PL/SQL

PL/SQL stands for Procedural Language extension of SQL.

By

MOHD MIRZA

OCP(Oracle Certified Professional)

PL/SQL is a combination of SQL along with the procedural features of programming languages.

Properties of programming as well as the great interaction with database.

PLSQL IS NOT A CASE SENSITIVE LANG.

COMMENTS IN PLSQL

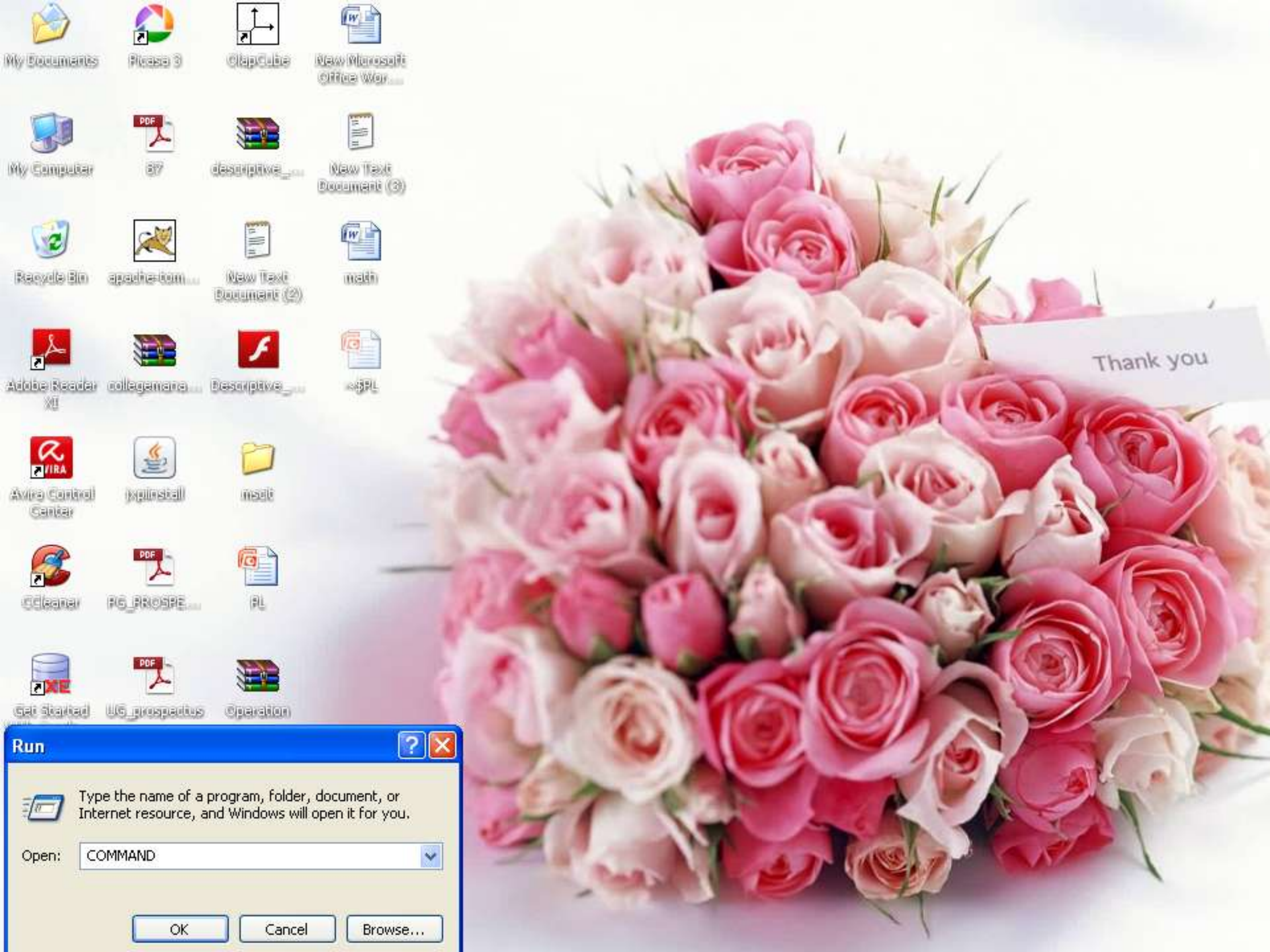
The PL/SQL compiler ignores comments but you should not.

Single-line comments begin with a double hyphen (--)

Multiline comments begin with a slashasterisk (/*), end with an asterisk-

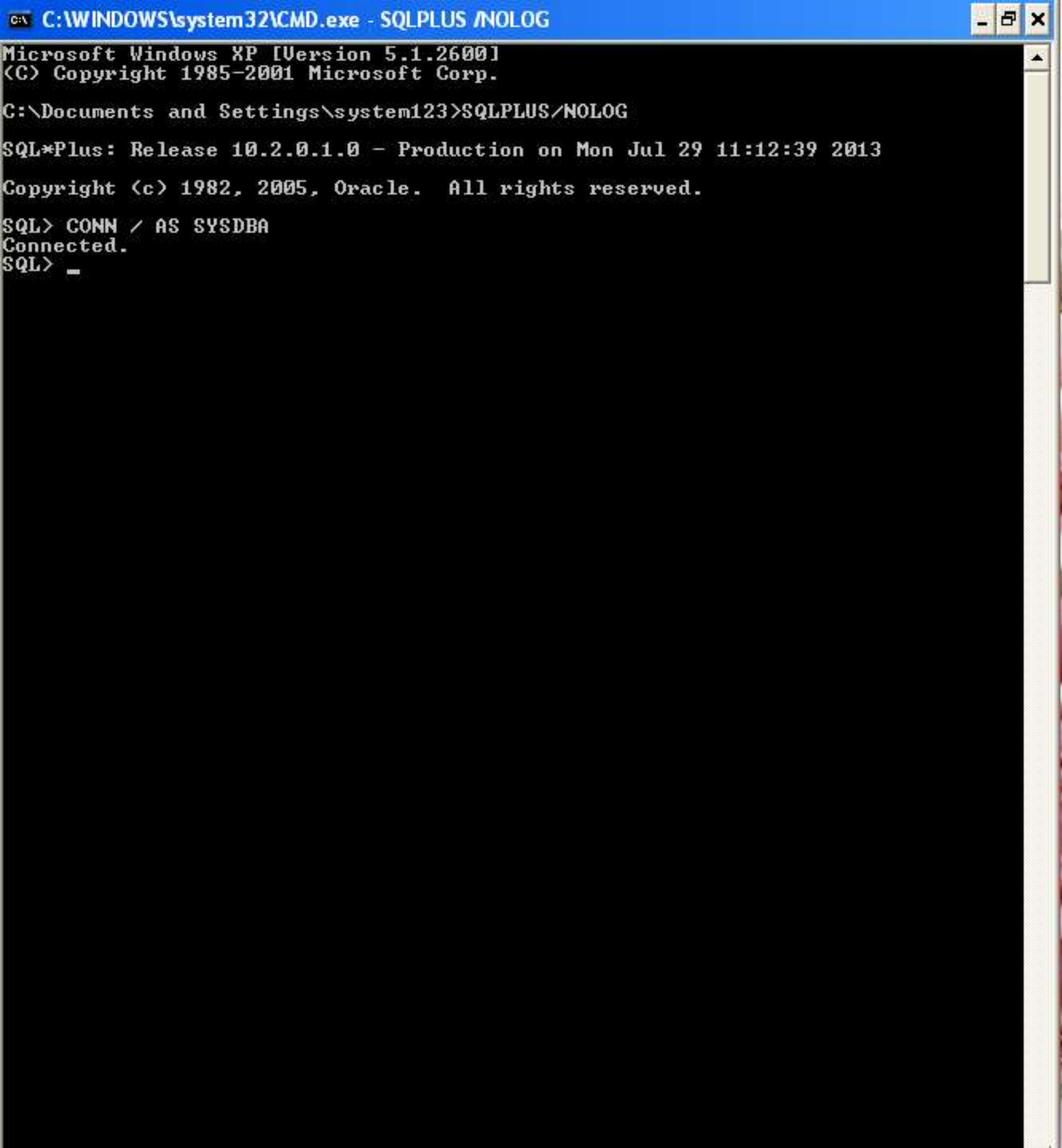
WHERE AND HOW TO RUN ORACLE PL/SQL IN WINDOWS ?

- YOU HAVE ORACLE 9i/10g/11g in your system.
- THEN FOLLOWS THESE STEPS
- 1) OPEN RUN PROMPT
- 2)TPYE SQLPLUS/NOLOG
- 3)TYPE CONNECT AND THEN ENTER
- USERNAME :HR
- PASSWORD:HR



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: COMMAND



C:\WINDOWS\system32\CMD.exe - SQLPLUS /NOLOG

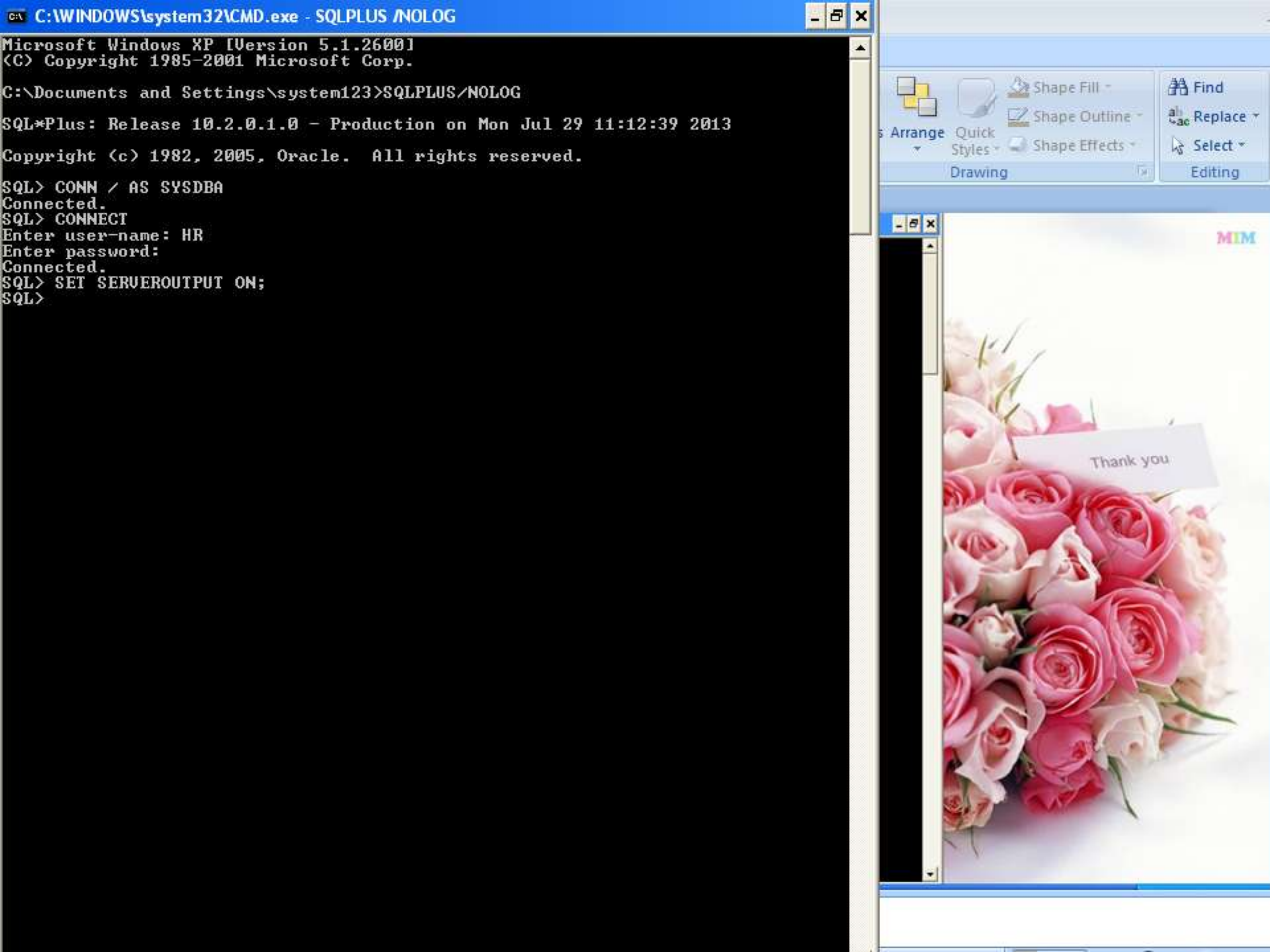
```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\system123>SQLPLUS/NOLOG

SQL*Plus: Release 10.2.0.1.0 - Production on Mon Jul 29 11:12:39 2013

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

SQL> CONN / AS SYSDBA
Connected.
SQL> _
```



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\system123>SQLPLUS/NOLOG

SQL*Plus: Release 10.2.0.1.0 - Production on Mon Jul 29 11:12:39 2013

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

SQL> CONN / AS SYSDBA
Connected.
SQL> CONNECT
Enter user-name: HR
Enter password:
Connected.
SQL> SET SERVEROUTPUT ON;
SQL>
```

IMPORTANT PL SQL CONCEPTS

DECLARE :- if you want to decalre a variable in plsql program then it takes place in declare section

BEGIN:- is used to start the working of program and end is used to terminate the begin.

Delimiter is used to run (/)

WHAT TO DO PREVIOUSLY

- SET SERVEROUTPUT ON ; is run before every time when you compiled a program in a session.
- SET ECHO ON ; is optional

FOR PRINT ON YOUR SCREEN USE

- DBMS_OUTPUT.PUT_LINE command for e.g. if sal=10 and you want to print it
- Then it looks like
- dbms_output.put_line(‘the salary is ‘ ||sal);

Value assign in variable

```
Declare
Num number(11);
Begin
Num:=5;
```

User’s input for a variable

```
DECALRE
N NUMBER(11);
BEGIN
N:=&N;
DBMS_OUTPUT.PUT_LINE(‘THE VALUE IS ‘||N);
END;
/
```

Sample program to print your ‘hello world’

```
BEGIN
Dbms_output.put_line(‘hello world’);
End;
/
```

/ is used to terminate plsql program called as delimeter

IF STATEMENT

IF STATEMENT WORKS AS SIMILAR AS C OR C++

Common syntax

```
IF condition THEN
statement 1;
ELSE
statement 2;
END IF;
```

Conditional statement IF then else

- DECLARE
- Age number(11);
- Begin
- Age:=&age;
- If age>18 then
- Dbms_output.put_line(‘u can vote’);
- Else
- Dbms_output.put_line(‘u cannot vote’);
- End if;
- End;
- /

USE OF IF WITH SQL TABLE

- Declare
- A number(11);
- Begin
- Select salary into a from emp where name='ram';
- If a>1000 then
- Update emp set bonus=bonus+1000 where name='ram';
- Else
- Update emp set bonus=bonus+500 where name='ram';
- End if;
- End;
- /

To print Pat salary from employees table using pl program

- Declare
- n number(11);
- Begin
- Select salary **<u>into</u>** n from employees where first_name=‘Pat’;
- Dbms_output.put_line(‘the Pat sal is ‘ ||n);
- End;
- /

INTO COMMAND

I

INTO command is used to catch a value in variable from table under some while condition

Only one value must be returned

For e.g. in the above example if there are two people who's name is john then it shows error

LOOPS IN PLSQL

1) SIMPLE LOOP
2) WHILE LOOP
3) FOR LOOP

;

FOR LOOP

- Print number from 1 to 10 using for loop

- BEGIN
- FOR i in 1 ..10 loop
- Dbms_output.put_line(i);
- End loop
- End;
- /
- (For has NO need to initialize explicitly but it need in while)

While loop

- PRINT NUMBERS FROM 1 TO 10 USING WHILE LOOP

- Declare
- i number(3):=0;
- Begin
- While i<=10 loop
- i:=i+1;
- Dbms_output.put_line(i);
- End loop;
- End;
- /

SIMPLE LOOP

- LOOP
- Statement 1;
- Statement 2;
- Exit condition
- End loop;

USE OF LOOP IN RDBMS

NAME	ID	SAL

ONE CAN EASILY INSERT ID FROM SAY 1 TO 100 USING LOOP

CREATE TRIGGERS

- A trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

Insert Triggers:

BEFORE INSERT Trigger

AFTER INSERT Trigger

Update Triggers:

BEFORE UPDATE Trigger

AFTER UPDATE Trigger

Delete Triggers:

BEFORE DELETE Trigger

AFTER DELETE Trigger

Drop Triggers:

Drop a Trigger

Disable/Enable Triggers:

Disable a Trigger

Disable all Triggers on a table

Enable a Trigger

Enable all Triggers on a table

TRIGGER RESTRICTION IS OPTIONAL (WHEN CLAUSE)

ONE TRIGGER MAY FIRE ANOTHER DATABASE TRIGGERS

- Create trigger abcd
- Before insert or update of sal on emp
- For each row
- when(new.sal>3000)
- begin
- :new.mgr:=1000;
- end;
- /

Explanation of last example

- If sal of any employee is updated and greator than 3000 then whose mgr values set to 1000.

SUPPOSE WE HAVE TWO TABLES ONE IS PRODUCT AND OTHER IS ORDER LIKE BIG BAZAR

PRODUCT AND ORDER TABLES

PNAME	PID	QTY

OPID	DESCRIPTIO	

- If qty from product table fall within 100 then automatically an order of that product is placed in order table.

- Create trigger abcd
- After update of qty on product
- For each row
- When(new.qty<100)
- Begin
- Insert into order values(:new.pid);
- End;
- /

EXCEPTION HANDLING
WHAT IS EXCEPTION …?

- AN EXCEPTION IS AN ERROR PL/SQL THAT IS RAISED DURING PROGRAM EXECUTION
- AN EXCEPTION CAN BE RAISED BY
- 1) IMPLICITLY BY THE ORACLE SERVER
- 2) Explicitly by the program

Type of Exception

- There are 3 types of Exceptions.

  a) Named System Exceptions
  b) Unnamed System Exceptions
  c) User-defined Exceptions

1) Named System Exceptions

- System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule.
- For e.g.
- 1)CURSOR_ALREADY_OPEN
- 2)NO_DATA_FOUND
- 3)TOO_MANY_ROWS
- 4)ZERO_DIVIDE

TOO_MANY_ROWS EXAMPLE

- SUPPOSE YOU WANT TO RETRIEVE ALL EMPLOYEES WHOSE NAME=‘JOHN’
- DECLARE
- a varchar(12)
- SELECT LAST_NAME into a from employees where first_name=‘john’
- Dbms_output.put_line(‘john last name is ‘ ||a);
- End;
- /

But if too many people have first_name=‘john’ then using exception handling ☟

- DECLARE
- a varchar(12)
- SELECT LAST_NAME into a from employees where first_name=‘john’
- Dbms_output.put_line(‘john last name is ‘ ||a);
- End;
- /
- Exception
- When too_many_rows then
- Dbms_output.put_line(‘your st. gets many rows ‘);
- End;
- /

2)Unnamed System Exceptions

- Those system exception for which oracle does not provide a name is known as unamed system exception
- There are two ways to handle unnamed sysyem exceptions:
  1. By using the WHEN OTHERS exception handler, or
  2. By associating the exception code to a name and using it as a named exception

Unnamed System Exceptions CONT..

- We can assign a name to unnamed system exceptions using a **Pragma** called **EXCEPTION_INIT.** **EXCEPTION_INIT** will associate a predefined Oracle error number to a programmer_defined exception name.

FOR E.G.

- DECLARE
- AAA EXCEPTION; -------→AAA IS EXCEPTION NAME
- PRAGMA --------------------→USE TO DEFINE UNMANED EXCEPTION
- EXCEPTION_INIT (AAA, -2292); -------------------------→MUST BE VALID EXCEPTION NUMBER
- BEGIN
- Delete FROM SUPPLIER where SUPPLIER_ID= 1;
- EXCEPTION
- WHEN AAA
- THEN Dbms_output.put_line(' $$Child records are present for this product_id.');
- END;
- /

WHAT HAPPENS IN PREVIOUS EXAMPLE

- IF U WANT TO DELETE SUPPLIER_ID=1 THEN AN EXCEPTION OCCURS WHICH WILL PRINT AS WHICH IS IN DBMS_OUTPUT.
- ACTUALLY THIS ECEPTION WORKS ON PARENT CHILD DELETION AND THE ERROR NUMBER SIGNIFIES THE TYPE OF ERR AND FOR USER EASE WE MAKE A USEFUL OR UNDERSTANDABLE PRINT STATEMENT

3) User-defined Exceptions

- Apart from sytem exceptions we can explicity define exceptions based on business rules. These are known as user-defined exceptions.

- DECLARE
- my-exception EXCEPTION;
- ----
- ----
- Raise name_of_exception;

FOR E.G.

- DECLARE
- ----
- Zero_commission Exception;
- BEGIN
- IF commission=0 THEN
- RAISE zero_commission
- EXCEPTION
- WHEN zero_commission THEN
- Process the error
- END;

For example
When the user enters an invalid ID, the exception invalid id is raised

- DECLARE
-    c_id customers.id%type := &cc_id;-------------→input id at run time
-    c_name customers.name%type; --------------→c_name variable as same datatype
-    c_addr customers.address%type;   as customer name datatype

-    -- user defined exception
-    ex_invalid_id EXCEPTION;------------→exception name
- BEGIN
-    IF c_id <= 0 THEN
-       RAISE ex_invalid_id;----------------→raise user condition
-    ELSE
-       SELECT name, address INTO c_name, c_addr
-       FROM customers
-       WHERE id = c_id;

- DBMS_OUTPUT.PUT_LINE ('Name: '|| c_name);
-       DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
-    END IF;
- EXCEPTION
-    WHEN ex_invalid_id THEN -------→user exception
-       dbms_output.put_line('ID must be greater than zero!');
-    WHEN no_data_found THEN   -------→predefined exception
-       dbms_output.put_line('No such customer!');
-    WHEN others THEN-------------------→predefined exception
-       dbms_output.put_line('Error!');
- END;
- /

STORED PROCEDURE

- SOMETHING LIKE FUNCIONS IN C/C++.
- A **stored procedure** is a subroutine available to applications that access a relational database system. A stored procedure (sometimes called a **proc**, **sproc**, **StoPro**, **StoredProc**, **sp** or **SP**) is actually stored in the database data dictionary.
- A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

A procedure may or may not return any value

Common syntax

- CREATE [OR REPLACE] PROCEDURE procedure_name
- [(parameter [,parameter])]
- IS
- [declaration_section]
- BEGIN
- executable_section
- [EXCEPTION exception_section]
- END [procedure_name];

EXAMPLE WITHOUT PARAMETER

- **CREATE OR REPLACE PROCEDURE** MYSTPROC **IS**
- **BEGIN**
- DBMS_OUTPUT.PUT_LINE('Hello World!');
- **END;**
- /

When you create a procedure or function, you may define parameters. There are three types of parameters that can be declared

1)IN
2)OUT
3)IN OUT

PARAMETER TYPES

- **1) IN type parameter:** These types of parameters are used to send values to stored procedures.
  2) OUT type parameter: These types of parameters are used to get values from stored procedures. This is similar to a return type in functions.
  3) IN OUT parameter: These types of parameters are used to send values and get values from stored procedures.

1) IN PARAMETER

This is similar to passing parameters in programming languages. We can pass values to the stored procedure through these parameters or variables.

```
CREATE OR REPLACE PROCEDURE MYPROC(param1 IN VARCHAR2)
IS
 BEGIN
  DBMS_OUTPUT.PUT_LINE('Hello World IN parameter ' || param1);
  END;
 /
```

2) OUT Parameter

- **CREATE OR REPLACE PROCEDURE** procname (outParam1 **OUT** VARCHAR2)
- **IS**
- **BEGIN**
- outParam1 := 'Hello World OUT parameter';
- **END**;
- /
- ***Run it***
- **DECLARE** outParam1 VARCHAR2(100);
- **BEGIN**
- Procname (outParam1);
- DBMS_OUTPUT.PUT_LINE(outParam1);
- **END**;
- /

DIFF B/W PROCEDURE AND FUNCTION

- The functions can return only one value and procedures not. Functions can be call from SQL Statements, procedures not and there are some things that you can
  do in a stored procedure that you can not do in a function.

FOR E.G.

- Create a stored procedure that adds 1000 to each employees commission watch for Null values
- Create procedure st_proc as
- Begin
- Update emp set comm=nvl(comm,0)+1000;
- End;
- /

Procedure can call at any time using

- Execute st_proc; -------→ procedure name
- OR

- Exec st_proc;---------------→procedure name

FUNCTIONS

- A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

FUNCTION EXAMPLE

- Create function funname
- Return number is
- a number(10);
- Begin
- Select avg(sal) into a from emp;
- return a;
- End;
- /

HOW TO EXECUTE FUNCTION ?

1) SELECT FUNCTIONNAME FROM DUAL;

2)DBMS_OUTPUT.PUT_LINE(FUNCTIONNAME);

CURSORS

- A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it.
- Cursors provide a way for your program to select multiple rows of data from the database and then to process each row individually.
- There are two types of cursors in PL/SQL:
- **1)IMPLICIT CURSORS**
- **2) *Explicit cursors***

A CURSOR CAN HOLD MORE THAN ONE ROW, BUT CAN PROCESS ONLY ONE ROW AT A TIME.

Implicit Cursors

- These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed.
- The set of rows returned by query is called active set.
- Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations. The are as follows
- ***1) %FOUND***
- ***2) %NOTFOUND***
- ***3) %ROWCOUNT***
- ***4) %ISOPEN***.

FOR E.G.

DECLARE
- n number(5);
- BEGIN
- UPDATE emp SET salary = salary + 1000;
- IF SQL%NOTFOUND THEN
- dbms_output.put_line('No sal are updated');
- ELSIF SQL%FOUND THEN
- n := SQL%ROWCOUNT;
- dbms_output.put_line('Sal for ' ||n || 'employees are updated');
- END IF;
- END;
- /

EXPLANATION

- ***%FOUND->***The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row and if SELECT ….INTO statement return at least one row.
- ***%NOTFOUND-***>same as above but if not found.
- ***%ROWCOUNT*** ->Return the number of rows affected by the DML operations

Explicit Cursors

- Explicit cursors are declared by and used by user to process multiple rows ,returned by select statement.
- An **explicit cursor** is defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. We can provide a suitable name for the cursor.

Explicit cursor management

- 1)Declare the cursor
- 2)Open the cursor
- 3)Fetch data from cursor
- 4)Close the cursor

Declaring the cursor

- CURSOR cursor_name IS select_statement;
- For e.g.
- Cursor cursor_name is
- Select name from emp where dept=‘maths’

Opening the cursor

- Open cursor_name
- For e.g.
- Open c_name
- Where c_name is the name of cursor.
- Open statement retrieves the records from db and places in the cursor(private sql area).

Fetching data from cursor

- The fetch statement retrieves the rows from the active set one row at a time. The fetch statement is used usually used in conjunction with iterative process (looping statements)
- Syntax: FETCH cursor-name INTO record-list
- Ex: LOOP
- -----------
- ------------
- FETCH c_name INTO name;
- -----------
- END LOOP;

Closing a cursor:

- Closing statement closes/deactivates/disables the previously opened cursor and makes the active set undefined.
- Syntax : CLOSE cursor_name

Cursor can store multiple rows at a time but without loop cursor cannot fetch multiple rows but only print very first row from your result e.g. on next slide

Without loop example ⇩

- declare
- a emp%rowtype;
- cursor cc is ------------→cursor name
- select * from emp where sal>1000;
- begin
- open cc;--------------------------------→open cursor
- fetch cc into a;-------------------------------→fetch cursor
- dbsm_output.put_line(a.ename || a.job);--→print multiple rows
- close cc;
- end;
- /
- output:-allen salesman

USING LOOP FOR FETCHING MULTIPLE ROWS THROUGH CURSORS

- declare
- cursor qaz is
- select ename,job from emp;
- a qaz%rowtype;------------------→a of cursor type variable
- begin
- open qaz;----------------------→open cursor
- loop
- fetch qaz into a;----------------→fetch cursor
- exit when qaz%notfound;-------------→exit when not found
- dbms_output.put_line(a.ename || a.job);
- end loop;
- end;
- /

Another Cursor example (not necessary)

- The HRD manager has decided to raise the salary for all the employees in the physics department by 0.05 whenever any such raise is given to the employees, a record for the same is maintained in the emp_raise table (the data table definitions are given below). Write a PL/SQL block to update the salary of each employee and insert the record in the emp_raise table.

Tables

- **Table: employee**
- emp_code varchar (10)
- emp_name varchar (10)
- dept varchar (15)
- job varchar (15)
- salary number (6,2)
- **Table: emp_raise**
- emp_code varchar(10)
- raise_date Date
- raise_amt Number(6,2)

- DECLARE
- CURSOR c_emp IS -------------------→cursor name
- SELECT emp_code, salary FROM employee----→query which stored in cursor
- WHERE dept = 'physics';
- a employee.emp_code %TYPE;-------→variable declare
- b employee.salary %TYPE;
- BEGIN
- OPEN c_emp;--------------------→open cursor
- LOOP--------------------------------→fetching records using loop
- FETCH c_emp INTO a, b;-----→ fetching records and stored in
- UPDATE employee SET salary : = b + (b* 0.05)
- WHERE emp_code = str_e;
- INSERT INTO emp_raise
- VALUES (str_emp_code, sysdate, num_salary * 0.05);
- END LOOP;
- Commit;
- CLOSE c_emp;
- END;

PACKAGES

- A Package is a container that may have many functions and procedures within it.
- A **PACKAGE** is a group of programmatic constructs combined.
- A package is a schema object that groups logically related PL/SQL types, items, and subprograms

A PACKAGE EXISTS IN TWO PARTS:

- ***Package Specification***:- The **specification** is the interface to your applications it declares the
  types, variables, constants, exceptions, cursors , and subprograms available for use. ***Package Body:-***This contains the definition of the constructs prototyped in the spec. It may also contain the private or locally defined program units, which can be used within the scope of the package body only..

OOP'S

- PACKAGES DEMONSTRATE ENCAPSULATION, DATA HIDING, SUBPROGRAM OVERLOADING AND MODULARIZATION IN PL/SQL

SIMPLE EXAMPLE ON PACKAGE

- STEP NO 1:- Package specification created first means the definition of constructs in pacakage

```
CREATE OR REPLACE PACKAGE PKG_NAME IS
PROCEDURE P_ENAME(P_VAR VARCHAR2);
END;
/
```

STEP NO 2) Creating package body

- CREATE OR REPLACE PACKAGE BODY

```
PKGNAME IS
PROCEDURE P_ENAME(P_VAR VARCHAR2) IS
BEGIN
DBMS_OUTPUT.PUT_LINE(P_VAR);
END;
END PKGNAME;→PACKAGE END
/
```

CALLING PACAKGE

- EXECUTE IS USED TO CALL A PACAKAGE
- EXEC PKG_UTIL.P_ENAME(‘MIRZA');
- OUTPUT:-MIRZA