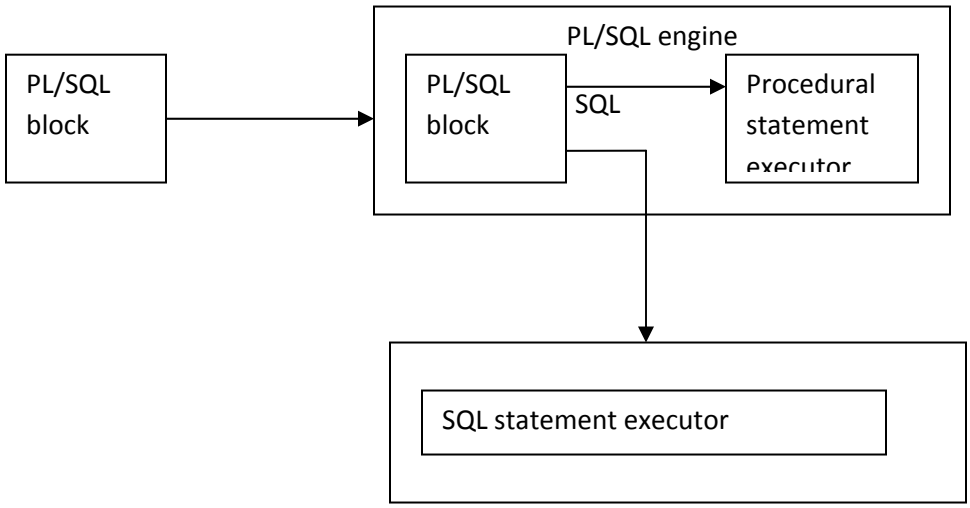
PL/SQL

Overview of PL/SQL

PL/SQL is the procedural extension to SQL with design features of programming languages. Data manipulation and query statements of SQL are included within procedural units of code.

Pl/SQL Environment



The PL/SQL engine in the oracle server process the pl/sql block and it separates SQL staments and sends them individually to the SQL statements executor

Benefits of PL/SQL

- Integration
- Improved performance
- Modularized program development
- Portability
- Identifiers

PL/SQL Block structure

DECLARE (optional)

Variables, cursors, user-defined exceptions

BEGIN (Mandatory)

-SQL statements

-PL/SQL statements

EXCEPTION (optional)

Action to perform when error occur

END;

The PL/SQL Block consists of three sections:

DECLARATIVE

It contains all variables, constants, cursors and user defined exceptions that are referenced in the executable and declarative sections.

EXECUTABLE

It contains SQL statements to manipulate data in the database and PL/SQL statements to manipulate data in the block.

EXCEPTION HANDLING

It specifies the actions to perform when errors and abnormal conditions arise in the executable section.

PL/SQL Block Types

A PL/SQL program comprises one or more blocks.

It is classified into two blocks

- **Anonymous Blocks**

It is unnamed blocks. It is declared at the point in an application where they are to be executed and are passed to the PL/SQL engine for execution at run time.

- **Subprograms**
  Subprograms are named PL/SQL blocks that can accept parameters and can be invoked. It can be declared either as procedures or as functions.

Sample PL/SQL programs

To write PL/SQL programs, create a script file and run the script file or use editor.

Steps to create script file

Step1:

SQL> edit z:\oracle\sql\var1.sql

Step2:

Type the program in notepad

Step 3:

Save the program

Step4:

Run the program

SQL> @z:\oracle\sql\var1.sql

SQL> **set serveroutput on;**

This command is used to display the statement executed by dbms_output.put_line package.

Program 1: Write a program to print a variable value.

```
SQL> declare
  2  a number:=3;
  3  begin
```

```
 4  dbms_output.put_line(a);
 5  end;
 6  /
   3
```

Program 2: Write a program to print your name and regno.

```
 1  declare
 2  v_name varchar2(10);
 3  v_regno number;
 4  begin
 5  v_name:='venkat';
 6  v_regno:=39;
 7  dbms_output.put_line( 'the name is' || v_name);
 8  dbms_output.put_line('the no is' || v_regno);
 9 end;

SQL> /
the name is venkat
        the no is 39

PL/SQL procedure successfully completed.
```

Program 3: Write a program to retrieve ssn number of employee whose name is x.

Assume the following table:

SSN	NAME	ESSN	DEPTNO	SALARY
101	x	102	1	
102	y	103	2	
103	z	102	3	
104	p	102	4	
105	q			

```
declare v_no number;
begin
select ssn into v_no from emp where name='x';
dbms_output.put_line(v_no);
end;
SQL>/
101
PL/SQL procedure successfully completed.
```

SCALAR VARIABLE

It holds a single value and has no internal components.

Examples : number, character, date, boolean

Example: using scalar variable

```
 1  declare
 2  v_name varchar2(10);
 3  V_count binary_integer:=10;
 4  V_totalsal number(9,2);
 5  v_orderdate date:=sysdate;
 6  c_tax constant number(3,2):=6.23;
 7  v_valid boolean not null:=true;
 8  v_regno number default 23;
 9  begin
10  v_name:='venkat';
11  v_totalsal:=10000.23;
12  dbms_output.put_line(v_name);
13  dbms_output.put_line(v_count);
14  dbms_output.put_line(v_orderdate);
15  dbms_output.put_line(c_tax);
16  dbms_output.put_line(v_regno);
17 end;
18  /
venkat
10
19-AUG-05
6.23
23
```

DECLARING VARIABLE WITH THE %TYPE ATTRIBUTE

The % type attribute is used to declare a variable according to:

1. A database column definition
2. Another previously declared variable

Example: using % type attribute

```
 1  declare
 2  v_no emp.ssn%type;
 3  V_name varchar2(10):='venkat';
 4  name  v_name%type;
 5  begin
 6  v_no:=10;
 7  name:='ven';
 8  dbms_output.put_line(v_no);
 9  dbms_output.put_line(name);
10*end;
11  /
10
ven

PL/SQL procedure successfully completed.
```

BIND VARIABLES

A bind variable is a variable that is declared in a host environment. Bind variables can be used to pass run-time values, which can be either number or character, into or out of one or more PL/SQL programs.

Example:

```
SQL> variable a number;

SQL> ed

File:

  1  begin

  2  select ssn into:a from emp where name='x';

  3  dbms_output.put_line(:a);

  4 end;

SQL> /

101

PL/SQL procedure successfully completed.

SQL> print a;

         A

----------

       101


PL/SQL procedure successfully completed.
```

REFERENCING NON PL/SQL VARIABLES

To reference host variables, prefix the references with a colon (:) to distinguish them from declared PL/SQL variable.

```
SQL> variable gg number;

SQL> define aa=1000;

SQL> set verify off

SQL> declare

  2  v_sal number(9,2):=&aa;

  3  begin

  4  :gg:=v_sal/12;

  5  end;

  6  /
```

PL/SQL procedure successfully completed.

```
SQL> print gg;

        GG

----------

83.3333333
```

PL/SQL BLOCK SYNTAX AND GUIDELINES

A line of pl/sql text contains groups of characters known as lexical units.

Lexicals are classified as follows:

- Delimiters
- Identifiers, which include reserved words
- Literals
- Character literals
- Numeric literals

COMMENTS:

-- single line commenting

/* beginning */ ending

PL/SQL HAS ITS OWN ERROR HANDLING:

- SQLCODE
- SQL ERRM

DATA TYPE CONVERSION

PL/SQL performs implicit conversions. For E.g. numeric to char.

The following program highlights conversion involving DATE.

```
SQL> ed
  1  declare
  2  vdate date;
  3  begin
  4  vdate:=to_date('aug 19 ,2005','mon dd,yyyy');
  5  dbms_output.put_line(vdate);
  6 end;
SQL> /
19-AUG-05
PL/SQL procedure successfully completed.
```

SQL FUNCTIONS IN PL/SQL:

All SQL functions are allowed except decode function and group functions.

NESTED BLOCKS AND VARIABLE SCOPE

```
SQL> declare
  2  v_a number:=3;
  3  begin
  4  declare
  5  v_b number:=4;
  6  begin
  7  dbms_output.put_line(v_b);
     dbms_output.put_line(v_a);
  8  end;
  9  dbms_output.put_line(v_a);
 10  end;
 11  /
4
3

PL/SQL procedure successfully completed.
```

QUALIFYING AN IDENTIFIER An identifier is qualified by using the block label prefix. In the example the outer block is labeled as outer. In the inner block the variable is reference by label, when variable names are same.

```
SQL> ed
  1  <<outer>>
  2  declare
  3  v_a number:=3;
  4  begin
  5  declare
  6  v_a number:=4;
  7  begin
  8  dbms_output.put_line(v_a);
  9  dbms_output.put_line(outer.v_a);
 10  end;
 11 end;
 12  /
4
3
PL/SQL procedure successfully completed.
```

PROGRAMMING GUIDELINES:

Category	Case Conversion	Examples
SQL statements	uppercase	SELECT,INSERT
PL/SQL statements	uppercase	DECLARE,BEGIN,IF
Data types	uppercase	VARCHAR2,BOOLEAN
Identifiers	lowercase	v_sal
Database tables and column	lowercase	emp, dept

INTERACTING WITH ORACLE SERVER:

- Extracts row of data from the database using **select**
- Effects changes in the database by using DML commands
- Controls a transaction with commit, rollback and save point

NOTES:

- An end in pl/sql block is not the end of transaction.
- A block can span multiple transactions, a transaction can span multiple blocks.
- DDL commands (create,alter,drop) and DCL commands(grant,revoke) are not directly supported.

SQL> select * from emp;

SSN	NAME	ESSN	DEPTNO	SALARY
101	x	102	1	

```
      102         y              103        2
      103          z              102        3
      104         p              102        4
      105         q
SQL> ed
  1  declare
  2  v_ssn number;
  3  v_name varchar2(10);
  4  begin
  5  select ssn,name into v_ssn,v_name from emp where name='x';
  6  dbms_output.put_line(v_ssn);
  7  dbms_output.put_line(v_name);
  8 end;
SQL> /
101
x
PL/SQL procedure successfully completed.
```

RETRIEVING DATA IN PL/SQL:

```
SQL> select * from job_grade;
GRA    LOWEST_SAL        HIGHEST_SAL
---- ---------- ----------------------------------------
a          3000                    4000
```

b 5000 6000

c 3000 6000

d 4000 10000

e 2000 6000

```
SQL> ed

  1  declare
  2  v_lsal job_grade.lowest_sal%type;
  3  v_hsal job_grade.highest_sal%type;
  4  begin
  5  select sum(lowest_sal),sum(highest_sal) into v_lsal,v_hsal from job_grade;
  6  dbms_output.put_line(v_lsal);
  7  dbms_output.put_line(v_hsal);
  8 end;
  9  /
17000
32000
PL/SQL procedure successfully completed.
```

NAMING CONVENTIONS

A local variable in pl/sql name must not be equal to column names present in database .

declare

lastname varchar2(10);

begin

delete from emp where lastname-lastname;

The above code will delete all employees because of the naming convention problem.

MANIPULATING DATA USING PL/SQL

SUBSTIUTION VARIABLE:

SQL> ed

Wrote file afiedt.buf

```
1  declare
2  v_sal number;
3  begin
4  v_sal:=&v_sal;
5  dbms_output.put_line(v_sal);
6 end;
7  /
```

Enter value for v_sal: 2000

2000

PL/SQL procedure successfully completed.

INSERTION

SQL> ed

```
Wrote file afiedt.buf

  1   begin
  2   insert into emp(ssn,name) values(123,'venkat');
  3   dbms_output.put_line('record inserted');
  4 end;
  5  /
record inserted
PL/SQL procedure successfully completed.
```

USAGE OF SUBSTITUTION VARIABLE:

```
SQL> ed
Wrote file afiedt.buf
  1   begin
  2   insert into emp(ssn,name) values(&ssn,'&name');
  3   dbms_output.put_line('record inserted');
  4  end;
SQL> /

Enter value for ssn: 124
Enter value for name: sampath
record inserted
PL/SQL procedure successfully completed.
```

UPDATE:

```
SQL> ed
Wrote file afiedt.buf
  1  declare
  2  v_sal number;
  3   begin
  4  v_sal:=&v_sal;
  5  update job_grade set lowest_sal=v_sal where gra='a';
  6   dbms_output.put_line('record updated');
  7  end;
SQL> /
Enter value for v_sal: 12000
record updated
PL/SQL procedure successfully completed.
SQL> select *from job_grade;

GRA  LOWEST_SAL          HIGHEST_SAL
---- ---------- ----------------------------------------
a         12000                    4000
b          5000                    6000
c          3000                    6000
```

d 4000 10000

e 2000 6000

DELETE:

```
SQL> ed
Wrote file afiedt.buf
  1  declare
  2  v_sal number;
  3   begin
  4  v_sal:=&v_sal;
  5   delete from job_grade where lowest_sal=v_sal;
  6  dbms_output.put_line('record deleted');
  7  end;
  8  /
Enter value for v_sal: 5000
record deleted


PL/SQL procedure successfully completed.
SQL> select * from job_grade;
GRA  LOWEST_SAL HIGHEST_SAL
---- ---------- -----------
a         12000        4000
c          3000        6000
d          4000       10000
```

e 2000 6000

CONTROL STRUCTURES

- **IF statements**
  - **If –then-end if**
  - **If-then-else-end if**
  - **If-then-elseif-end if**
- **Case expressions**
- **Loop statements**
  - **Basic loops**
  - **While loops**
  - **For loops**

Syntax of IF:

If condition **then**

Statements;

Else if condition **then**

Statements;

Else

Statements;

End if;

Examples:

1) Find the greatest among two numbers

```
1 declare
2 a number;
3 b number;
4 begin
5 a:=&a;
6 b:=&b;
7 if a>b then
8 dbms_output.put_line('gratest number is'||a);
9 else
10 dbms_output.put_line('gratest number is'||b);
11 end if;
12 end;

SQL> /

Enter value for a: 12 old 5: a:=&a;
new 5: a:=12;
Enter value for b: 4 old 6: b:=&b;

new 6: b:=4;
greatest number is12

PL/SQL procedure successfully completed
```

2) **if else with database**

DEPT_ID	DEPT_NAME	MANAGER_ID	LOCATION_ID
10	**cse**	**200**	**1700**
20	**it**	**300**	**1800**
30	**mech**	**400**	**1500**
40	**ece**	**500**	**1600**

```
1 declare
2 v_id departments.dept_id%type;
3 v_dname departments.dept_name%type;
4 begin
5 select dept_id,dept_name into v_id,V_dname from departments where manager_id=200;
6 if v_id=11 then
7 dbms_output.put_line(v_dname);
8 elsif v_dname='cse' then
9 dbms_output.put_line(v_id);
10 else
11dbms_output.put_line('recorde not match');
12 end if;
13 end;
SQL> / 10
```

PL/SQL procedure successfully completed.

3) **if/else if/ else**

```
1 declare
2 a number;
3 b number;
4 c number;
5 begin
6 a:=&a;
7 b:=&b;
8 c:=&c;
9 if (a>b) and (a>c) then
10 dbms_output.put_line('gratest number is'||a);
11 elsif(b>a) and (b>c) then
12 dbms_output.put_line('greatest number is'||b);
13 else
14 dbms_output.put_line('greatest number is'||c);
15 end if;
16 end;
SQL>/
Enter values for a,b and c: 6 4 12
C is greater :12.
```

PL/SQL procedure successfully completed.

4) Case Expressions

A case expression selects a result and returns it. To select the result, the case expression uses an expression whose value is used to select one of several alternatives.

Syntax :

CASE selector

WHEN

expression1

THEN result1 WHEN

expression2

THEN result2

WHEN expression N THEN result N [ELSE resultN+1

END;

Example:

1 declare

2 va varchar2(10);

3 v_result varchar2(10);

4 begin

5 va:=&va;

```
6 v_result:=
7 CASE va
8 WHEN 'a' THEN 'excellent'
9 WHEN 'b' THEN 'very good'
10 WHEN 'c' THEN 'good'
11 ELSE 'poor'
12 end;
13 dbms_output.put_line('grade is'||v_result);
14 end;
15 /
SQL> Enter value for va: 'a' old 5: va:=&va;
new 5: va:='a';
grade is excellent

PL/SQL procedure successfully completed.

SQL> /

Enter value for va: 'b' old 5: va:=&va;
new 5: va:='b';
grade is very good

PL/SQL procedure successfully completed.
SQL> /
```

Enter value for va: 'c' old 5: va:=&va;

new 5: va:='c';

grade is good

PL/SQL procedure successfully completed.

5) For Structure

SQL> 1 begin

2 for emp_record in (select * from wer1) loop

3 insert into wer1(name,ssn) values(emp_record.name,emp_record.ssn);

4 end loop;

5 commit;

6 end;

7 /

PL/SQL procedure successfully completed

CURSORS

The oracle server uses work areas, called private SQL areas, to execute SQL statement and to store processing information. This area is called cursor.

Cursor types:

- Implicit: queries returns only one row
- Explicit : queries returns more than one row

Explicit cursor

Active set: set of rows returned by multiple rows

Controlling explicit cursor

Open the cursor and execute the query associated with the cursor which identifies the result set.

Fetch

Retrieves the current row an advance the current row

Close the cursor

Syntax:

Cursor declaration

cursor cuname is select;

Open the cursor

open cursor name;

Close the cursor

close cursor name;

Fetch

fetch cname into variable or record

Explicit Cursor Attributes: To determine the status of the cursor, the cursor's attributes are checked.Cursors have the following four attributes that can be used in a PL/SQL program.

%isopen -To check if the cursor is opened or not

%found-To check if a record is found and can be fetched from the cursor

%rowcount-To check for the number of rows fetched from the cursor

%notfound-To check if no more records can be fetched from the cursor

%isopen, %found,%notfound are boolean attributes which are set to either TRUE or FALSE.

A Simple Example:

```
1 declare
2 v_name wer1.name%type;
3 v_ssn wer1.ssn%type;
4 cursor emp_c is select * from wer1;
5 begin
6 open emp_c;
7 for i in 1..5 loop
8 fetch emp_c into v_name,v_ssn;
9 dbms_output.put_line(v_name);
```

```
10 end loop;
11 close emp_c;
12 end;

PL/SQL procedure successfully completed.
SQL> set serveroutput on;

SQL> / x

x
x
y
y
```

2) %row count

```
1 declare
2 v_name wer1.name%type;
3 v_ssn wer1.ssn%type;
4 cursor emp_c is select * from wer1;
5 begin
6 open emp_c;
7 for i in 1..5 loop
8 fetch emp_c into v_name,v_ssn;
9 exit when emp_c%rowcount>4;
10 dbms_output.put_line(v_name);
```

11end loop;

12 close emp_c;

13 end;

SQL> / x

x

x

y

PL/SQL procedure successfully completed.

3) Cursor with record

It processes the rows of the active set by fetching values into a PL/SQL record.

1 declare

2 cursor emp_c is select * from wer1;

3 emp_record emp_c%rowtype;

4 begin

5 open emp_c;

6 for i in 1..5 loop

7 fetch emp_c into emp_record;

8 exit when emp_c%notfound;

9 insert into wer(name,ssn) values(emp_record.name,emp_re

10 end loop;

11 commit;

12 close emp_c;

```
13 end;
14 /
```

PL/SQL procedure successfully completed.

```
SQL> select * from wer;
```

NAME	SSN
x	101
x	101
x	101
x	101
x	101
x	101
x	101
x	101
x	101
x	101
x	101
y	102
y	102

12 rows selected.

4) Cursor with parameters

It passes the parameter values to the cursor in a cursor FOR loop. This means that you can open and close an explicit cursor several times in a block, returning a different active set on each occasion.

Example:

```
1 declare
2 v_number number;
3 v_name varchar2(10);
4 cursor c1(eno number,ename varchar2) is
5 select ssn,name from emp where ssn=eno and name=ename;
6 begin
7 open c1(101,'x');
8 fetch c1 into v_number,v_name;
9 dbms_output.put_line(v_number);
10 close c1;
11 open c1(102,'y');
12 fetch c1 into v_number,v_name;
13 dbms_output.put_line(v_number);
14 close c1;
15 end;
16 /
101
102
PL/SQL procedure successfully completed.
```

5) Update

The update clause in the cursor query locks the affected rows when the cursor is opened.

Example:

```
declare

v_number number;
v_name varchar2(10);

cursor c1(eno number,ename varchar2) is select ssn,name from emp where ssn=eno and
name=ename for update of name nowait;
begin
open c1(101,'x');
fetch c1 into v_number,v_name;
dbms_output.put_line(v_number);
close c1;
open c1(102,'y');
fetch c1 into v_number,v_name;
dbms_output.put_line(v_number);
close c1;
end;
```

EXCEPTIONS

Syntax

When exception1

then Statement1

Statement2

……..

When exception2

then Statement1

Statement2

……..

When others

then Statement1

Statement2 ……..

Sample predefined exceptions:

NO_DATA_FOUND

TOO_MANY_ROWS

INVALID_CURSOR

ZERO_DIVIDE

DUP_VAL_ON_INDEX

Example:

1)

1 declare

```
2 a number;
3 b number;
4 c number;
5 begin
6 a:=5;
7 b:=0;
8 c:= a/b;
9 exception
10 when zero_divide then
11 dbms_output.put_line('zero divide error');
12 end;

SQL> /
zero divide error
PL/SQL procedure successfully completed.
```

2) Non-predefined error

Trapping a non-predefined exception

1. Declare the name for the exception within the declarative section
2. Associate the declared exception with the standard oracle server error number using the PRAGMA EXCEPTION_INIT statement

Syntax : PRAGMA EXCEPTION_INIT(exception, error_number);

3. Reference the declared exception within the corresponding exception –handling routine.

Example:

```
1 declare
2 emp_remain exception;
3 pragma exception_init
4 (emp_remain,-2292);
5 begin
6 delete from emp where deptno=&deptno;
7 commit;
8 exception
9 when emp_remain then
10 dbms_output.put_line('cannot remove dept'|| 'employee exist');
11end;

SQL> /
Enter value for deptno: 2

old 6: delete from emp where deptno=&deptno;
new 6: delete from emp where deptno=2;
cannot remove deptemployee exist

PL/SQL procedure successfully completed

SQL> /
Enter value for deptno: 8
old 6: delete from emp where deptno=&deptno;
```

new 6: delete from emp where deptno=8;

PL/SQL procedure successfully completed

3) Functions for trapping exceptions

When an exception occurs, you can identify the associated error code or error message by using two functions.

SQLCODE: It returns the numeric value for the error code

SQLERRM: It returns character data containing the message associated with the error number.

Syntax:

declare;

v_error_code number;

v_error_messgage varchar2(255);

begin

when others then rollback;

v_error_code:=sqlcode;

v_error_message:=sqlerrm;

dbms_output.put_line(v_error_code||v_error_message);

end;

User defined function:

User defined PL/SQL exception must be

- Declared in the declare section of a PL/SQL block
- Raised explicitly with RAISE statements

Example:

```
1 declare
2 invalid_dept exception;
3 begin
4 delete from emp where deptno=&deptno;
5 if sql%notfound then
6 raise invalid_dept;
7 end if;
8 exception
9 when invalid_dept then
10 dbms_output.put_line('the deptnumber is not valid');
11 end;

Enter value for deptno: 10
old 4: delete from emp where deptno=&deptno;
 new 4: delete from emp where deptno=10;
the deptnumber is not valid
PL/SQL procedure successfully completed.
```

PL/SQL Block Types

A PL/SQL program comprises one or more blocks.

It is classified into two blocks

- **Anonymous Blocks**
  It is unnamed blocks. It is declared at the point in an application where they are to be executed and are passed to the PL/SQL engine for execution at run time.

- **Subprograms**
  Subprograms are named PL/SQL blocks that can accept parameters and can be invoked. It can be declared either as procedures or as functions.

Overview of subprograms

A subprogram is named PL/SQL block that ca accept parameters and be invoked from a calling environment.

Two types of subprograms

- A procedure that performs an action
- A function that computes a value

Benefits of subprograms

- Easy maintenance
- Improved data security and integrity
- Improved performance
- Improved code clarity

Procedure

A procedure is a type of subprogram that performs an action. A procedure can be stored in the database, as a schema object, for repeated execution.

Syntax for creating procedure:

Create [or replace] procedure <procedure_name>

[(parameter1 [mode1] datatype1,parameter2 [mode2] datatype2,..)]

Is| As

PL/SQL Block;

The replace option indicates that if the procedure exists, It will be dropped and replaced with the new version created by the statement. Parameter name of a PL/SQL variable whose value is passed to or populated by the calling environment.

Mode: type of argument

IN, OUT, IN OUT

IN : It is the default mode and value is passed into subprogram.

OUT : It must be specified and is returned to calling environment.

IN OUT: It is passed into subprogram and returned to calling environment.

IN parameter

IN parameters are passed as constants from the calling environment into the procedure.

Example:1

```
1  create or replace procedure raise_salary
2  (grade in job_grade.gra%type)
```

```
 3  is
 4  begin
 5  update job_grade set lowest_sal=lowest_sal*1.10 where gra= grade;
 6*  end raise_salary;
 7  /
```

Procedure created.

SQL> execute raise_salary('a'); // executing procedure

PL/SQL procedure successfully completed.

SQL> select * from job_grade;

GRA	LOWEST_SAL	HIGHEST_SAL
a	13200	4000
c	3000	6000
d	4000	10000
e	2000	6000

IN, OUT parameter

Example:1

```
 1  create or replace procedure info
 2  (g in job_grade.gra%type,
 3   l_sal out job_grade.lowest_sal%type,
 4   h_sal out job_grade.highest_sal%type)
 5  is
 6  begin
 7  select lowest_sal,highest_sal into l_sal,h_sal from job_grade where gra=g;
 8* end info;
 9  /
```

```
SQL> edit g:\oracle\sql\info1.sql
SQL> @g:\oracle\sql\info1
Procedure created.
```

How to view the value of OUT parameters with sql *plus

1.Run the sql script file to generate and compile the source code.

2.Create host variables in sql*plus, using the **variable** command

3.Invoke the procedure, supplying these host variables as the OUT parameters.: reference the host variables in the execute command.

4.To view the values passed from the procedure to the calling environment ,use the print command.

```
SQL> variable g_sal number;
SQL> variable g1_sal number;
```

```
SQL> execute info('a',:g_sal,:g1_sal)

PL/SQL procedure successfully completed.

SQL> print g_sal;

     G_SAL
    ----------
     13200

SQL> print g1_sal;

    G1_SAL
    ----------
      4000
```

IN OUT parameter

Example:1

```
 1   create or replace procedure info
 2   (g in  out number)
 3   is
 4   begin
 5   select lowest_sal into g from job_grade where highest_sal=g;
 6*  end info;
 7  /

Procedure created.

SQL> variable g_sal number;
 1   begin
 2   :g_sal:=4000;
```

```
  3*  end;
```

PL/SQL procedure successfully completed.

```
SQL> print g_sal;

     G_SAL
    ----------
      4000
```

SQL> execute info (:g_sal)

PL/SQL procedure successfully completed.

```
SQL> print g_sal;

     G_SAL
   ----------
     13200
```

Methods for passing parameters

Positional : List actual parameters in the same order as formal parameters

Named : List actual parameters in library order by associating each with its corresponding formal parameter

Combination : List some of the actual parameters as positional and some as named.

Removing procedures

Drop a procedure stored in the database

Syntax:

Drop procedure procedure_name

Example:

Drop procedure raise_salary;

Functions

A function is a named PL/SQL block that returns a value. A function can be stored in the database as a schema object for repeated execution. A function is called as part of an expression.

Syntax:

Create [or replace] function function_name

[(parameter1 [mode1] datatype1,

Parameter2 [mode2] datatype2,

….)]

Return datatype

Is/as

PL/SQL block ;

Example:

declare

summation number;

```
average number;
function summa(m4 number,m5 number) return number is
begin
return(m4+m5);
end;
function aver( summ1 number) return number is
begin
return(summ1/2);
end;
begin
 summation:=summa(&m1,&m2);
 average:=aver(summation);
 dbms_output.put_line('summation is:'||summation);
 dbms_output.put_line('average is:'||average);
end;
```

Removing functions

Drop function function_name

Example:

Drop function summa;

Packages

Packages bundle are related PL/SQL types, items, and subprograms into one container.

A package usually has a specification and a body, stored separately in the database.

Package specification

It is the interface to the application. It declares the types, variables, constants, exceptions, cursors and subprograms.

A package specification can exist without a package body, but a package body cannot exist without a package specification.

Syntax

Create [or replace] package package_name

is| as

Public type and item declarations

Subprograms specifications

End package_name;

Example:

```
1  create or replace package commp is
2  g_comm number:=0.10;
3  procedure reset_comm
4  (p_comm in number);
5 end commp;
```

Package created.

Package body

Syntax

Create [or replace] package body package_name

Is| as

Private type and item declarations

Subprogram bodies

End package_name;

Example

```
 1  create or replace package body commp
 2  is
 3  function validate_comm(p_comm in number)
 4  return boolean
 5  is
 6  v_max_comm number;
 7  begin
 8  select max(lowest_sal) into v_max_comm from job_grade;
 9  if p_comm>v_max_comm then return(false);
10  else return(true);
11  end if;
12  end validate_comm;
```

```
 13  procedure reset_comm(p_comm in number)
 14  is
 15  begin
 16  if validate_comm(p_comm)
 17  then g_comm:=p_comm;
 18  else
 19  raise_application_error(-20210,'invalid commision');
 20  end if;
 21  end reset_comm;
 22 end commp;
 23  /
```

Package body created.

Invoking package constructs:

```
SQL> execute commp.reset_comm(0.15);

PL/SQL procedure successfully completed.

SQL> create or replace package global_con is
  2  a constant number:=2;
  3  b constant number :=3;
  4  end global_con;
  5  /
```

Package created.

```
SQL> execute dbms_output.put_line('20 miles='||20*global_con.a||'km');
```

20 miles=40km

PL/SQL procedure successfully completed.

Referencing a public variable from a stand alone procedure:

```
SQL> ed
Wrote file afiedt.buf
  1  create or replace procedure me( x in number, y out number)
  2  is
  3  begin
  4  y :=x *global_con.a;
  5 end me;
SQL> /
Procedure created.

SQL> variable ya number;
SQL> execute me(3,:ya);
PL/SQL procedure successfully completed.
SQL> print ya;
        YA
----------
         6
```

Removing packages:

drop package package name;

drop package body package_name;

Overloading

It is the use of same name for different subprograms inside a PL/SQL block, a subprogram, or a package.

Example:

```
1  create or replace package   over
2  is
3  procedure add_dept(p_n in emp.ssn%type,p_na in emp.name%type);
4  procedure add_dept(p_n in emp.ssn%type,p_na in emp.name%type,p_dept in
emp.deptno%type);
5 end over;
6  /
```

Package created

```
1  create  package body  overp is
2  procedure add_dept(p_n emp.ssn%type,p_na emp.name%type)
3  is
4  begin
5  insert into emp (ssn,name) values(p_n,p_na);
6  end add_dept;
7  procedure add_dept(p_n emp.ssn%type,p_na emp.name%type,p_dn emp.deptno%type)
8  is
9  begin
10  insert into emp (ssn,name,deptno) values(p_n,p_na,p_dn);
```

```
11  end add_dept;
12 end overp;
```

Trigger

A trigger is aPL/SQL block or a PL/SQL procedure associated with a table ,view, schema, or the database. It executes implicitly whenever a particular event takes place.

It can be:

Application trigger: fires whenever an event occurs with a particular application

Database trigger: fires whenever a data event or system event occurs on a schema or database.

A triggering statement contains:

- Triggering timing
  - For table: BEFORE, AFTER
  - For view: INSTEAD OF
- Triggering event: INSERT, UPDATE, or DELETE
- Table name: on table, view
- Trigger type: row or statement
- When clause: restricting condition
- Trigger body: PL/SQL block

Trigger type

Statement trigger: The trigger body executes once for the triggering event. this is default. A statement trigger fires once, even if no rows are affected at all.

Row trigger: The trigger body executes once for each row affected by the triggering event. A

row trigger is not executed if the triggering event affects no rows.

Syntax:

CREATE [OR REPLACE] TRIGGER trigger_name

Timing

Event1 [OR event2 OR event3]

ON table_name

Trigger _body

Example:

```
create trigger ab
 before insert or delete or update on a
 for each row
 begin
 raise_application_error(-20000,'not accessible')
end
```

This program raises an error during insertion and deletion and update operation in a row.
