: sql> create or replace procedure p2

is

begin

p1(7,0)

exception

when others then

dbms_output.put_line ('any error');

end;

/

Sql> exec p2;

(output) \Rightarrow y can not be zero

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

THURSDAY

MAR 12

29

089-277 WK-13

All stored procedure information
stored under user-procedure,
user-source.

we can also drop procedure using -

drop procedure procedurename only.

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	
	25	26	27	28	29	30	31																		

FRIDAY

MAR 12

30

090-276 WK-13

ii. Functions

Function is a named pl/sql block which is used to solve some particular task. And also by default functions returns a single value.

Functions also having two parts :

1. Function specification
2. Function body

In function specification we are specifying name of the function & type of the parameters whereas in function body we are solving the actual task.

Syntax: create or replace function functioname(
formal parameters)

return datatype parametername [mode]
 IS / AS datatype

→ variable declarations, cursors,
 Begin

Return expression

end [function];

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	♦

SATURDAY

MAR 12

31

Executing a function

091-275 WK-13

Method 1 :-

Select functionname (actual parameters) from dual;

Method 2:- (using anonymous block)

```
begin  
variablename := functionname (actualparameters);  
end;
```

ex: sql> create or replace function f1(a varchar2)
return varchar2

```
is  
begin  
return a;  
end;  
/
```

execution : method 1

sql> select f1 ('hi') from dual;

(output) \Rightarrow hi

method 2 (using anonymous block)

```
declare  
z varchar2(10);
```

March	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

WK

ACTION PLAN

APRIL '12

Mon	Tue	Wed	Thu	Fri	Sat	Sun
30						1
2	3	4	5	6	7	8
14						
9	10	11	12	13	14	15
15						
16	17	18	19	20	21	22
17	23	24	25	26	27	28
						29

OBJECTIVES THIS MONTH

DATE

DESCRIPTION

REMARKS

SUNDAY

APRIL 12

1

```
begin
z := f1 ('hi');
dbms_output.put_line (z);
end;
```

092-274 WK-13

/

? write a pl/sql stored function for passing number return a message either even or odd based on that number.

```
SQL> create or replace function f1 (a number)
return varchar2
is
begin
if mod (a,2) = 0 then
return 'even';
else
return 'odd';
end if;
end;
```

/

execution: method 1

```
sql> select f1(5) from dual;
```

/

(output) \Rightarrow odd

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	♦

MONDAY

APR - 2

2

093-273 WK-14

Method 2 : using anonymous block

```

declare
x varchar2(10);
begin
x := f1(6);
dbms_output.put_line(x);
end;
/

```

Method 3: using bind variable

sql> variable z varchar2(10);

```

sql> begin
:z := f1(7);
end;
/

```

sql> print z;

(output) z
odd

Method 4 :

sql> exec dbms_output.put_line(f1(3));

(output) odd

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

APR 12

3

Method 5:

094-272 WK-14

begin

dbms_output.put_line (f1(4));

end;

/

(output) \Rightarrow even

-27 Aug -

Note

we can also use user defined statement
fun in insert statement.

ex: sql> create table y1 (sno number(10),
msg varchar2(10));

sql> insert into y1 values (1, '# f1(8));

sql> select * from y1;

SNO	MSG
1	even

? Write a pl/sql stored function for passing
emno as parameter return gross salary
from emp table based on following condition

gross := basic + hra + da - pf;

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

WEDNESDAY

APR 12

hra ---> 10% of sal
 da ---> 20% of sal
 pf ---> 10% of sal

4

095-271 WK-14

SOLⁿ create or replace function f1 (p-empno
 number)

return number

is

v-sal number (10) ;

gross number (10) ;

hra number (10) ;

da number (10) ;

pf number (10) ;

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 * 0.1 ;

gross := v-sal + hra + da + pf ;

return gross ;

end ;

execution: method 1

sql> Select f1 (7566) from dual ;

(output) → 60987

Method 2 : using anonymous block

declare

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

APR 12

5

096-270 WK-14

z number (10);

begin

z := p1 (7566);

dbms_output.put_line (z);

end;

/

Note

We can also use predefined aggregate fun in user defined functions and also # use this user defined functions in same table or different table.

Ex: Create or replace function maximum
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, maximum-sal from emp;

Note

We are not allow to use DML statements in functions.

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

FRIDAY

APRIL 12

6

097-269 WK-14

Out

If we want to return more no. of values from fun we are using out parameter

ex: create or replace function f1 (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;

/

execution : using bind variable

Sql> variable a varchar2(10);

Sql> variable b varchar2(10);

Sql> variable c varchar2(10);

sql> begin
:a := f1 (10,:b,:c);
end;
/

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

APR 12

7

Sql > print b c ;

098-268 WK-14

(Output) \Rightarrow B

ACCOUNTING

C

NEW YORK

PL/SQL

? Write a \uparrow stored function for passing
emno, date as parameter return number
of years that employee is working
based on date from emp table.

Soln Create or replace function f1 (p-emno
number, p-date date)

return number

is

z number (10);

begin

Select months_between (p-date, hiredate) /
12 into z from emp where emno=p-emno;
return round(z);

end;

/

execution

Sql > Select emno, ename, hiredate, f1(emno,

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SUNDAY

APR 12

8

sysdate) || 'years' "exper" from emp
where empno = 7902

/

099-267 WK-14

EMPNO	ENAME	HIREDATE	EXPER
7902	FORD	03-DEC-81	31 years

Note

we are not allow to use named, mixed notations in fu's. To overcome this problem Oracle 11g introduce named, mixed notation.

- 28 Aug -

Note

we can also develop our own aggregatable fu's & also use these fu's in group by clause and also if these fu's returns multiple values, we must use cursors in these user defined fu's.

ex: create or replace function f1 (p-deptno number)

return varchar2

is

a varchar2 (200);

cursor c1 is select ename from emp
where deptno = p-deptno;

begin

for i in c1

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

MONDAY

APR 2

9

100-266 WK-15

```
loop
a := a || '|| i.ename ;
end loop;
return a;
end;
1
```

execution:

```
sql> select deptno, f1(deptno) from emp
group by deptno
```

Note

Oracle 10g introduce `wm_concat()` predefined aggregate fun which returns number of rows group wise. This fun accept all datatype columns.

```
ex: Select deptno, wm_concat(ename) from emp
group by deptno
```

```
ex: select job, wm_concat(ename) from emp
group by job
```

? write a pl/sql stored function for passing empno as parameter, calculate tax based on following condition using emp table

- i) IF annual sal > 10000 then tax → 10%.
- ii) IF annual sal > 15000 then tax → 20%.
- iii) If annual Sal > 20000 then tax → 30%.

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

TUESDAY

APR 12

SQlD create or replace function f1
(p-empno number)

return number

is

v-sal number (10);

annsal number (10);

it number (10);

begin

Select sal into v-sal from emp where

empno = p-empno;

annsal := v-sal * 12;

if annsal > 10000 ~~then~~ and annsal <= 15000
then

it := annsal * 0.1;

elsif annsal > 15000 and annsal <= 20000

then

it := annsal * 0.2;

elsif annsal > 20000 then

it := annsal * 0.3;

else

it := 0;

end if;

return it;

end;

/

sql> select f1 (7902) from dual;

10

101-265 WK-15

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

WEDNESDAY

APR 12

Note

11

we can also drop function using
drop function functionname;

102-264 WK-15

triggers

trigger is also same as stored procedure & also it will automatically invoked whenever DML operation performed against table or view. There are two types of triggers supported by pl/sql.

1. Statement level triggers
2. Row level triggers

In statement level trigger, trigger body is executed only once for DML statement. Whereas in row level trigger, trigger body is executed ~~on~~ for each row for DML statements.

Syntax:

```
create [or replace] trigger triggername
before /after /Triggers trapping Trigger event
trigger      insert / update / delete on tablename
Specification [ for each row ] --> row level
               [ where condition ] -> trigger
               [ declare ]
```

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

APR 12

12

103-263 WK-15

→ variable declarations, cursors;

trigger
body

```

begin
  — —
  — —
end;

```

Row Level Triggers

In row level trigger, trigger body is executed for each row for DML statement, that's why we are using for each row clause in trigger specification and also data internally stored in 2 rollback segment qualifiers these are old, new.

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

When we are using these modifiers in trigger body we must use colon prefix in the qualifiers.

Syntax: :old.Columnname

(or)

:new.Columnname

But when we are using these qualifiers in when clause we are not allow to use colon in front of the qualifiers.

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

FRIDAY

APR 12

13

104-262 WK-15

Syntax: Old.Columnname
(or)

new.Columnname

	insert	update	delete
:new	✓	✓	✗
:old	✗	✓	✓

? write a pl/sql row level trigger on emp table whenever user inserting data into a emp table sal should be (above) more than 5000.

```

SOL' create or replace trigger tz1
before insert on emp
for each row
begin
if :new.sal < 5000 then
raise_application_error (-20123, 'Salary should
be more than 5000');
end if;
end;
/

```

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

⇒ ORA-20123 : salary should be more than 5000

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SATURDAY

- 29 Aug -

APR 12

? write a pl/sql trigger using
using emp, dept table implement on delete
cascade concept, Without using on delete
clause.

14

105-261 WK-15

Solⁿ Create or replace trigger t81
after delete on dept
for each row
begin
delete from emp where deptno = :old.deptno;
end;
/

Sql> delete from dept where deptno = 20;

? write a pl/sql row level trigger on dept
table whenever updating deptno's in dept
table automatically those deptno's modified into
emp table.

Solⁿ

create or replace trigger t81
after update on dept
for each row
begin
update emp set deptno = :new.deptno where
deptno = :old.deptno;
end;
/

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

APR 12

15

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

106-260 WK-15

sql> select * from emp;

auto increment

If we want to generate primary key values automatically all database system uses auto increment concept.

In oracle if we want to implement auto increment concept we are using triggers, or sequences i.e we are creating sequences in sql and use that sequences in triggers.

ex:sql>create sequence s1 start with 1;

sql> create table test (sno number(10) primary key, name varchar2(10));

sql> create table or replace trigger tw2

before insert on test

for each row

begin

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

end;

/

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

sql> insert into test (name) values ('murali');

16

107-259 WK-16

sql> insert into test (name) values ('subhas');

sql> insert into test (name) values ('Satish');

sql> select * from test;

(output) =>	sno	name
	1	murali
	2	Subhas
	3	Satish

Generally if we want to generate sequence values we are using dual table in pl/sql blocks, but Oracle 11g introduce without dual table also we are generating sequences in pl/sql block.

syntax: begin
variablename := sequencename.nextval;
end;

11g : Create or replace trigger tw2
before insert on test

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

APR 12

17

108-258 WK-16

for each row

begin

:new.sno := s1.nextval;

end;

/

Ex: Create or replace trigger tw2

after insert on test

for each row

begin

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

end;

/

error:- ↗ Cannot change NEW values for this
trigger type

before, after triggers

In before triggers, trigger body is executed before DML statements are affected into database, whereas in after triggers, trigger body is executed after DML statements are affected into database. Generally if we want to restrict invalid data entry always we are using before triggers, whereas if we are performing operation on the one table those operations are affected into another table

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

WEDNESDAY

APR 12

18

109-257 WK-16

then we are using after trigger.
whenever we are inserting
values into new qualifier we must
use before triggers otherwise oracle server
returns an error.

? write a pl/sql row level trigger whenever user inserting data into ename column after inserting data must be converted into upper case.

SQL create or replace trigger tw3
before insert on emp
for each row
begin
:new.ename := upper (:new.ename);
end;
/

SQL> insert into emp (empno, ename) values (1, 'abc');

- 30 Aug -

In statement level trigger, trigger body is executed only once for each DML statement that's why generally statement level triggers used to define type based condition and also used to implement auditing reports.

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

APR'12

19

These triggers does not contain new, old qualifiers.

110-256 WK-16

? write a pl/sql statement level trigger on emp table not to perform DML operations in saturday, sunday.

SOLⁿ Create or replace trigger te1
before insert or update or delete on emp
begin
if to_char(sysdate, 'DY') in ('SAT', 'SUN')
then
raise_application_error (-20123, 'we can not
perform DMLs in sat, sun day');
end if;
end;
/

Sql> delete from emp where empno=7902;

⇒ ora-20123 : we can not perform DMLs in sat,
sun day.

? write a pl/sql statement level trigger
on emp table not to perform DML
operation on last day of the month

SOLⁿ Create or replace \$ trigger te2
before insert or update or delete on emp

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

FRIDAY

APR '12

20

111-255 WK-16

```
begin
if sysdate = last_day(sysdate) then
raise_application_error (-20123, 'we can
not perform DMLs on lastday');
end if;
end;
/
```

sql> delete from emp where empno=7902;

h

⇒ ORA-20123 : we cannot perform DMLs on lastday.

Note 1

If to_char(sysdate) = to_char(last_day(sysdate)) then

Note 2

If to_char(sysdate, 'DR') = TO_CHAR(last-day(sysdate), 'DR')) then

Triggering Events [OR]

Trigger Predicate clauses

If we want to define multiple conditions on multiple tables then all database systems uses triggering events.

These are inserting, updating, deleting clauses

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

SATURDAY

APR 12

21

112-254 WK-16

syntax: if inserting then
statements;

elsif updating then
statements;

elsif deleting then
statements;

end if;

These clauses are used in either row level or statement level triggers.

? Write a pl/sql statement level trigger on emp table not to perform any DML operation in any days using triggering event

Soln create or replace trigger te3
before insert or update or delete on emp
begin
if inserting then
raise_application_error(-20123, 'we can not
perform inserting operation');
elsif updating then
raise_application_error(-20124, 'we can not
perform update operation');
elsif deleting then
raise_application_error(-20125, 'we can not
perform deleting operation');

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	♦							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	♦

SUNDAY

APR 12

22

113-253 WK-16

end if;
end;
/

ex: sql> create table test (msg varchar2(10));

sql> create or replace trigger te4
after insert or update or delete on emp
declare
z varchar2(20);
begin
if inserting then
z := 'rowsinserted';
elsif updating then
z := 'rowsupdated';
elsif deleting then
z := 'rowsdeleted';
end if;
insert into test values (z);
end;
/

sql> insert into emp (empno) values (1);
sql> insert into emp (empno) values (2);
sql> insert into emp (empno) values (3);

sql> update emp set sal = sal + 100 where
deptno = 30;

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

MONDAY

APR 12

23

⇒ 6 rows updated

114-252 WK-17

sql> select * from test;

⇒ MSG

rowsinserted

rowsinserted

rowsinserted

rowsupdated

? write a pl/sql row level trigger on emp table whenever user inserting data those values stored in another table, whenever user updating data those values stored in another table, whenever user deleting data those values are stored in another table.

sql> create table t1 as select * from emp
where 1=2 ;

sql> create table t2 as select * from emp
where 1=2 ;

sql> create table t3 as select * from emp
where 1=2 ;

sql> create or replace trigger t45
after insert or update or delete on emp

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

APR 12

24

115-251 WK-17

for each row
begin
if inserting then
insert into t1 values (:new.empno, :new.ename,...);
elsif updating then
update insert into t2 values (:old.empno,
:old.ename,...);
elsif deleting then
insert into t3 values (:old.empno, :old.ename
...);
end if;
end;

/

Execution order in Triggers

I. before statement level

II. before row level

III. after row level

IV. after statement level

whenever we are creating same level of triggers on same table we can not control execution order of the triggers.

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

WEDNESDAY

APR 25

25

116-250 WK-17

To overcome this problem Oracle 11g introduces follows clause in triggers.

Follows clause provides "guarantee" execution order.

This clause used in trigger specification only.

- 31 Aug

Syntax: create or replace trigger triggername

before / after

insert / update / delete on tablename

[for each row]

follows

another trigger1, trigger2, ..

declare

begin

— —

— —

end;

{ ex: Create or replace trigger tj2

before insert on test

for each row

declare } // This is wrong one

ex: create table test (col1 number(10), col2
number(10), col3 date);

sql> create sequence s1 start with 5878;

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

THURSDAY

APR 12

26

117-249 WK-17

sql> create or replace trigger tj1
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 tj2

before insert on test

for each row

declare

v_col2 varchar2(10);

begin

select to_char(reverse(:new.col1)) into
v_col2 from dual;

:new.col2 := v_col2;

dbms_output.put_line ('trigger2 fired');

end;

/

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

trigger2 fired

trigger1 fired

1 row created

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

FRIDAY

APR '12

27

118-248 WK-17

sql> select * from test;

	Col 1	Col 2	Col 3
	5878		81-AUG-12

Solution (Oracle 11g)

create or replace trigger tj2
before insert on test
for each row
follows tj1
declare

Calling a procedure into trigger

Using call statement we are calling procedure
into trigger.

Syntax: create or replace trigger triggername
before / after
insert / update / delete on tablename
call procedurename
/

ex: sql> create table test (totsal number(10));

sql> create or replace procedure p1

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

SATURDAY

APR 12

28

119-247 WK-17

is
z number (10);

begin

delete from test;

select sum (sal) into z from emp;

insert into test values (z);

end;

/

Sql> Create or replace trigger t13

after insert or update or delete on emp

call p1

/

Sql> update emp set sal = sal + 100 ;

Sql> select * from test;

(output) TOTSAL

44956

? write a pl/sql trigger on emp table whenever user deleting records from emp table automatically display remaining number of existing record number in bottom of the delete statement.

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

APR 12

29

120-246 WK-17

SOLN SQL > create or replace trigger tj4
after delete on emp

declare

z number(10);

begin

select count(*) into z from

emp;

dbms_output.put_line(z);

end;

/

SQL > delete from emp where empno=7902;

(Output) \Rightarrow 13

ex: Create or replace trigger tj4

after delete on emp

for each row

declare

z number(10);

begin

select count(*) into z from emp;

dbms_output.put_line(z);

end;

/

SQL > delete from emp where empno=1;

error:- ora-4091 : table scott.emp is mutating

April	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

MONDAY

-01 Sep-

APR '12

Mutating

30

121-245 WK-18
Into a row level trigger based on a table trigger body can not read data from same table and also we can not perform DML operations on same table.

If we are trying this oracle server returns an error ora-4091 : table is mutating.

This error is called mutating error.

This trigger is called mutating trigger.

This table is called mutating table.

Mutating errors are not occurred in statement level trigger because through these statement level trigger when we are performing DML operations automatically data committed into the database, whereas in row level triggers when we are performing transaction data is not committed & also again we are reading this data from the same table then only mutating error is occurred.

To avoid mutating errors we are using autonomous transaction in triggers.

But autonomous transaction gives previous result.

Soln - (using autonomous transaction)

Sql> create or replace trigger tj1
after delete on emp

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Notes



```
for each row  
declare  
z number(10);  
pragma autonomous_transaction;  
begin  
select count(*) into z from emp;  
commit;  
dbms_output.put_line (z);  
end;  
/
```

sql> delete from emp where empno=2;

(output) \Rightarrow 16

DDL Triggers

we can also create triggers on schema level, database level. These type of triggers are called DDL triggers or system triggers.

These type of triggers are created by database administrator.

These triggers are created on DDL events.

Syntax: Create or replace trigger triggername
before / after
create / alter / drop / truncate / rename
on username.schema

WK

ACTION PLAN

MAY '12

Mon	Tue	Wed	Thu	Fri	Sat	Sun
18		1	2	3	4	5
19	7	8	9	10	11	12
20	14	15	16	17	18	19
21	21	22	23	24	25	26
22	28	29	30	31		

OBJECTIVES THIS MONTH

APRIL '12

	T	F	S	S	M	T	W	T	F	S	S
	1	2	3	4	5	6	7	8	9	10	11
E	12	13	14	15	16	17	18	19	20	21	22
S	26	27	28	29	30						

2012

TUESDAY

MAY 12

1

122-244 WK-18

declare

begin

end;

/

? Write a pl/sql DDL trigger on scott schema not to drop emp table.

Solⁿ create or replace trigger tj2
before drop on scott. schema

begin

if ora_dict_obj_name = 'EMP' and

ora_dict_obj_type = 'TABLE' then
raise_application_error (-20123, 'we can not
drop emp table');

end if;

end;

/

sql> drop table emp;

ora-20123: we can not drop emp table.

Note

There are 12 types of triggers supported
by oracle based on statement, row level

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

WEDNESDAY

MAY 12

2

123-243 WK-18

before, after, insert, update, delete
 and also oracle support instead
 of triggers on views & DDL
 triggers are databases.

enable / disable triggers

1) enable / disable a single trigger

Syntax: alter trigger triggername enable/disable;

2) enable / disable all triggers in a table

Syntax: alter table tablename enable/disables
 all triggers;

ex: alter table emp disable all triggers;

All triggers information stored under
 user_triggers; datadictionary.

ex: desc user_triggers;

we can also drop trigger.

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

THURSDAY

MAY 17

3

124-242 WK-18

Packages

Package is a database object which is used encapsulate variables, constants, procedures, cursors, functions, types into single unit.

Packages does not accept parameters, can not be nested, can not be invoked.

Generally packages are used to improve performance of the application because when we are calling packaged subprogram first time total package automatically loaded into memory area. Whenever we are calling subsequent subprogram calls pl/sql run time engine calling those subprogram from memory area. This process automatically reduces disk I/O that's why packages improves performance of the application.

Packages having two parts

1. Package Specification
2. Package body

In package specification we are defining global data and also declare objects, subprograms whereas in package body we are implementing subprograms & also package body subprogram

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

FRIDAY

MAY 12

4

125-241 WK-18

internally behaves like a private
subprogram.

Package Specification

Syntax: create or replace package packagename
 is / AS
 global variable declaration;
 constant declaration;
 cursor declaration;
 types declaration;
 procedure declaration;
 function declaration;
 end;
 /

Package Body

Syntax: create or replace package body
 packagename
 IS / AS
 procedure implementations;
 function implementations;
 end;
 /

Invoking packaged subprograms

1> sql> exec
 packagename.procedurename (actual
 parameters);

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SATURDAY

MAY 12

5

126-240 WK-18

2) select packagename.functionname
(actual parameters) from dual;

ex: Sql > create or replace package ph1
is

```
procedure p1;  
procedure p2;  
end;  
/
```

ex: Sql > create or replace package body ph1
is

```
procedure p1  
is  
begin  
dbms_output.put_line ('First proc');  
end p1;  
procedure p2  
is  
begin  
dbms_output.put_line ('Second proc');  
end p2;  
end;  
/
```

execution

Sql > exec ph1.p2;

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

MAY 12

6

127-239 WK-18

global variables used in package

In pl/sql we are defining global variables in package specification only.

ex: sql> create or replace package ph2
 is
 g number(10) := 500 ;
 procedure p1
 function f1 (a number) return number;
 end;
 /

sql> create or replace package body ph2
 is
 procedure p1
 is
 z number(10);
 begin
 $z := g/2;$
 dbms_output.put_line(z);
 end p1;
 function f1 (a number) return number
 is
 begin
 return a*g;
 end f1;
 end;
 /

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

MONDAY

MAY 12

7

128-238 WK-19

sql> exec ph2.p1;

(output) 250

sql> select ph2.f1(3) from dual;

(output) 1500

-03 Sep-

State of the global variable

If we want to maintain state of the global variable we are using serially-reusable pragma in packages.

Syntax: pragma serially_reusable;

ex: sql> create or replace package pj1
is g number(10):=5;
pragma serially_reusable;
end;
/

sql> begin
pj1.g := 70
end;
/

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

MAY 12

8

129-237 WK-19

```
sql> begin
      dbms_output.put_line ( pj1.g );
    end;
/

```

(output) 5

State of the globalized cursor

```
ex: sql> create or replace package pj2
  is
    cursor c1 is select * from emp;
    pragma serially_reusable ;
  end;
/

```

```
sql> begin
      open pj2.c1 ;
    end;
/

```

```
sql> begin
      open pj2.c1;
    end;
/

```

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

WEDNESDAY

MAY 12

9

Overloading Procedures

130-236 WK-19

overloading refers to same name can be used for different purposes i.e we are implementing overloading procedures through packages only, these procedures having same name and also different type and number of arguments.

ex: sql> create or replace package pj1

is

procedure p1(a number, b number);

procedure p1(x number, y number);

end;

/

sql> create or replace package body pj1

is

procedure p1 (a number, b number) -

is

c number(10);

begin

c:=a+b;

dbms_output.put_line(c);

end p1;

end;

+

procedure p1 (x number, y number)

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

MAY 12

10

131-235 WK-19

is

z number (10);

begin

z := x - y;

dbms_output.put_line (z);

end p1;

end;

/

execution

sql> exec pj1.p1 (a=>9, b=>5);

(output) 14

sql> exec pj1.p1 (x=>8, y=>5);

(output) 3

Forward declaration

Whenever we are calling procedures into another procedure then only we are using forward declaration i.e. whenever we are calling local procedure into global procedure first we must implement local procedures before calling otherwise use a forward declaration in package body.

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

FRIDAY

MAY 12

11

ex: Sql> create or replace package pj2,
is

procedure p1;
end;

/

Sql> create or replace package body pj2

is

procedure p2;

procedure p1

is

begin

p2;

end;

procedure p2

is

begin

dbms_output.put_line ('Local proc');

end p2;

end;

execution

Sql> & exec pj2.p1;

(output) local proc

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

MAY 12

12

133-233 WK-19

Types used in Packages

In Oracle

we can also create our own user defined types using type keyword.

When we are creating types we are using two step process -

We are creating type using appropriate syntax then only we are creating a variable of that type.

Oracle server supports following types

- 1. PL/SQL Record
- 2. Index by table (or)
PL/SQL table (or)
- Collections
Associative arrays
- 3. Nested tables
- 4. Varrays
- 5. Ref cursors

1. PL/SQL Record

This is an user defined type which is used to represent different datatypes into single unit.

It is also same as structures in C language.

This is an user defined type so we are creating two step process.

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SUNDAY

MAY 12

13

134-232 WK-19

Syntax: (i) Type Typename is Record (attribute1 datatype (size), attribute2 datatype (size), ...);

(ii) variablename Typename;

ex: Sql> create or replace package pj3

is

type t1 is record (a1 number(10),
a2 number(10), a3 number(10));
procedure p1;
end;
/

sql> create or replace package body pj3

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;

/

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

MONDAY

MAY 12

14

135-231 WK-20

execution

sql> exec pj3.p1;

(output) 7839 KING 7700

2. Index by Table

This is an user defined type which is used to store multiple data items in a single unit.

Basically this is an unconstraint table.

Generally these tables are used to improve performance of applications because these tables are stored in memory area that's why these tables are also called as memory tables.

Basically these table contains key value pairs i.e. actual data is stored in value field & indexes are stored in

These indexes range from negative to positive numbers.

These indexes are not consecutive.

These indexes key behaves like a primary key i.e it does not accept duplicate, null values.

Basically this key datatype is binary-integer.

- 04 Sep -

Syntax: ① Type Typename is Table of datatype (size)
index by binary-integer;

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

TUESDAY

MAY 15

15

136-230 WK-20

① variablename Typename;

If we want to manipulate data in collection
we are using collection methods.

Index by table having following collection methods
exists, first, last, prior, next, count, delete (range
of indexes), delete.

ex. 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;
  v_t(5) := 50;
  dbms_output.put_line ( v_t(3));
  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(4) );
  dbms_output.put_line ( v_t.count );
  v_t.delete (1,3);
  dbms_output.put_line ( v_t.count );
  v_t.delete;
  dbms_output.out_line ( v_t.count );
end;
```

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

16

? Write a pl/sql prog to transfer all employee names from emp table & store it into index by table & also display data from index by table.

```

SOL) 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%not found;
        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;

```

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

THURSDAY

MAY 12

17

138-228 WK-20

ex: 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;
1

ex: declare
type t1 is table of date
index by binary_integer;
v_t t1;
begin
for i in 1..10
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;
1

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

FRIDAY

MAY 12

18

139-227 WK-20

? write a pl/sql prog to retrieve all joining dates from emp table & store it into index by table & also display content from index by table.

Soln 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;
/

```

ex: declare

```

type t1 is table of varchar2(10)
index by varchar2(10);
v_t t1;
x varchar2(10);
begin
v_t('a') := 'murali';
v_t('b') := 'abc';
v_t('c') := 'xyz';

```

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SATURDAY

MAY 12

19

140-226 WK-20

$x := 'a'$

loop

dbms_output.put_line (v_t(x));

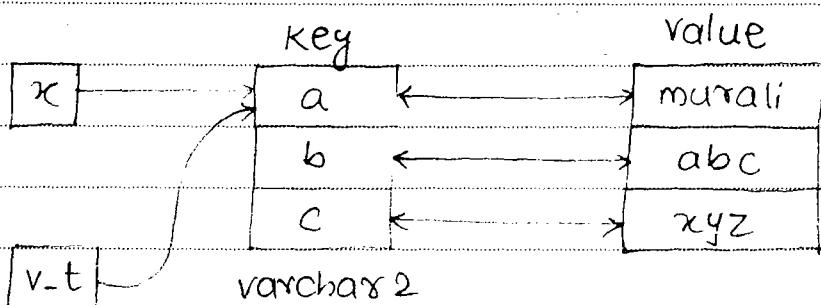
$x := v_t.next(x);$

exit when $x \neq$ null;

end loop;

end;

/



-05 Sep-

ex: declare

type t1 is table of emp%rowtype

index by binary_integer;

v_t t1;

x number(10);

begin

Select * bulk collect into v_t from emp;

$x := 1;$

loop

dbms_output.put_line (v_t(x).empno || '||'

v_t(x).ename || '||' v_t(x).sal);

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

MAY 12

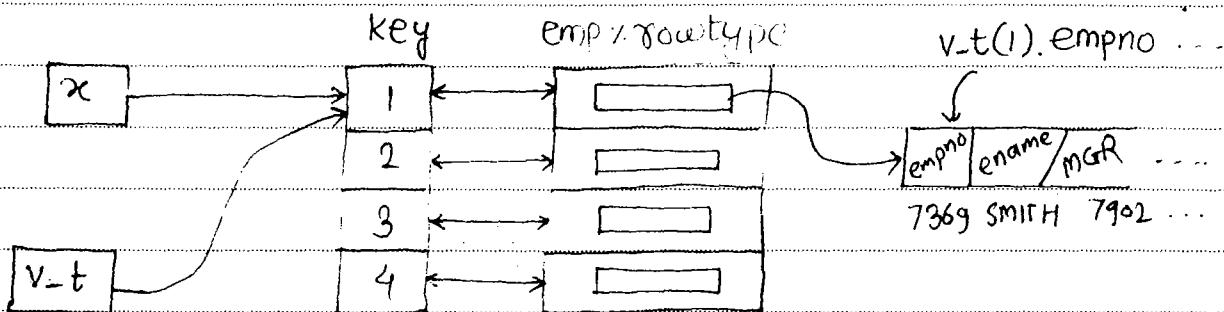
20

141-225 WK-20

```

x := v_t.next(x);
exit when x is null;
end loop;
end;
/

```



14

(or)

```

ex: 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 );
        end loop;
    end;
/

```

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

MONDAY

MAY 12

21

Return Result Set

142-224 WK-21

If we want to return large amount of data from Oracle database into client application we are using following two methods

- a) method 1 :- Using index by tables
- b) method 2 :- Using ref cursors

Method 1 :- Using index by tables

If we want to implement these type of appn we must implement one server appn using functions. Here we are specifying fun return type as user-defined type & also to execute the server appn we must implement client appn.

Server Application

ex: create or replace package p81

is

type t1 is table of emp%rowtype

index by binary_integer;

function f1 return t1;

end;

/

sql> create or replace package body p81

is

function f1 return t1

is

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr.	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

MAY 22

22

143-223 WK-21

```

is
v-t t1;
begin
  select * bulk collect into v-t from emp;
  return v-t;
end f1;
end;
/

```

To execute this app we use pl/sql client
(execution)

-ex: declare

```

x pr1.t1;
begin
x:=pr1.f1;
for i in x.first..x.last
loop
dbms_output.put_line (x(i).ename || '||' || x(i).sal);
end loop;
end;
/

```

Method 2:- Using ref cursors

Oracle 7.2 introduce ref cursors.

Ref cursors are user-defined types which is used to process multiple records and also this is a record by record process.

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

WEDNESDAY

MAY 12

23

144-222 WK-21

Generally through the static cursors we are using only one select statement at a time for single active set area whereas in ref cursors we are executing no. of select statements dynamically for a single active set area.

That's why these type of cursors are also called as dynamic cursors.

This is an user-defined type so we are creating in two step process i.e first we are creating type & then only we are creating a variable of ^{that} datatype. That's why these type of ~~vari~~ cursors are also called as cursor variables.

Generally we are not allowed to pass static cursors as parameters to the sub-programs.

To overcome this problem ANSI / ISO SQL introduce ref cursors. i.e ref cursors are used to pass parameters to the subprograms because these cursors basically user-defined types.

There are two types of ref cursors supported by oracle

- (i) Strong ref cursor
- (ii) Weak ref cursor

Strong ref cursor is a ref cursor

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

24

145-221 WK-21

which having a return type whereas weak ref cursor is ref cursor which does not have a return type.

Syntax: Type Typename is ref cursor return recordtype datatype ;

variablename Typename ;

↳ strong ref cursor variable

Syntax: Type Typename is ref cursor;

variablename Typename ;

↳ weak ref cursor variable

Note

In ref cursors we are executing select statements using open...for statement.

These statements are used in executable section of the pl/sql block.

Syntax: open refcursorvar for select statement;

06-Sep

Ex: declare

type t1 is ref cursor;

v_t t1;

i emp%rowtype;

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

FRIDAY

MAY 12

25

146-220 WK-21

```
begin
  open v_t for select * from emp
  where sal > 2000;
  loop
    fetch v_t into i;
    exit when v_t % not found;
    dbms_output.put_line (i.ename || '||' || i.sal);
  end loop;
  close v_t;
end;
/
```

? write a pl/sql program using ref cursor whenever user get the deptno display 10th department details from Emp table & also whenever user enter deptno 20 then display 20th department details from dept table.

SOLⁿ declare
type t1 is ref cursor;
v_t t1;
i emp%.rowtype;
j dept%.rowtype;
v_deptno number(10) := &deptno;
begin
if v_deptno = 10 then
open v_t for select * from emp where

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

MAY 26

26

147-219 WK-21

```

deptno= 10 ;
loop
fetch v_t into i ;
exit when v_t%notfound ;
dbms_output.put_line (i.ename || ' ' || i.deptno) ;
end loop;
close v_t ;
elsif v_deptno=20 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) ;
end loop;
close v_t ;
end ;
/

```

Passing ref cursor as parameter to the
Subprogram

sql> Create or replace package Pg1

is

```

type t1 is ref cursor return emp%rowtype;
type t2 is ref cursor return dept%rowtype;
procedure p1( p_t1 out t1);
procedure p2( p_t2 out t2);

```

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SUNDAY

MAY 27

27

148-218 WK-21

end;

/

sql> create or replace package body pg1

is

procedure p1 (p-t1 out t1)

is

begin

open p-t1 for select * from emp;

end p1;

procedure p2 (p-t2 out t2)

is

begin

open p-t2 for select * from dept;

end p2;

end;

/

execution

sql> variable a refcursor;

sql> variable b refcursor;

sql> exec pg1.p1 (:a);

sql> exec pg1.p2 (:b);

sql> print a b;

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

MONDAY

MAY 28

28

149-217 WK-22

Note

We are not allow to use ref cursors directly in packages.

Note

In place of weak ref cursor oracle give introduce sys-ref cursor predefined type.

Syntax: variable sys-refcursor;

ex: declare

v_t sys-refcursor;

i emp%rowtype;

begin

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.deptno);

end loop;

close v_t ;

end ;

/

ex: Create or replace function f1 (a varchar2)

return sys-refcursor

is

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

TUESDAY

MAY 29

29

150-216 WK-22

v-t sys_refcursor;

begin

open v-t for a;

return v-t;

end;

execution

sql> select f1 ('select * from emp where
sal > 2000') from dual;

3. Nested tables, varray

Oracle 8.0 introduce nested tables, varrays.

These are user defined types which is used to store multiple data items in a single unit but before we are storing actual data we must initialize using constructor.

Here constructor name is same as typename. Generally we are not allow to store index by tables permanently into database, to overcome this problem Oracle 8.0 introduce extension of the index by tables & these user defined types stored permanently into database using Sql.

In index by tables we cannot add a remote indexes whereas in nested tables, varray we can add a remote indexes using extend, trim collection methods.

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

WEDNESDAY

MAY 12

30

151-215 WK-22

Nested Table

Nested table basically is an unconstrained table and also this is an user defined type which stores number of data items.

Syntax: Type Typename is Table of datatype(size);
 Variablename Typename := Typename();
 ↳ constructorname

In nested tables, varrays always indexes start with 1 onward, & also these indexes are consecutive.

Nested tables are dense, whereas index by tables are basically no need to allocate the memory explicitly.

ex: declare

```
type t1 is table of number(10);
```

```
v_t t1 := t1();
```

```
begin
```

```
v_t.extend(500);
```

```
v_t(500) := 10;
```

```
dbms_output.put_line(v_t(500));
```

```
end;
```

```
/
```

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

THURSDAY

MAY-12

31

152-214 WK-22

ex: declare

type t1 is table of number(10);

v_t t1 := t1(10, 20, 30, 40, 50);

begin

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(3));

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;

end;

/

07-Sep-12

ex: 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

May	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

FRIDAY

JUN 12

1

153-213 WK-22

dbms_output.put_line ('v_t2 is null');
else

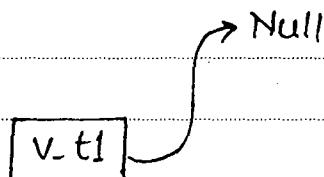
dbms_output.put_line ('v_t2 is not null');
end if;
end;

(output) \Rightarrow v_t1 is null

v_t2 is not null

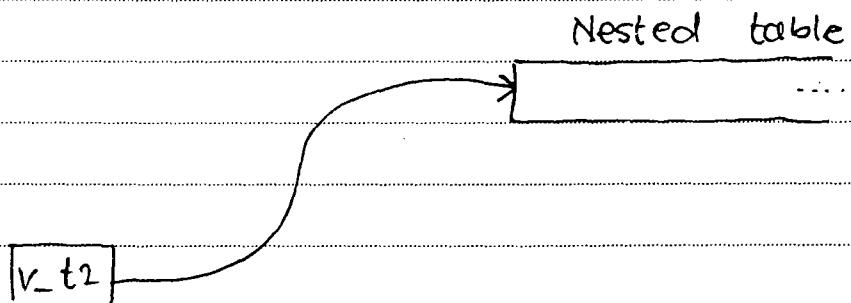
i] Type t1 is table of number(10);

v_t1 t1;



ii] Type t1 is table of number(10);

v_t2



iii] declare

Type t1 is table of number(10);

v_t t1 := t1();

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SATURDAY

JUN-12

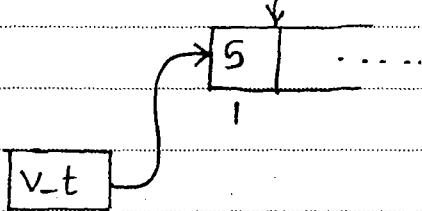
2

154-212 WK-22

```

begin
    vt extend;
    vt(1) := 5;
    dbms_output.put_line ( vt(1));
end;

```



? Write a pl/sql program to transfer all employee names from emp table & store it into nested table & also display content from nested table.

Soln sql> declare

```
type t1 is table of varchar2(10);
```

```
vt t1 := t1();
```

```
cursor c1 is select ename from emp;
```

```
n number(10) := 1;
```

```
begin
```

```
for i in c1
```

```
loop
```

```
vt.extend();
```

```
vt(n) := i.ename;
```

```
n := n+1;
```

```
end loop;
```

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

JUN 12

3

155-211 WK-22

```
for i in v-t.first..v-t.last
loop
```

```
dbms_output.put_line (v-t(i));
```

```
end loop;
```

```
end;
```

```
/
```

- (or) -

```
sql> declare
```

```
type t1 is table of varchar2(10);
```

```
v-t t1 := 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;
```

```
/
```

Varray

Oracle 8.0 introduced varrays.

It is an user-defined datatype which is used to store number of dataitems in a single unit.

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

MONDAY

JUN 12

4

156-210 WK-23

It is also same as arrays in normal language.

Here always index start with 1 and also indexes are consecutive.

We can also store varrays permanently into database using sql, this is also same as nested table i.e before we are storing actual data we must use constructor.

Syntax : ① Type typename is varray(maxsize) of datatype (size);
 ② variablename typename := typename();

ex: declare

```

type t1 is varray(4) of number(10);
v_t t1 := t1 ('a', 'b', 'c', 'd');
z boolean;
begin
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);
for i in v_t.first..v_t.last
loop
dbms_output.put_line (v_t(i));
end loop;
z := v_t.exists(3);
  
```

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

JUN 12

5

157-209 WK-23

```
if z=true then
    dbms_output.put_line ('ux index exists
    with an element'||'|| v_t(3));
else
    dbms_output.put_line (' index 3 does
    not exists');
end if;
v_t.extend;
v_t(5):='e';
v_t.extend(2);
v_t(6):='f';
v_t(7):='g';
v_t.extend(3,2);
for i in v_t.first.. v_t.last
loop
    dbms_output.put_line (v_t(i));
end loop;
v_t.trim(5);
dbms_output.put_line (v_t.count);
v_t.delete;
dbms_output.put_line (v_t.count);
end;
```

- /

? write a pl/sql program to transfer first
10 ename from emp table & store it
into varray & also display content from
varray.

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23



6

158-208 WK-23

Soln

declare

type t1 is varray(10) of varchar2(10);

v-t t1 := t1();

cursor c1 is select ename from emp where,

n number(10):=1;

rownum <=10;

begin

for i in c1

loop

v-t.extend();

v-t(n) := i.ename

n:=n+1;

end loop;

for i in v-t.first..v-t.last

loop

dbms_output.put_line(v-t(i));

end loop;

End;

/

- 10 Sep -

Difference betⁿ index by table, nested table, varray

Index by table	Nested table	Varray
This is an unconstraint table.	This is an unconstraint table.	This is an unconstraint table.
It is not	It is stored	It is stored

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

THURSDAY

JUN-12

7

159-207 WK-23

Stored
perminently
in

database

Stored
perminently
in

database
using SQL.

Stored
perminently
in

database
using SQL.

we can not add
or remove
indexes.

we can add or
remove indexes
using extends,
trim collection
method.

we can add or
remove indexes
using extends,
trim collection
method.

Indexes starting
from negative
to positive
numbers and
also having key
value pairs.

Indexes
starting from
1.

Indexes
starting from
1.

It supports exists,
first, last, count,
prior, Next, delete,
delete (range of
indexes)
collection methods.

It supports exists,
extends, trim,
first, last, prior,
next, count, delete,
delete (range of
indexes)
Collection methods

It supports
exists, limit,
extends, trim,
first, last,
prior, next
count, delete
Collection
methods.

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

FRIDAY

JUN 12

8

160-206 WK-23

Bulk bind

whenever we are submitting pl/sql block always sql statements are executed in sql engine and always procedure statements are executed in procedure statement executor.

Whenever pl/sql block contain large amount of coding & also sql & procedure statements are used frequently then internally oracle server control transfer betn sql engine, pl/sql engine no. of times. These type of execution methods are also called as context switching execution method.

Always context switching execution methods degrades performance of the app? To improve the performance of the application oracle introduce bulk bind process using collection i.e. in this process we are putting all sql statement related values into collection & in this collection we are performing insert, update, delete at a time using for all statement

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

SATURDAY

JUN 12

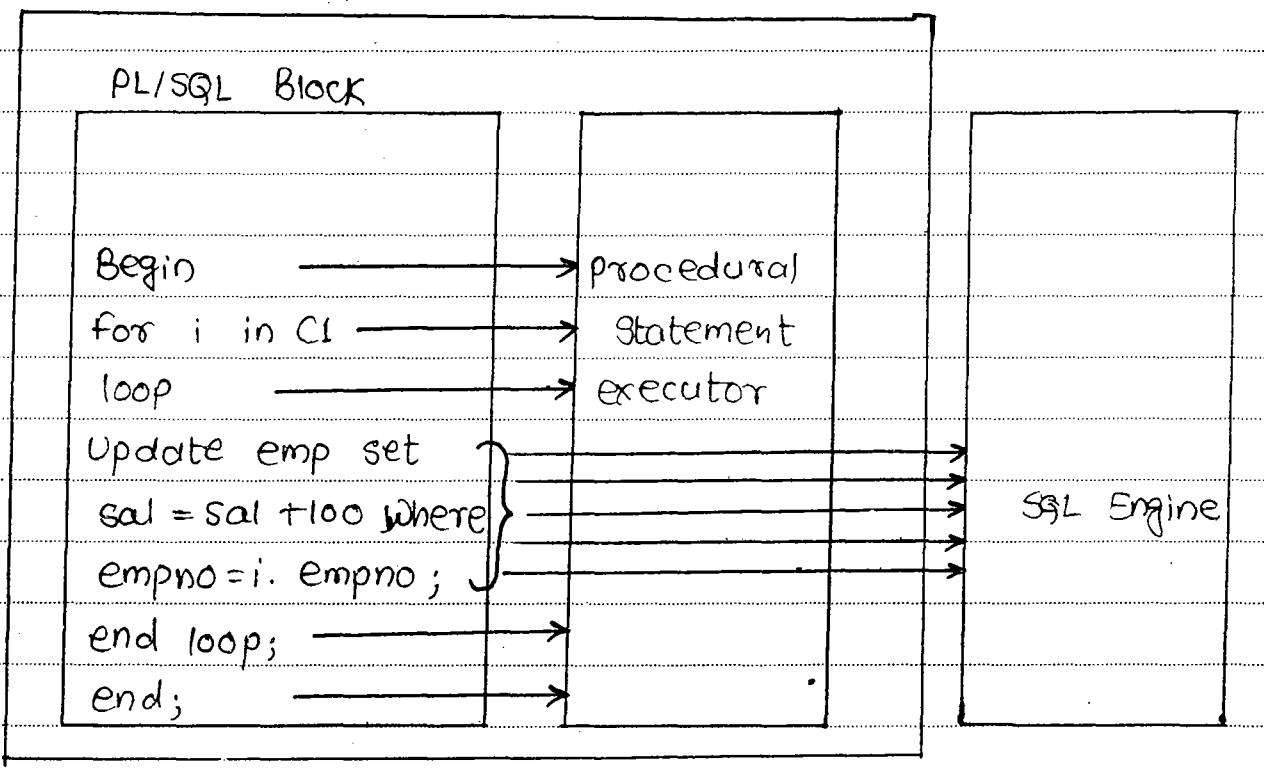
9

161-205 WK-23

Without bulk bind

oracle server

PL/SQL Runtime Engine



June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	24	25	26	27	28	29	30																	

SUNDAY

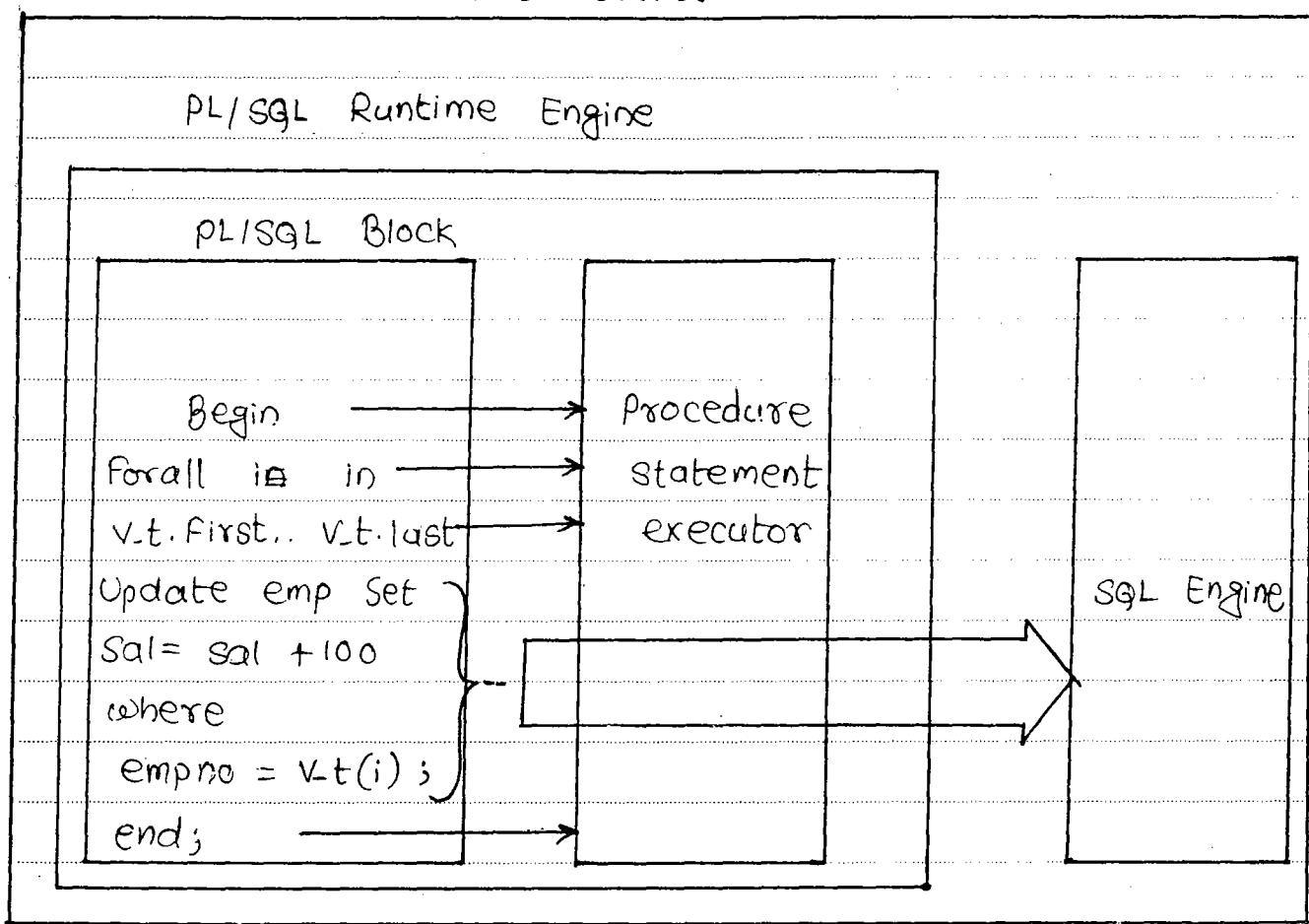
JUN 12

10

162-204 WK-23

with bulk bind

oracle Server



July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

MONDAY

JUN 12

11

163-203 WK-24

Before we are perform bulk of dml operation using select clause we must fetch the data from the resource into collection we are using bulk collect clause i.e. before bulk bind process we must use bulk collect clause.

Bulk Collect

This clause is used to fetch data from resource into collection.

This clause used in

- (i) Select into ... clause
- (ii) cursor fetch statement
- (iii) dml.... returning.... clauses

(i) bulk collect used in select....into clause

Syntax: Select * bulk collect into Collection var
from tablename where condition;

ex: declare

type t1 is table of emp%rowtype
index by binary_integer;

vt t1;

begin

Select * bulk collect into vt from emp;

for i in vt.first.. vt.last

loop

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

TUESDAY

JUN 12

12

164-202 WK-24

```

dbms_output.put_line (v-t(i).ename);
end loop;
end;
/

```

(ii) bulk collect used in cursor... fetch... statement

Syntax: fetch cursorname bulk collect into
collectionvar [limit variablename]

ex: declare

```

type t1 is table of varchar2(10) index
by binary_integer;

```

```
v_t1 t1;
```

```
v_t2 t1;
```

```
cursor c1 is select ename, job from emp;
```

begin

```
open c1;
```

```
fetch c1 bulk collect into v_t1, v_t2;
```

```
close c1;
```

```
for i in v_t1.first .. v_t1.last
```

loop

```
dbms_output.put_line (v_t1(i)||' '|| v_t2(i));
```

```
end loop;
```

```
end;
```

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

WEDNESDAY

JUN 12

13

165-201 WK-24

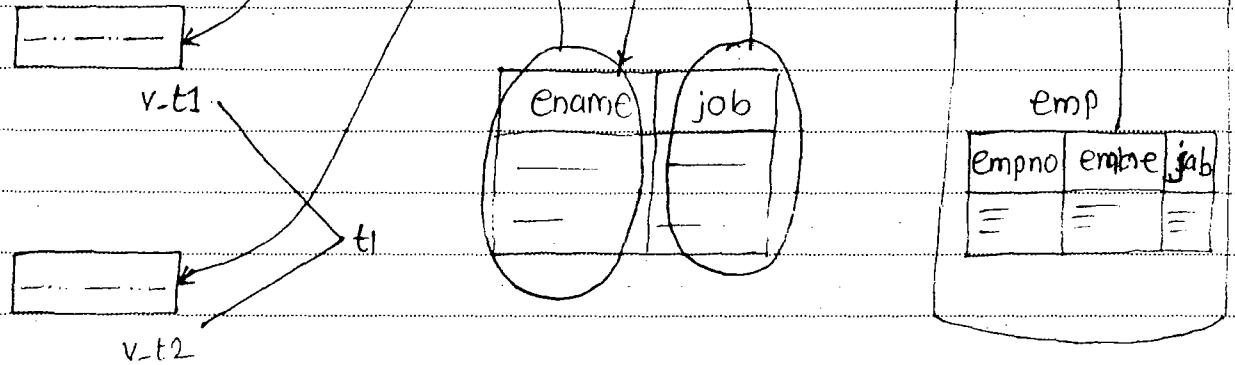
Fetch C1 bulk
Collect t
into
v_t1, v_t2;

cursor c1 is select

ename, job from
emp;
Open C1;

oracle d/b

emp		
empno	ename	job
1	SCOTT	ANALYST



- 11 Sep -

Ex: declare

type t1 is table of all_objects.OBJECT_NAME%type
index by binary_integer;

v_t1 t1;

v_t2 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;

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

THURSDAY

JUN 14

14

166-200 WK-24

```

z2 := dbms_utility.get_time;
dbms_output.put_line ('elapsed time for
normal fetch' || (z2-z1) || ' ' || 'sec');
z1 := dbms_utility.get_time;
open c1;
fetch c1 bulk collect into v_t1, v_t2;
close c1;
z2 := dbms_utility.get_time;
dbms_output.put_line ('elapsed time for bulk
fetch' || ' ' || 'hsec') (z2-z1) || ' ' || 'hsec');
end;
/

```

(iii) bulk collect used in dml ---- returning clauses

Returning clauses used in dml statements only.
These clauses are used to return values from
table after processing & store it into variables.

Syntax: dml statement returning columnname into
variablename;

ex: variable z varchar2(10);

sql> update emp set sal = sal + 100 where
ename = 'KING' returning job into :z ;

Sql> print z;

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

FRIDAY

JUNE 12

15

(output) z

PRESIDENT

167-199 WK-24

when we are processing bulk of data using implicit cursor we can also use bulk collect clause in dml-- returning clauses.

? Write a pl/sql stored procedure, modify salaries of the clerk from emp table & also these modified value immediately stored into index by table using dml-- returning clause & also display content from index by table.

SOL

```
ex: create or replace procedure p1
is
type t1 is table of emp%rowtype
index by binary_integer;
v_t t1;
begin
update emp set sal=sal+100 where
job = 'clerk' returning empno, ename, job,
mgr, hiredate, sal, comm, deptno bulk
collect into v_t;
dbms_output.put_line ('affected no. of clerks
are: '||' '|| sql%rowcount);
for i in v_t.first .. v_t.last
loop
```

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SATURDAY

JUN 12

16

168-198 WK-24

```

dbms_output.put_line ( v_t(i).ename || ' '
                      v_t(i).sal || '||' v_t(i).job );
end loop;
end;
/

```

sql > exec p1;

Bulk Bind

In bulk bind process we are performing bulk of operations using collection i.e in this process we are using bulk update, bulk delete, bulk insert using forall statement.

Before we are using bulk bind process we are fetching data from d/b into collⁿ using bulk collect clause.

Syntax: forall indexvar in collectionvar first ..
 collectionvar.last
 update tablename set columnname = newvalue
 where columnname = collectionvar (indexvar);

Ex: declare

type t1 is varray(10) of number(10);

v-t

t1:= t1 (20, 30, 40, 50, 60, 70, 80);

begin

forall i in v-t.first .. v-t.last

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

JUN 12

17

update emp set sal = sal + 100 where
deptno = v_t(i);
end;
/

? write a pl/sql prog to modify salaries of all employees from emp table using bulk bind process in this case first we are fetching empno from emp table & store it into index by table & then only we are using bulk bind process.

Sol?
declare
type t1 is table of emp%rowtype
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;
/

ex: declare

type t1 is table of emp%rowtype
index by binary_integer;

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	♦
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

MONDAY

JULY 17

18

170-196 WK-25

```

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;
/

```

error - element at index [3] does not exist

Whenever collections having gaps then we are not allow to use bulk bind process.

To overcome this problem oracle 10g introduce indices of clause in bulk bind Process.

indices of clause internally works based on array indexes.

Syntax: forall indexvar in indices of collectionvar
 update tablename set columnname = new value
 where columnname = collectionvar (indexvar);

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

TUESDAY

JUN 12

19

171-195 WK-25

Collection var

first

7566	
7839	
7492	
7902	
	Last

update all rows based on
collection method

Oracle 10g (indices of)

first

first

update all
rows based on
any
indexes

7566	
	indices
7839	$v-t(1) := 7566$
7492	$v-t(2) := 7839$
7902	$v-t(3) := 7492$
	$v-t(4) := 7902$
	Last

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

WEDNESDAY

JUN 12

20

172-194 WK-25

```

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%bulk_rowcount

- 12 Sep -

oracle introduce sql%bulk_rowcount attributes to interact number of rows in each iteration in bulk bind process.

syntax: sql%bulk_rowcount (indexvariablename)

ex: declare

```

type t1 is varray(10) of number(10);
v_t t1 := t1 (20, 30, 50, 70);
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

```

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

JUN 12

21

173-193 WK-25

loop
dbms_output.put_line ('affected number
of rows in: ' || v-t(i) || ' are ' ||
' || sql%rowcount (i));
end loop;
end;
/

bulk delete

ex: declare

type t1 is varray(10) of number(10);
v-t t1 := t1 (20, 30, 40, 50, 60, 70);
begin
forall i in v-t.first.. v-t.last
delete from emp where ~~v-t~~ empno = v-t(i);
end;
/

bulk insert

ex: sql> create table test (sno number(10));

sql> declare

type t1 is table of number(10)
index by binary_integer;
v-t t1;

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

FRIDAY

JUN 22

22

174-192 WK-25

```

begin
  for i in 1..5000
    loop
      v_t(i) := i ;
    end loop;
  
```

```

forall i in v_t.first .. v_t.last
  insert into test values (v_t(i));
end;
/ 
```

Sql> select * from test;

dbms_utility package

This package used by either database administrator or database developers.

This package having get_time method which returns elapsed time.

This method returns always number datatype.

This method is used by database administrator to calculate elapsed time.

Syntax: variablename := dbms_utility.glt_time ;

This package used by developers because this package internally having index by table.

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

JUN 12

23

175-191 WK-25

This package also having two methods

- i) Comma_to_table
- ii) table_to_comma

These methods are used to transfer pl/sql values into comma separated strings & comma separated strings into pl/sql table values. Before we are using these methods we must create an index by table variable using ucl_array type from dbms_utility package.

Syntax: variablename dbms_utility.ucl_array;

- i) Comma_to_table

This method is used to convert comma separated strings into index by table values.

Syntax: dbms_utility.comma_to_table (stringname,
binary_integer variablename,
index_by_table variable);

Ex: declare

v_t dbms_utility.ucl_array;

z binary_integer;

str varchar2(200);

begin

str := 'a, b, c, d, e, f';

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SUNDAY

JULY 24

24

176-190 WK-25

```
dbms_utility.comma_to_table(str, z, vt);
```

```
for i in vt.first .. vt.last
```

```
loop
```

```
dbms_output.put_line(vt(i));
```

```
end loop;
```

```
end;
```

```
/
```

ii) table_to_comma

This method is used to convert index by table values into comma separated string.

Syntax: dbms_utility.table_to_comma(index_by_table var
iable, binary_integer variable, stringname);

? write a pl/sql program using dbms_utility package, display all dept-names from dept table into separate string.

SOLN declare

```
v_t dbms_utility.uncl_array;
```

```
z binary_integer;
```

```
str varchar2(200);
```

```
begin
```

```
select dname, bulk collect into v_t from dept;
```

```
dbms_utility.table_to_comma(v_t, z, str);
```

```
dbms_output.put_line(str);
```

```
end;
```

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

MONDAY

JUN 12

25

177-189 WK-26

LLOBJS (Large objects)

Oracle 8.0 introduced large objects.

If we want to store more than 4000 bytes of alphanumeric data, we are using long datatype.

This datatype stores upto 2GB data but there can be only one long column for a table & also we can not create primary key on that column.

To overcome these problems Oracle 8.0 introduces CLOB datatype.

Syntax: Columnname long

Syntax: Columnname clob

Ex: sql> create table k1(col1 long);

If we want to store binary data we are using RAW datatype, but raw support upto 2000 bytes binary data.

If you want to store more than 2000 bytes of binary data we are using long raw datatype.

Syntax: columnname raw (maxsize):

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

JUN 26

26

Syntax: columnname long raw;

ex: sql> create table k2 (col1 raw(100)); 178-188 WK-26

ex: sql> create table k3 (col1 long raw);

If we want to store more than 2000 bytes of binary data we are using blob datatype

Syntax: Columnname blob

There are two types of large objects supported by oracle.

1. Internal Large Objects
2. External Large Objects

Internal large objects are stored within database
There are two types of internal large objects supported by oracle

(a) clob (character large object)

(b) blob (binary large object)

External large objects are stored outside of the database.

This is a b-file datatype

before we are handling large object we must create an alias directory related to physical directory using

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

WEDNESDAY

JUN 12

27

179-187 WK-26

following Syntax

Syntax: Create or replace directory
directoryname as 'path of physicalfile';

Before we are creating alise directory admin user must give create any directory privilege using following syntax

Syntax: grant create any directory to
username1, username2, ...

ex: Sql> conn sys as sysdba;

Enter password: sys

sql> grant create any directory to scott;

sql> conn scott/tiger;

sql> create or replace directory zzz as 'E:\';

Storing large amount of data or in image into database

Step 1 :- Create a table in Oracle database.

Step 2 :- develop a pl/sql block using dbms_lob package

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

THURSDAY

JUN 28

28

180-186 WK-26

Step-i) before we are storing actual data into lob table we must initialize using empty_clob() or empty_blob() Functions & also using returning clauses we are returning point to appropriate lob column.

Syntax: Insert into tablename values (empty_clob()) returning columnname into variablename;

Step-ii) we must concatenate actual file with aliased directory using bfilename function.

This fun also returns bfile datatype.

Syntax: variablename := bfilename ('aliasedirectory', 'actualfilename');

Step 3 :- Using load from file method from dbms_lob package we are loading large amount of data into appropriate column.

Syntax: dbms_lob.loadfromfile (clobvariable, bfilevar, length OF bfilevar);

Step 4 :- Before we are loading data, we must open the file using fileopen method from dbms_lob package. And also after processing close the file using fileclose method from

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

FRIDAY

JUN 12

29

dbms_lob package.

181-185 WK-26

Syntax: dbms_lob.Fileopen (bFilevar);

dbms_lob.fileclose (bFilevar);

SQL> Create table test (col1 number (10), col2 clob);

Ex: declare

v_clob clob;

v_bfile bfile;

begin

insert into test values (1, empty_clob())

returning col2 into v_clob;

v_bfile := bfilename ('zzz', 'first.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;

UTL_file package

Oracle 7.3 introduce utl_file package.

This package used to write a data into a file & read data from a file.

Before we are performing operations admin user must give read, write privileges on alise

June	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

SATURDAY

JUN 19

30

182-184 WK-26

directory using following syntax

syntax: grant read, write on directory
directoryname to username;

ex: sql> conn sys as sysdba;
Enter password: sys

sql> grant read, write on directory zzz to
scott;

Writing data into a file

Step-1) Before we are performing file operations
we must create a file pointer opⁿ
from file-type in utl-file package.

syntax: filepointervar UTL-file. file-type;

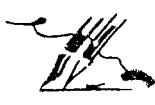
Step-1) We must open the file before we are
performing read, write operation using
fopen method from utl-file package.

This method accepts three parameters & also
return file-type datatype.

syntax: filepointervar := utl_file.fopen ('alisedirectory
name', 'filename', 'mode');

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Notes



Step-III) Using putf method from utl_file package we are storing actual data into a file.

Syntax: utl_file.putf (filepointervar, 'actualdata');

Step-IV) After processing file operations we must close the file using fclose method from utl_file package.

Syntax: utl_file fclose (variablename);

ex: declare

```
fp utl_file.file_type;
begin
fp:=utl_file.fopen ('zzz', 'fan.txt', 'w');
utl_file.putf (fp, 'tomorrow class shifted
into block3');
utl_file	fclose (fp);
end;
```

/

? write a pl/sql program using utl_file package store all employee names into an external file using emp table.

declare

```
fp utl_file.file_type;
```

WK

ACTION PLAN

JULY '12

Mon	Tue	Wed	Thu	Fri	Sat	Sun
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

OBJECTIVES THIS MONTH

DATE

DESCRIPTION

REMARKS

JUNE 12

2012

SUNDAY

JUL '12

1

183-183 WK-26

cursor c1 is select ename from emp;
begin
fp := utl_file.fopen ('zzz', 'fan.txt', 'w');
for i in c1 %
loop
utl_file.putf (fp, i.ename);
end loop;
utl_file.fclose (fp);
end;
/

Reading data from file

If we want to read data from a file we are using get-line() from utl-file package

Syntax: utl_file.get_line (filepointervariable,
buffervariable);

ex: declare

fp utl_file.file-type;
x varchar (200);
begin
fp := utl_file.fopen ('zzz', 'first.txt', 'r');
utl_file.get_line (fp, x);
dbms_output.put_line ('data from file' || ' ' || x);
utl_file.fclose (fp);
end;

/

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

SQL Loader

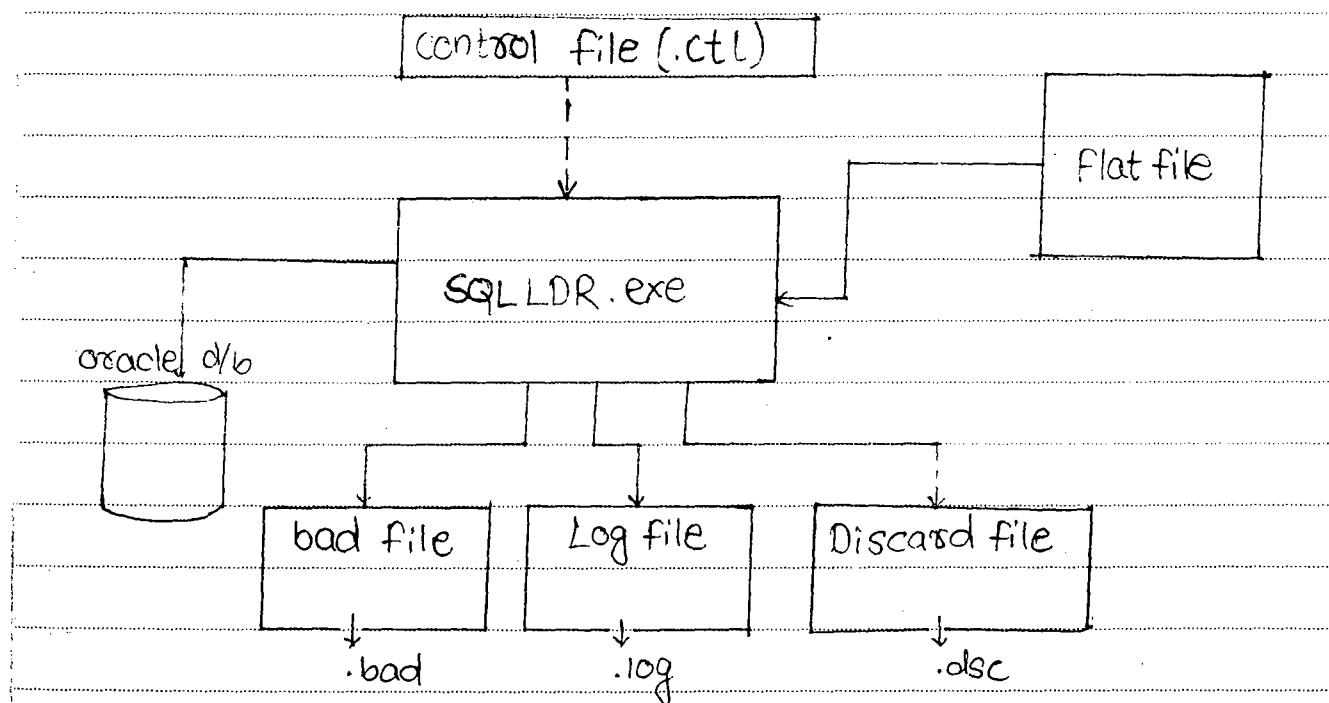
2

SQL Loader is an utility program which 184-182 WK-27 is used to transfer data from flat file & store it into oracle database.

SQL Loader is also called as Bulk Loader.

Always SQL Loader executes control file based on the type of flat file we are creating control file & submit to the sql loader then only sql loader fetch data from flat file & store it into target table. During this process sql loader generates following types of files

1. Logfile (.log)
2. Bad file (.bad)
3. Discardfile (.dsc)



August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

TUESDAY

JUL 12

3

185-181 WK-27

Log file stores all other files information and also stores loaded, skipped, rejected record numbers and also stores elapsed time.

Bad, discard file stores rejected records.

Flat File

Flat file is a structured file, which contains data.

There are two types of flat files supported by all system.

(i) variable Record FlatFile -

(ii) Fixed Record Flatfile

A flatfile which contains delimiters is called variable record flat file.

Ex: 101, abc, 2000

102, xyz, 3000

A flatfile which does not contain delimiters is called fixed record flat file.

Ex: 101 abc 2000

102 xyz 3000

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

WEDNESDAY

JULY 12

Control file

This file extension is .ctl,
based on the type of flat file we
are generating control file

186-180 WK-27

4

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 in file clause.

Syntax: load data
infile 'path of flatfile'

Note

If control file having a flat file then we
must use * in place of flat file path in
in file clause & also use begin data clause
above of the flat file

ex: load data
infile *

begin data

101, abc, 2000

102, xyz, 4000

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

JUL 12

5

187-179 WK-27

Using into table tablename clauses we are loading data into target table.

Before into table tablename clauses we are using either insert or truncate or replace or append clauses.

If target table is an empty table then only we are using insert and also by default clause is insert.

After into table tablename clauses we are using following clauses

a) fields terminated by 'delimetername'

b) optionally enclosed by 'delimetername'

After these clauses we must specify target table column within parenthesis.

load data

infile 'path of flatfile'

badfile 'path of badfile'

discardfile 'path of discardfile'

insert/ truncate/ replace / append into table

tablename

fields terminated by 'delimetername'

optionally enclosed by 'delimetername'

(col1, col2, ... coln)

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu								
2012		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

FRIDAY

JUL 12

6

invoking sqlloader

d:\>sqlldr userid = scott/tiger
control= ---- filename.ctl

188-178 WK-27

101, abc, 2000

102, xyz, 3000

sql> create table target (empno number(10),
ename varchar2(10), sal number(10));

load data

infile 'E:\One.txt'

insert into table target fields terminated
by ',' (ename,empno, ename , sal)

D:\>sqlldr userid=scott/tiger

control= E:\murali.ctl

Storing Default Values

Using constant clause we are storing default values
into database.

When flat file having less no. of fields &
target table having more no. of Fields then
Only we are using constant clause.

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

SATURDAY

JUL 12

7

Syntax: columnname constant 'actualvalue'

189-177 WK-27

If we want to skip no. of columns within flatfile then we are using filler clause, i.e if flat file having more no. of field & target table having less no. of columns then we are using filler clause.

ex. load data

infile 'E:\one.txt'

insert

into table target fields terminated by ''
(empno, ename, loc constant 'ammerpet')

ex: load data

infile 'E:\one.txt'

insert

into table target fields terminated by ''
(empno, ename filler, sal)

Bad files stores rejected records that cause an error based on following condition

- a) Data type mismatch
- b) Business rule violation

This file extension is .bad.

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

we can also create bad file explicitly
 using bad file clause & also we are
 specifying ↓ .xls or .doc .
 badfile extension

8

190-176 WK-27

a) Data type mismatch

101, abc, 2000
 '102', xyz, 3000
 '103', pqr, 4000
 104, zzz, 5000

Sql> create table target (empno number(10),
 ename varchar2(10), sal number(10));

load data infile 'E:\one.txt'
 insert into target ↓ fields terminated by ','
 table
 (empno, ename, sal)

'102', xyz, 3000
 '103', pqr, 4000

Note

when we are transferring data from flat file
 into oracle d/b using sql loader automatically
 log file is created as same name as control
 filename & also bad file is created as same

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

MONDAY

JUL 12

9

name as flatbadfile filename

191-175 WK-28

b) Business Rules Violation

ex: 101, abc, 7000

102, xyz, 3000

103, zzz, 4000

104, ggg, 8000

sql> create table target (empno number(10),
ename varchar2(10), sal number(10)
check (sal > 6000));

102, xyz, 3000

103, zzz, 4000

RECNUM

This clause is used to assign no. to records in d/b table.

This clause internally assign no. to loaded, Skipped, rejected records.

Syntax: Columnname recnum

ex: 101, abc, 7000

'102' xyz, 3000

'103' zzz, 4000

104, ggg, 8000

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

JULY 12

sql> select * from target;

10

Dates used in Control file

192-174 WK-28

- i] Using date datatype
- ii] Using to_date() function

i] Using date datatype

Syntax: Columnname date " flat File date format"

ex: 101, abc, 09/05/03

102, xyz, 06/04/12

103, zzz, 07/08/11

sql> create table target (empno number(10),
ename varchar2(10), DOJ date);

load data infile 'E:\one.txt'
insert into table target fields terminated
by ',' (empno, ename, DOJ date
" DD/MM/YY").

Functions Using Control file

we can also use predefined, user defined
oracle fun within control file but in this case
we must specify fun functionality within double

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

WEDNESDAY

JUL '12

11

193-173 WK-28

quotes & also use a colon operator in front of the columnname in function functionality.

syntax: columnname "functionname (:columnname)"

ex: 101, abc, m
102, xyz, f
103, zzz, m

sql> create table target (empno number(10),
ename varchar2(10), gender varchar2(10));

load data infile 'E:\one.txt'
insert into table target fields
terminated by ';' (empno, ename, gender,
"decode (:gender, 'm', 'male', 'f', 'female')")

Discard File

Discard file stores rejected records that cause error
then we are using when clause

But discard file we must specify explicitly
using discard file clause.

This file extension is .dsc

ex: 101, abc, 10

102, xyz, 20

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

THURSDAY

JULY 12

103, zzz, 30

104, aaa, 10

12

194-172 WK-28

load data infile 'E:\one.txt'
 discard file 'E:\sunday.dsc'
 insert into table target when deptno='10'
 Fields terminated by ','
 (emno, ename, deptno)

sql> create table target (enamno number(10),
 ename varchar(10), deptno number(10));

102, xyz, 20

103, zzz, 30

Note

In when clause we must specify condition value within single quote.

In when clause we are using "=", "!=" relational operators only.

In when clause we can use only logical operator 'and' we are not allow to use logical operator 'or'.

Creating control file for fixed Record Flatfile

A flatfile which does not have delimeter is

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

FRIDAY

JULY 12

13

195-171 WK-28

called fixed record flat file.

When we are using fixed record flat file we must use position clause in control file.

In position clause we must specify starting, ending position of the every field within the flat file using color operator. Alongwith position clause we must specify sql loader datatypes.
SQL Loader supports 3 datatypes:

1. integer external
2. decimal external
3. Char

Syntax: Columnname position (startingposition: ending position) sqlloaderdatatype

Note

When we are using funs within control file then we are not allow to use sqlloader datatype in place of this one we are using funs functionality.

Syntax: columnname position (startingpos: endingpos)
" function (:columnname)"

ex: 101 abc 1000

102 xyz 2000

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

JUL 14

103 zzz 3000
104 aaa 1000

14

196-170 WK-28

sql> create table target (empno number(10),
ename varchar(10), sal number(10));

load data infile 'E:\one.txt'
insert into table target (empno
position (01:03) integer external,
ename position (04:06) char,
sal position (07:10) integer external)

Sequences Used in Control file

Syntax: columnname "sequencename.nextval"

ex: 101
102
103
104
105
106

sql> create table target (sno number(10));
sql> create sequence s1 start with 1;

load data infile 'E:\one.txt'
insert into table target (sno "s1.nextval")

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

JULY 12

15

Flat file used in control file self

197-169 WK-28

ex: `Sql> create table target (empno
number(10), ename varchar2(10),
sal number(10));`

load data infile * insert into table target

fields terminated by ','
(empno, ename, sal)

begindata

1021, abc, 1000

102, xyz, 2000

103, xxx, 5000

default 50

hours, mins

Load data

Target(table)

infile *
insert into
table target

sno	value	Time	col1	col2	col3

`Sql>`
`create Sequence S1 start
with 1 ;`

col1/100

1000aaaaaa06512

upper

BeginData

2000blabb 07902

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

MONDAY

JUL '12

16

```

( sno      "SI.nextval",
  value    constant'50',
  time     "to_char (sysdate , 'HH12:MI:SS')"198-168 WK-29
  col1     position (01:05)   ":col1/100"
  col2     position (06:13)   "upper (:col2)"
  col3     position (14:19)   "To_date (:col3 , 'DDMMYY')"
)

```

Note

When flatfile contains quotations then we are using optionally enclosed by clause within control file.

Using sql loader we can also transfer data from ~~one~~ one flat file into multiple target table using multiple into clauses. And also no. of flat files data transfer into single target table using multiple infile clause but using sql loader we can not transfer source as ~~one~~ flatfile, database source as different databases.

Nested Block

Block within another block is called nested block.

Generally Child block is also called as nested block. In nested block we can also use same variable name with parent block, child block.

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

JUL '12

17

Generally nested block executable

Statement access nested block variables.

199-167 WK-29

If we want to access parent block variable we must use parent block name within executable statement. Before that one must specify parent block name within double angular bracket.

ex. <<parent>>

declare

v_empno emp.empno%type :=&empno1;

begin

<< child>>

declare

v_empno emp.empno%type :=&empno2;

v_ename emp.empname%type;

begin

Select ename into v_ename from emp

where empno = parent.v_empno;

dbms_output.put_line (v_ename);

end;

end;

/

(output) enter value for empno1: 7902

enter value for empno2: 79566

FORD

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

When a pl/sql block contains no. of select-- into clauses & also if you want to display flexible messages than the generalized messages in exception handler we must use nested block.

When exceptions are raised in declare section, exception section those exceptions must be handled using outer block Only.

ex. declare

v_empno1 emp.empno%type := &empno1;

v_empno2 emp.empno%type := &empno2;

v_ename emp.ename%type;

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 (' your employee
does not exist+ with employee number ####'||
v_empno1);

end;

begin

Select ename into v_ename from emp
where empno = v_empno2;

18

200-166 WK-29

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

JUL 12

19

201-165 WK-29

```
dbms_output.put_line ('my second  
employee' || '' || v_empno3);  
exception  
when no_data_found then  
dbms_output.put_line ('your employee  
does not exist with empno' || '' ||  
v_empno2 );  
end;  
end;  
/
```

Note

Nested blocks also access parent block variables & their own variables.

Local Procedures, Local Functions

Local subprograms are used to solve some particular task.

These subprograms are not stored in database. These subprograms does not create or replace key words.

These subprograms are created in either anonymous or in nested blocks or in stored procedures.

Always these subprograms are defined in declare section of the pl/sql block, & also call these blocks in immediate executable section.

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

FRIDAY

JULY 12

20

202-164 WK-29

We must define local subprograms in
last of the declare section in pl/sql
block.

Syntax: declare
variable declarations;
cursors declarations;
types declarations;
procedure procedurename (formal
parameters)

is

begin

[exception]

end [procedurename];
function functionname (formal
parameters) return datatype

is

begin

return expr;

end [functionname];

begin

procedurename (actual parameters);

var:=functionname (actual parrams);

end;

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

SATURDAY

JUL '12

21

Passing PL/SQL Table As In Parameter Into Local Procedure

203-163 WK-29

ex. declare

```
type t1 is table of emp%rowtype  
index by binary_integer;
```

```
v_t t1;
```

```
procedure p1 (p_t in t1)
```

```
is
```

```
begin
```

```
for i in p_t.first..p_t.last
```

```
loop
```

```
dbms_output.put_line (p_t(i).ename);
```

```
end loop;
```

```
end p1;
```

```
begin
```

```
select * bulk collect into v_t from emp;
```

```
p1 (v_t);
```

```
end;
```

```
/
```

Passing PL/SQL Table As Out

Parameters Into Local procedure

ex. declare

```
type t1 is table of emp%rowtype
```

```
index by binary_integer;
```

```
v_t t1;
```

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SUNDAY

JUL 22

22

204-162 WK-29

```
procedure p1 (pt out t1)
is
begin
select * bulk collect into pt from emp;
end p1;
begin
p1 (v-t);
for i in v-t.first.. v-t.last
loop
dbms_output.put_line ( v-t(i).ename);
end loop;
end;
/
```

Return PL/SQL Table As Local Function

Return Type

Ex: declare

```
type t1 is table of emp%rowtype
index by binary_integer;
procedure p1 (p-t t1)
is
begin
for i in p-t.first.. p-t.last
loop
dbms_output.put_line ( p-t(i).ename);
end loop;
```

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

MONDAY

JUL 12

23

205-161 WK-30

```
function f1 return t1
is
v_t t1;
begin
Select * bulk collect into v_t
from emp;
return v_t;
end f1;
begin
p1(f1);
end;
/
```

Dynamic SQL

Oracle 7.1 introduced dynamic SQL.

It is the combination of SQL, PL/SQL i.e SQL statements are executed dynamically with PL/SQL block using execute immediate clause.

Generally in PL/SQL block we are not allow to use DDL, DCL statements. Using dynamic SQL DDL, DCL statements within PL/SQL block.

Syntax:

```
begin
execute immediate 'Sql statement';
end;
/
```

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

JULY 22

24

206-160 WK-30

ex: begin
execute immediate ' create table
x1 (sno number(10))';
end;
/

Passing Values Into Dynamic SQL Statement

through using clause we are passing values into dynamic SQL statement.

Here we are passing values either statically or dynamically using placeholders.

In oracle place holders are represented using colon operator.

? Write a dynamic SQL program to insert a record into dept table.

SQL> declare
v_deptno number(10):= 5 ;
v_dname varchar2(10) := 'a' ;
v_loc varchar2(10) := 'b' ;
begin
execute immediate ' insert into dept values
(:1, :2, :3)' using v_deptno, v_dname, v_loc ;
end ;
/

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

WEDNESDAY

JUL 12

25

Retrieving Values From Dynamic SQL Statements

207-159 WK-30

through into clause we are retrieving values from dynamic statement.

? write a dynamic SQL prog to display no. of records no. from emp table.

Solⁿ declare
z number(10);
begin
execute immediate ' Select count(*) from emp' into z ;
dbms_output.put_line (z);
end;

? write a dynamic SQL prog to retrieve all employee names from emp table & store it into index by table & also display the content from index by table.

Solⁿ declare
type t1 is table of varchar2(10)
index by binary_integer;
v_t t1;
begin
execute immediate ' select ename from emp'

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

THURSDAY

JUL 12

26

208-158 WK-30

```
bulk collect into vt;
for i in vt.first..vt.last
loop
  dbms_output.put_line(vt(i));
end loop;
end;
```

Note

When a dynamic SQL statement contains using, into clauses always into clause precedes using clause.

? Write a dynamic SQL prog for passing deptno
to retrieve deptname, loc from dept table.

SOL? declare

v_deptno number(10):=20;

v_dname varchar2(10);

v_loc varchar2(10);

begin

execute immediate 'select dname, loc from dept where deptno:=1' into v_dname, v_loc
using v_deptno;

dbms_output.put_line (v_dname||' '||v_loc);

end;

/

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

FRIDAY

JUL 12

27

209-157 WK-30

Oracle Versions

Oracle 2.0 --> 1979

- > basic SQL functionality
- > joins

Oracle 3.0 --> 1983

- > Commit, rollback
- > rewritten in C language

Oracle 4.0 --> 1984

- > read Consistency

Oracle 5.0 --> 1986

- > Client-Server architecture

Oracle 6.0 --> 1988

- > PL/SQL introduced
- > row level locks

Oracle 7.0 --> 1992

- > datatype varchar changed into varchar2

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

SATURDAY

JUL 12

28

210-156 WK-30

- > integrity constraints
- > stored procedures, functions,
triggers;
- > truncate table
- > View Compilation

Oracle 7.1

- > ANSI / ISO SQL92
- > Dynamic SQL

Oracle 7.2

- > ref cursors (or) cursor variables
- > inline views
- > dbms-job package

Oracle 7.3

- > Bitmap indexes
- > utl-file package

Oracle 8.0 --> 1996

object technology
nested tables, varrays
large objects
columns increased per a table upto 1000

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

SUNDAY

JUL '12

29

Oracle 8i

211-155 WK-30

- > Analytical function
- > instead of triggers
- > materialized views
- > function based indexes

Oracle 9i --> 2001

- > rename a column
- > merge statements
- > ANSI joins
- > multitable inserts
- > flashback query

Oracle 10g --> 2005

- > rename tablespace
- > flashback table
- > recycle bin
- > wmi_concat()
- > indices of clause

Oracle 11g --> 2007

11g introduce read only tables along with alter command.

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

MONDAY

JULY 12

Syntax: alter table tablename read
Only;

30

212-154 WK-31

Syntax: alter table tablename read write;

Virtual column - Before 11g if you want to store stored expr in db we are using fun based indexes, view. But 11g introduce virtual column using generated always as clause directly storing stored expression within database.

Syntax: columnname datatype (size) generated
always as expression [virtual]

ex: sql> create table w1 (a number (10), b
number (10), c number (10) generated
always as (a+b) virtual);

sql> insert into w1 values (10, 20);

sql> select * from w1;

If we want to view virtual expression
we are using data_default property from
user_tab_columns datadictionary.

sql> desc user_tab_columns;

August	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TUESDAY

JUL '12

31

213-153 WK-31

sql> Select data_default from user_tab_column where tablename = 'W1';

Simple_integer data in pl/sql

This datatype performance is very high compare to binary_integer, pls_integer_datatype. This datatype can not accept null values.

ex: declare
a simple_integer := 50;
begin
dbms_output.put_line(a);
end;

/

Oracle 11g introduce continuous statement in pl/sql loops as same as c-language Continuous Statement.

Oracle 11g introduce follows clause in triggers.

11g introduce variable assignment concept when we are sequences in pl/sql blocks.

Oracle 11g introduce enable, disable keywords within trigger specification itself.

Oracle 11g introduce named, mixed notations in functions calling in select statements.

- mrvmsrinivas@gmail.com 9959430399

July	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu							
2012	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31