**PL/SQL**

**Procedural language of oracle is called PL/SQL. This may consist of SQL or NON SQL statements.**

**Structure of a PL/SQL block**

**DECLARE**

**---------------- // Declaration part**

**BEGIN**

**------------------ // Execution part**

**EXCEPTION**

**------------------ // Exception handling part (optional)**

**END**

**SET SERVEROUTPUT ON**

**This command should be given at the beginning of each PL/SQL session.**

**To create a new file or open an existing file,**

**You may give the following command at the SQL prompt.**

**For eg to create a file called prg1.sql, give the following command**

**Ed prg1.sql and press Enter key. This will open notepad editor. There you can type the program.**

**DECLARE  
A NUMBER :=15;  
B NUMBER :=20;  
C NUMBER;  
BEGIN  
 C := A+B;  
 DBMS\_OUTPUT.PUT\_LINE('Sum is '||C);  
END;**

**After saving the program, you may run the program with the following command.**

**@PRG1.SQL**

**/**

**Program to accept values from the user**

**DECLARE  
A NUMBER := &A;  
B NUMBER := &B;  
C NUMBER;  
BEGIN  
 C := A+B;  
 DBMS\_OUTPUT.PUT\_LINE('Sum is '||C);  
END;**

**After running this program, Oracle will ask you enter values.**

**IF CONDITION**

**IF.…… ELSE…… END IF**

**Syntax**

**IF CONDITION THEN**

**STATEMENTS**

**ELSE**

**STATEMENTS**

**END IF**

**Eg**

**DECLARE  
 A NUMBER := &A;  
 B NUMBER := &B;  
BEGIN  
IF A>B THEN  
 DBMS\_OUTPUT.PUT\_LINE(A||' is greater');  
 ELSE  
 DBMS\_OUTPUT.PUT\_LINE(B||' is greater');  
 END IF;  
END;**

**IF…….. ELSIF…….ELSE……..END IF**

**Eg - Logical operator AND**

**DECLARE  
 A NUMBER :=&A;  
 B NUMBER :=&B;  
 C NUMBER :=&C;  
BEGIN  
IF A>B AND A>C THEN  
 DBMS\_OUTPUT.PUT\_LINE(A||' is the biggest no');  
 ELSIF B>A AND B>C THEN  
 DBMS\_OUTPUT.PUT\_LINE(B||' is the biggest no');  
 ELSE  
 DBMS\_OUTPUT.PUT\_LINE(C||' is the biggest no');  
 END IF;  
END;**

**Eg - Logical operator OR**

**DECLARE  
 X CHAR(1) := '&X';  
BEGIN  
 IF X = 'A' OR X = 'E' OR X = 'I' OR X = 'O' OR X = 'U' THEN  
 DBMS\_OUTPUT.PUT\_LINE(X||' is a vowel character');  
 ELSE  
 DBMS\_OUTPUT.PUT\_LINE(X||' is not a vowel character');  
 END IF;  
END;**

**Eg for NESTED IF**

**DECLARE  
 X CHAR(1) := '&X';  
BEGIN  
 IF (X >= 'A' AND X <= 'Z') OR (X >= 'a' AND X <= 'z') THEN  
 DBMS\_OUTPUT.PUT\_LINE('THIS IS A LETTER');  
 ELSE  
 DBMS\_OUTPUT.PUT\_LINE('THIS IS NOT A LETTER');  
 IF X BETWEEN '0' AND '9' THEN  
 DBMS\_OUTPUT.PUT\_LINE('THIS IS A NUMBER');  
 ELSE  
 DBMS\_OUTPUT.PUT\_LINE('NOT A NUMBER');  
 END IF;  
 END IF;  
END;**

**CASE…. END CASE**

**Eg.**

**DECLARE   
 N NUMBER := &N;  
 X NUMBER;  
 BEGIN  
 X := MOD(N,2);  
 CASE X  
 WHEN 0 THEN  
 DBMS\_OUTPUT.PUT\_LINE(N||' is an even number');  
 ELSE  
 DBMS\_OUTPUT.PUT\_LINE(N||' is an odd number');  
 END CASE;  
 END;**

**LOOPING STATEMENTS**

**These are used to repeatedly execute a statement.**

**WHILE LOOP**

**Syntax**

**WHILE (CONDITION)**

**LOOP**

**STATEMENTS**

**END LOOP**

**Eg.**

**DECLARE  
 A NUMBER := 1;  
 BEGIN  
 WHILE (A<=10)  
 LOOP  
 DBMS\_OUTPUT.PUT\_LINE(A);  
 A := A+1;  
 END LOOP;  
 END;**

**FOR LOOP**

**Syntax**

**FOR VARIABLE NAME IN LOWER BOUND..UPPER BOUND**

**LOOP**

**STATEMENTS**

**END LOOP**

**Eg.**

**DECLARE  
 A NUMBER;  
BEGIN  
 FOR A IN 1..10  
 LOOP  
 DBMS\_OUTPUT.PUT\_LINE(A);  
 END LOOP;  
 END;**

**Program to print numbers in reverse order**

**DECLARE  
 A NUMBER;  
BEGIN  
FOR A IN REVERSE 1..10  
 LOOP  
 DBMS\_OUTPUT.PUT\_LINE(A);  
 END LOOP;  
END;**

**Using SQL statements in a PL/SQL block.**

**EMPLOYEE**

|  |  |
| --- | --- |
| **ENO** | **NUMBER(4)** |
| **ENAME** | **VARCHAR(20)** |
| **DGN** | **CHAR(18)** |
| **SAL** | **NUMBER(6)** |
| **DEPTNO** | **NUMBER(2)** |

**Eg.**

**DECLARE  
 N NUMBER(4) :=&N;  
 EN VARCHAR(20);  
 S NUMBER(6);  
BEGIN  
SELECT ENO, ENAME, SAL INTO N, EN, S FROM EMPLOYEE WHERE ENO = N;  
 DBMS\_OUTPUT.PUT\_LINE(N||' '||EN||' '||S);  
END;**

**Note.**

## Ensure that the data type and size of column name matches with the respective variable.

**PL/SQL ATTRIBUTES**

**%TYPE**

**Refers to the column of a table.**

**Syntax.**

**Variable name Table name.column name%type**

**Eg.**

**S EMPLOYEE.SAL%TYPE;**

**%ROWTYPE**

Refers to the row of a table.

**Syntax.**

**Variable name Tablename%rowtype**

**Eg.**

**X EMPLOYEE%ROWTYPE;**

**Eg for %TYPE**

DECLARE  
 N EMPLOYEE.ENO%TYPE := &N;  
 EN EMPLOYEE.ENAME%TYPE;  
 S EMPLOYEE.SAL%TYPE;  
BEGIN  
 SELECT ENO,ENAME,SAL INTO N,EN,S FROM EMPLOYEE WHERE ENO = N;  
 DBMS\_OUTPUT.PUT\_LINE(N||' '||EN||' '||S);  
END;

**Eg for %ROWTYPE**

**DECLARE  
 N EMPLOYEE.ENO%TYPE := &N;  
 X EMPLOYEE%ROWTYPE;  
BEGIN  
SELECT \* INTO X FROM EMPLOYEE WHERE ENO = N;  
 DBMS\_OUTPUT.PUT\_LINE(X.ENAME||' '||X.DGN||' '||X.SAL);  
END;**

**Eg.**

**DECLARE  
 N EMPLOYEE.ENO%TYPE := &N;  
 DG EMPLOYEE.DGN%TYPE;  
BEGIN  
 SELECT DGN INTO DG FROM EMPLOYEE WHERE ENO = N;  
 IF DG IN('MANAGER') THEN  
 UPDATE EMPLOYEE SET SAL = SAL+2000 WHERE ENO = N;  
 ELSIF DG IN('ACCOUNTANT') THEN  
 UPDATE EMPLOYEE SET SAL = SAL+1000 WHERE ENO = N;  
 ELSE  
 UPDATE EMPLOYEE SET SAL = SAL+500 WHERE ENO=N;  
 END IF;  
 COMMIT;  
 DBMS\_OUTPUT.PUT\_LINE('Table updated');  
END;**

**EXCEPTION**

**Run time error. Exceptions can be classified into 2, ie Predefined exception and User defined exception.**

**Eg.**

**DECLARE  
 A NUMBER := &A;  
 B NUMBER := &B;  
 C NUMBER;  
BEGIN  
 C := A/B;  
 DBMS\_OUTPUT.PUT\_LINE(C);  
EXCEPTION  
 WHEN ZERO\_DIVIDE THEN  
 RAISE\_APPLICATION\_ERROR(-20000,'Division by zero is illegal');  
END;**

**Note. Error code should be in the range of -20000 to -20999.**

**Eg.**

**DECLARE  
 N EMPLOYEE.ENO%TYPE := &N;  
 EN EMPLOYEE.ENAME%TYPE;  
 S EMPLOYEE.SAL%TYPE;  
BEGIN  
SELECT ENO, ENAME, SAL INTO N, EN, S FROM EMPLOYEE WHERE ENO = N;** **DBMS\_OUTPUT.PUT\_LINE(N||' '||EN||' '||S);  
EXCEPTION  
 WHEN NO\_DATA\_FOUND THEN  
 RAISE\_APPLICATION\_ERROR(-20001,'Particular empno not exists');  
END;**

**User defined exception**

**Exception which is created by the user is called User defined exception. This should be declared in the declaration part and should be raised in the execution part.**

**Eg.**

## DECLARE EN EMPL.ENO%TYPE :=&EN; N EMPL.ENAME%TYPE; S EMPL.SAL%TYPE; NVAL EXCEPTION; BEGIN SELECT ENO,ENAME,SAL INTO EN,N,S FROM EMPL WHERE ENO = EN; IF S IS NULL THEN RAISE NVAL; ELSE DBMS\_OUTPUT.PUT\_LINE(EN||' '||N||' '||S); END IF; EXCEPTION WHEN NVAL THEN RAISE\_APPLICATION\_ERROR(-20004,'This column should not have null value'); END;

**CURSOR**

**Cursor can be defined as a memory area in which the result of SQL statements is stored. There are two types of cursors, ie. Implicit cursor and Explicit Cursor.**

**Implicit Cursor**

**This is automatically created by ORACLE whenever an SQL statement is executed.**

**Explicit Cursor**

**This is created by the user when he wants to use a multiple row select statement in a PL/SQL block.**

**Steps for creating a cursor**

1. **Declare the cursor**

**This will tell the database server to allocate memory space for the query which is to be executed.**

1. **Open the cursor**

**Here the query actually gets executed, a result set is created and the cursor points to the first row.**

1. **Fetch the records one by one from the result set using a loop command.**
2. **Close the cursor.**

**Cursor attributes**

1. **%ISOPEN**

**This attribute will return true if the cursor is already opened and false if it is not opened.**

1. **%FOUND**

**This attribute will return true if the FETCH command finds record in the result set and false if it doesn’t find records in the result set.**

1. **%NOTFOUND**

**This attribute will return true if the FETCH command doesn’t find record in the result set and false if it finds records in the result set.**

1. **%ROWCOUNT**

**This attribute will return the number of records that are fetched from the result set.**

**Eg.**

**DECLARE  
 CURSOR C1 IS SELECT \* FROM EMPLOYEE WHERE DEPTNO = 20;  
 X EMPLOYEE%ROWTYPE;  
BEGIN  
 OPEN C1;  
 LOOP  
 FETCH C1 INTO X;  
 EXIT WHEN C1%NOTFOUND;  
 DBMS\_OUTPUT.PUT\_LINE(X.ENAME||' '||X.SAL||' '||X.DEPTNO);  
 END LOOP;  
IF C1%ISOPEN THEN  
 DBMS\_OUTPUT.PUT\_LINE('Cursor is opened');  
 ELSE  
 DBMS\_OUTPUT.PUT\_LINE('Cursor is not opened');  
 END IF;  
 DBMS\_OUTPUT.PUT\_LINE('No of records '||C1%ROWCOUNT);  
 CLOSE C1;  
END;**

**CURSOR FOR LOOP**

**In the case of Cursor For Loop, all you have to do is to declare the cursor. Opening, fetching and closing are taken care of by the For loop.**

**DECLARE  
 CURSOR C1 IS SELECT \* FROM EMPLOYEE WHERE DEPTNO = 20;  
 X EMPLOYEE%ROWTYPE;  
 BEGIN  
FOR X IN C1  
LOOP  
 DBMS\_OUTPUT.PUT\_LINE(X.ENAME||' '||X.SAL||' '||X.DEPTNO);  
 END LOOP;  
 END;**

**Implicit Cursor attributes**

1. **SQL%ISOPEN**
2. **SQL%FOUND**
3. **SQL%NOTFOUND**
4. **SQL%ROWCOUNT**

**Eg for Implicit Cursor**

**DECLARE  
 N EMPLOYEE.ENO%TYPE :=&N;  
 EN EMPLOYEE.ENAME%TYPE;  
 S EMPLOYEE.SAL%TYPE;  
BEGIN  
SELECT ENO,ENAME,SAL INTO N,EN,S FROM EMPLOYEE WHERE ENO=N;  
 DBMS\_OUTPUT.PUT\_LINE(N||' '||EN||' '||S);  
 DBMS\_OUTPUT.PUT\_LINE('Total no of records '||SQL%ROWCOUNT);  
END;**

**Sub Program**

**This is a program which can be called from other programs (Oracle itself, Java, .net, Visual Basic etc). There are two types of sub programs, ie PROCEDURE and FUNCTION.**

**PROCEDURE**

**This performs an action and it may or may not return a value.**

**Eg**

**CREATE OR REPLACE PROCEDURE PR1 (A NUMBER, B NUMBER) IS  
 C NUMBER;  
BEGIN  
 C := A+B;  
 DBMS\_OUTPUT.PUT\_LINE('Sum is '||C);  
END;**

**To call the procedure, you may give the command,**

**EXEC PR1 (12, 34);**

**PARAMETERS IN PROCEDURE**

1. **IN PARAMETER**
2. **OUT PARAMETER**
3. **IN OUT PARAMETER**

**IN PARAMETER (Default parameter)**

**This will receive values from the calling program.**

**OUT PARAMETER**

**This will return the output value to the calling program.**

**IN OUT PARAMETER**

**This will both receive values from the calling program and return the output value back to the calling program.**

**Eg for OUT parameter**

**CREATE OR REPLACE PROCEDURE PR2 (N IN NUMBER, COMM OUT NUMBER) IS  
S EMPLOYEE.SAL%TYPE;  
BEGIN  
SELECT SAL INTO S FROM EMPLOYEE WHERE ENO = N;  
 COMM := S \* .1;  
END;**

**After creating the procedure (by running the above program), you may write a calling program.**

**Calling program**

**DECLARE  
 N EMPLOYEE.ENO%TYPE := &N;  
 COMM NUMBER;  
BEGIN  
 PR2(N,COMM);  
 DBMS\_OUTPUT.PUT\_LINE('Commission is '||COMM);  
END;**

**Eg for IN OUT PARAMETER**

**CREATE OR REPLACE PROCEDURE PR3 (CURRENCY IN OUT NUMBER) IS  
BEGIN  
CURRENCY := CURRENCY \* 82;  
END;**

**Calling program**

**DECLARE  
 CURRENCY NUMBER:=&CURRENCY;  
BEGIN  
 PR3(CURRENCY);  
 DBMS\_OUTPUT.PUT\_LINE(CURRENCY);  
END;**

**Dropping a Procedure**

**Eg.**

**DROP PROCEDURE PR1;**

**FUNCTION**

**This performs a calculation and it should return a value.**

**Eg.**

**CREATE OR REPLACE FUNCTION FUN1 (A NUMBER, B NUMBER) RETURN NUMBER IS  
 C NUMBER;  
BEGIN  
 C := A+B;  
 RETURN C;  
END;**

**Calling program**

**DECLARE  
 X NUMBER := &X;  
 Y NUMBER := &Y;  
 Z NUMBER;  
BEGIN  
 Z := FUN1(X,Y);  
 DBMS\_OUTPUT.PUT\_LINE('Sum is '||Z);  
END;**

**Eg.**

**CREATE OR REPLACE FUNCTION FUN2 (N NUMBER) RETURN NUMBER IS  
 COMM NUMBER;  
 S EMPLOYEE.SAL%TYPE;  
BEGIN  
SELECT SAL INTO S FROM EMPLOYEE WHERE ENO = N;  
 COMM := S \*.1;  
 RETURN COMM;  
END;**

**Calling program**

**DECLARE  
 N EMPLOYEE.ENO%TYPE := &N;  
 COMM NUMBER;  
BEGIN  
 COMM := FUN2(N);  
 DBMS\_OUTPUT.PUT\_LINE('Commission is '||COMM);  
END;**

**Dropping a Function**

**Eg.**

**DROP FUNCTION FUN1;**

**PACKAGE**

**This can be defined as a container for functions, procedures, exceptions etc. Package has two parts Package specification and Package body.**

**Package specification**

**Here we declare functions, procedures etc.**

**Package body**

**Here we implement the functions, procedures etc which we have declared in the package specification.**

**Eg for Package specification**

**CREATE OR REPLACE PACKAGE PK AS  
 PROCEDURE PRR (CURR IN OUT NUMBER);  
 FUNCTION FNN (A NUMBER, B NUMBER) RETURN NUMBER;  
END PK;**

**Eg for Package body**

**CREATE OR REPLACE PACKAGE BODY PK IS  
 PROCEDURE PRR (CURR IN OUT NUMBER) IS  
 BEGIN  
 CURR := CURR \* 82;  
 END PRR;  
FUNCTION FNN (A NUMBER, B NUMBER) RETURN NUMBER IS  
 C NUMBER;  
 BEGIN  
 C := A+B;  
 RETURN C;  
 END FNN;  
END PK;**

**Calling Program**

**DECLARE  
 X NUMBER := &X;  
 Y NUMBER := &Y;  
 Z NUMBER;  
BEGIN  
 Z := PK.FNN (X,Y);  
 DBMS\_OUTPUT.PUT\_LINE ('Sum is '||Z);  
END;**

**Procedure Overloading and Function Overloading**

**Procedure Overloading**

**This refers to the ability of a Package to have more than one Procedure with the same name, Procedures should differ either in the no of parameters or in the data type of parameters.**

**Function Overloading**

**This refers to the ability of a Package to have more than one Function with the same name, Functions should differ either in the no of parameters or in the data type of parameters.**

**TRIGGER**

**Trigger can be defined as a named PL/SQL block which is created on a table. This is automatically fired based on database events (INSERT, UPDATE and DELETE).**

**The most important advantage of Trigger is that it ensures a very high level of data integrity. Complex validations can be performed using Triggers.**

**Eg.**

**CREATE OR REPLACE TRIGGER T1 BEFORE INSERT OR UPDATE OR DELETE ON EMPLOYEE FOR EACH ROW  
 DECLARE**

**DT CHAR (15);  
 BEGIN  
SELECT TO\_CHAR (SYSDATE,'DAY') INTO DT FROM DUAL;  
IF DT IN ('SUNDAY','SATURDAY') THEN  
 RAISE\_APPLICATION\_ERROR(-20003,'Dont try insert, update, delete on weekends');  
 END IF;  
END;**

**Eg.**

**CREATE OR REPLACE TRIGGER T2 BEFORE INSERT OR UPDATE ON EMPLOYEE FOR EACH ROW  
 DECLARE  
 CTR NUMBER(2);  
 BEGIN  
SELECT COUNT(\*) INTO CTR FROM EMPLOYEE WHERE ENO = :NEW.ENO;  
 IF CTR > 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20003,'Duplication not possible');  
 END IF;  
END;**

**Command to display Trigger names**

**SELECT TRIGGER\_NAME FROM USER\_TRIGGERS;**

**Command to disable a Trigger**

**Eg.**

**ALTER TRIGGER T1 DISABLE;**

**Command to enable a Trigger**

**Eg.**

**ALTER TRIGGER T1 ENABLE;**

**Command to drop a Trigger**

**Eg.**

**DROP TRIGGER T1;**