PL /SQL

# INTRODUCTION

* PL/SQL stand for Procedural Language / Structured Query Language
* An extension to SQL with design features of programming languages.
* Allows user to write a program to do the various operations on databases.
* PL/SQL is a combination of SQL statements with the programming Language.

# DIFFERENCES BETWEEN SQL AND PL/SQL

* SQL is a nonprocedural language. When you issue a SQL command, your command tells the database server what to do. However, you cannot specify how to do it.
* PL/SQL integrates control statements and conditional statements with SQL. This gives you better control of your SQL statements and their execution.

# ADVANTAGES OF PL/SQL

#### Procedural Language Capability

* + PL/SQL consists of Procedural language such as conditional statement (if statement ) and loops like (FOR loop).

#### Modularized Program Development

* + The basic unit in a PL/SQL program is a block. All PL/SQL programs consist of blocks.
  + These blocks can be thought of as modules and can be “modularized” in a sequence or nested in other blocks.
  + You can break your application into smaller modules. If you are designing a complex application, PL/SQL allows you to break down the application into smaller, manageable, and logically related modules.
  + You can easily read, maintain, and debug the code.

#### Improved Performance

* + PL/SQL allows you to logically combine multiple SQL statements as one unit or block.
  + PL/SQL engine Processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic, and provides high performance for the application.

#### Portability

* + Applications written in PL/SQL are portable to any computer hardware and operating systems.
  + You can write portable program packages and create libraries that can be reused on Oracle databases in different environments.

#### Exception Handling

* + An exception is an error that occurs in the database or in a user‟s program during runtime.
  + Examples of errors include: hardware or network failures, application logic errors, data integrity errors, and so on.
  + You can prepare for errors by writing exception handling code.
  + Exception handling code tells your program what to do in the event of an exception.
  + PL/SQL allows you to handle database and program exceptions efficiently. You can define separate blocks for dealing with exceptions.

# PL/SQL BLOCK STRUCTURE

PL/SQL blocks contain three sections

1. Declare section
2. Executable section and
3. Exception-handling section.

**Declare Section** - This is optional section starts with key word „declare‟. Declare section is used to declare variables, constants, cursor etc.

**Execution Section** – This Section is Compulsory (mandatory). This section is starts with keyword „begin‟ and ends with „end‟. The Program‟s Logic is written in this section.

**Exception Handling Section**- This section is optional and starts with the keyword „exception‟. It is used to write the code for handling the errors in the program.

**DECLARE (Optional)**

/\*Variables, cursors, user-defined exceptions

**BEGIN (Mandatory)**

/\* SQL statement, PL/SQL statements

**EXCEPTION (Optional)**

/\* Actions to perform when errors occur

**END; (Mandatory)**

# PL/SQL EXECUTION ENVIRONMENT

ORACLE ENGINE

PL/SQL ENGINE

SQL STATEMENT EXECUTOR

PL/SQL BLOCK OF CODE

**DECLARE**

Procedural Statements;

**BEGIN**

Procedural Statements; SQL Statements; **EXCEPTIONS**

SQL Statements:

**END;**

**VARIABLES AND DATATYPES**

Variables are entities which stores certain type of values. Variables must be declared before its use.

#### PL/SQL Variable Types

* 1. Scalar (char, varchar2, number, date, etc.)
  2. Composite (%rowtype)
  3. Reference (pointers)
  4. LOB (large objects)

**PL/SQL DATATYPES**

1. **Char(size):** This data type is used to store strings values of fixed length. The size in brackets determines the number of characters the cell can hold. It can hold maximum 255 characters**.**

#### Example : char(50)

1. **varchar (size)/varchar2(size):** This data type is used to store variable length alphanumeric data. The maximum this data type can hold up to 4000 characters. **Example : varchar2(80)**
2. **Date**: This data type is used to represent date and time. The Default format is : DD-MM-YY

#### Example : ‘20-oct-14’

1. **Number(p,s):** This data type is used to store fixed or floating point numbers. P is the precision & S specifies scale. The maximum precision is 38 digits.

#### Example : number(5,2) , number(5).

1. **Long:** This data type is used to store variable length character strings containing up to 2 GB. Only one long Column allowed per Table.
2. **RAW :-** This data type is used to store Binary Data such as Digitized, Picture or Image Maximum storage up to 255 bytes.
3. **RAW LONG** : - This data type is used store Binary data such as as Digitized, Picture or Image Maximum storage up to 2 GB.

# TYPES OF LARGE OBJECT

1. **BLOB** –This Data type is used to store Binary Character Like songs, videos, Images maximum size up to 4 GB.
2. **CLOB** – This data type is used to store character value with size up to 4 GB.

# REFERENCE VARIABLES

* + Reference variables directly reference specific database column or row
  + Reference variables assume data type of associated column or row
  + %TYPE data declaration syntax:

#### variable\_name tablename.fieldname%TYPE;

* + The (%TYPE) reference data type specifies a variable that references a single Database field.

#### Example:

**Empid emp.empno%TYPE;**

the empid variable assumes a data type of VARCHAR2(30), because this is the data type of the empno in the EMP table.

# GENERATING OUTPUT

SET SERVEROUTPUT ON;

Put the command at the beginning of the program, right before the declaration section.

# DISPLAYING OUTPUT

## Syntax:

DBMS\_OUTPUT.PUT\_LINE(<string>);

In which PUT\_LINE is the procedure to generate the output on the screen, and DBMS\_OUTPUT is the package to which the PUT\_LINE belongs.

**Example:** DBMS\_OUTPUT\_PUT\_LINE(„My age is „ || num\_age);

# HINTS AND GUIDELINES

All variables must be declared before their use. Variable\_name datatype(size);

Variable\_name datatype(size) :=value;

The assignment statement

: =

Is not the same as the equality operator

=

All statements end with a semi-colon (;).

To run the PL/SQL block, you may need to Type „/‟ at the End of PL /SQL block. When PL/SQL is executed it prompts, the output.

# EXAMPLE OF A PL/SQL PROGRAM

SQL > SET SERVEROUTPUT ON;

BEGIN

dbms\_output.put\_line(„Welcome To PL/SQL Programming‟);

END;

/

#### Output:

Welcome To PL/SQL Programming PL/SQL procedure successfully completed.

## Program for addition of two numbers

SQL > SET SERVEROUTPUT ON;

SQL> declare

1. a number;
2. b number;
3. c number;
4. begin
5. a := &a;
6. b:= &b;
7. c:= a+b;
8. dbms\_output.put\_line('Addition is'||c);
9. end;
10. /

### OUTPUT

Enter value for a: 2

old 6: a := &a; new 6: a := 2;

Enter value for b: 3 old 7: b:= &b;

new 7: b:= 3;

Addition is 5

PL/SQL procedure successfully completed.

## Program for calculating area of a circle

SQL> declare

1. radius number(3);
2. pi number:=3.14;
3. area number(8,3);
4. begin
5. radius:=& radius;
6. area:= pi \* power(radius,2);
7. dbms\_output.put\_line('area of circle is' || area);
8. end;
9. /

### OUTPUT

Enter value for radius: 2 old 6: radius:=& radius; new 6: radius:=2;

area of circle is12.56

## Program to increase the salary of employee from emp table

SQL> declare

1. newsal emp.sal%type;
2. val number;
3. begin
4. val:=& val;
5. update emp set sal= sal + val;
6. dbms\_output.put\_line( 'the salary is increased by' || val);
7. select sal into newsal from emp where ename= 'ajay';
8. dbms\_output.put\_line('salary of ajay is'|| newsal);
9. end;
10. /

### OUTPUT

Enter value for val: 200 old 5: val:=& val; new 5: val:=200;

the salary is increased by200 salary of ajay is12900

PL/SQL procedure successfully completed.

#### Write PL/SQL program to display square of any number.

declare

no number(10); sqr number:=1; begin no:=&no; sqr:=no\*no;

dbms\_output.put\_line(„Square of a number „ ||no||‟is‟||sqr); end;

# CONTROL STRUCTURES

Control structure is used for decision making and changing the control flow of the program.

1. Conditional Control
   1. IF - THEN
   2. IF – THEN- ELSE
   3. IF –THEN –ELSE –IF
2. Iterative Control
   1. LOOP
   2. While LOOP
   3. FOR LOOP
3. Sequential Control
   1. GOTO Statement

## Conditional Control

### IF -THEN

* An IF-THEN statement allows you to specify only one group of actions to take.
* The simplest form of IF statement associated a condition with a sequence of statement enclosed by the keyword THEN and END IF
* If condition is TRUE then statement get execute, if False then if statement does nothing.

#### Syntax:

IF <condition> then Statements;

End IF;

#### Program to find out whether second number is larger from two given numbers.

SQL> declare

1. a number:=10;
2. b number:=20;
3. begin
4. if b > a then
5. dbms\_output.put\_line(' B is Big ');
6. end if;
7. end;
8. /

B is Big

PL/SQL procedure successfully completed.

### IF – THEN- ELSE

* If the value of *condition* is true, the *statements* run; otherwise, the *else\_statements* run

#### Syntax:

IF <condition > then statements;

Else statements; End if;

#### 1. Program to find out the largest between two numbers

Declare

A number(3);

B number(3); Begin A:=&A;

B:=&B;

If (A>B) then

Dbms\_output.put\_line(„ Greater number=„||A); else

Dbms\_output.put\_line(„ Greater number=„||B); End if;

End;

/

### OUTPUT

Enter value for a: 10 old 5: A:=&A;

new 5: A:=10;

Enter value for b: 20 old 6: B:=&B;

new 6: B:=20;

Greater number=20

PL/SQL procedure successfully completed

#### Program to find out if the entered character is a vowel or a consonant.

SQL> declare

1. ch varchar2(30);
2. begin
3. ch:='&ch';
4. if (ch=„a' or ch='i' or ch='e' or ch='o' or ch='u')
5. then
6. dbms\_output.put\_line(ch||'is a vowel');
7. else
8. dbms\_output.put\_line(ch||'is a not vowel');
9. end if;
10. end;
11. /

### OUTPUT

Enter value for ch: i old 4: ch:='&ch';

new 4: a:=„i'; i is a vowel

PL/SQL procedure successfully completed.

### IF –THEN–ELSE –IF

* The IF THEN ELSE IF statement runs the first *statements* for which *condition* is true. Remaining conditions are not evaluated.
* If no *condition* is true, the *else\_statements* run

Syntax :

IF < condition 1> Then statements;

ELSE IF < condition 2> Then statements;

ELSE IF < condition n> Then statements;

ELSE

statements;

END IF;

END IF;

END IF;

#### Program to find out the largest among three numbers.

SQL> declare

1. A number;
2. B number;
3. C number;
4. begin
5. A :=&a;
6. B :=&b;
7. C :=&c;
8. if ( A > B) and (A >C) then
9. dbms\_output.put\_line(' A IS BIG ');
10. else if ( B > A) and (B >C) then
11. dbms\_output.put\_line(' B IS BIG ');
12. else
13. dbms\_output.put\_line(' C IS BIG ');
14. end if;
15. end if;
16. end;

17 /

### OUTPUT

Enter value for a: 20 Enter value for b: 30 Enter value for c: 40 C IS BIG

PL/SQL procedure successfully completed.

## Iterative Control

* Iterative control Statements are used when we want to repeat the execution of one or more statements for specified number of times.

1. LOOP
2. WHILE LOOP
3. FOR LOOP

# LOOP

* + A Simple Loop is used when a set of statements is to be executed at least once before the loop terminates.
  + An EXIT condition must be specified in the loop, When the EXIT condition is satisfied the process exits from the loop.

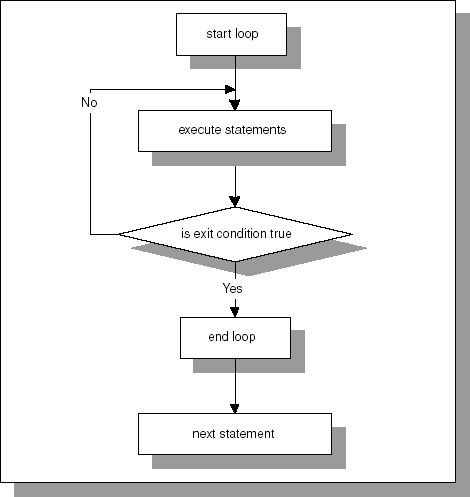
#### Syntax :

LOOP

statements;

EXIT WHEN condition; END LOOP;

### FLOWCHART OF LOOP

****

#### Program to print numbers from 1 to 5.

SQL> declare

1. i number:=1;
2. begin
3. loop
4. dbms\_output.put\_line(i);
5. i:=i+1;
6. EXIT WHEN i >5;
7. END LOOP;
8. end;
9. / **OUTPUT** 1

2

3

4

5

PL/SQL procedure successfully completed.

#### Program to print numbers from 20 to 15 in reversed order.

SQL> declare

2 i number(10):=20;

1. begin
2. loop
3. dbms\_output.put\_line(i);
4. i:=i-1;
5. exit when i<15;
6. end loop;
7. end;
8. / **OUTPUT** 20

19

18

17

16

15

#### Program to print even numbers from 1 to 10.

SQL> declare

1. i number:=1;
2. begin
3. loop
4. if (mod(i,2)=0) then
5. dbms\_output.put\_line(i);
6. else
7. null;
8. end if;
9. i:=i+1;
10. exit when i>10;
11. end loop;
12. end;
13. /

### OUTPUT

2

4

6

8

10

PL/SQL procedure successfully completed

## While Loop

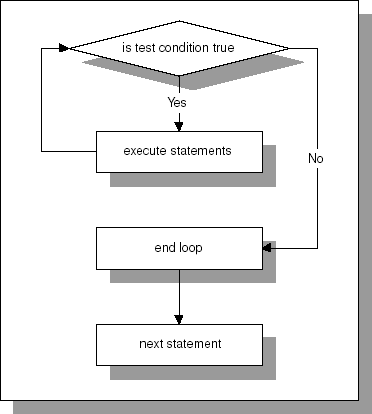
* + - A WHILE LOOP is used when a set of statements has to be executed as long as a condition is true.
    - The condition is evaluated at the beginning of each iteration. The iteration continues until the condition becomes false

#### Syntax:

WHILE <condition> LOOP statements;

END LOOP;

### FLOWCHART FOR WHILE LOOP

****

#### Program to print numbers from 1 to 5

SQL> declare

1. a number :=1;
2. begin
3. while a <= 5 loop
4. dbms\_output.put\_line(a);
5. a:=a+1;
6. end loop;
7. end;
8. /

OUTPUT 1

2

3

4

5

PL/SQL procedure successfully completed.

#### Program to find the factorial of a number

SQL> declare

2 i number(10):=1;

1. fact number:=1;
2. num number(10);
3. begin
4. num:=&num;
5. while (i<=num) loop
6. fact:=fact\*i;
7. i:=i+1;
8. end loop;
9. dbms\_output.put\_line(fact);
10. end;
11. /

### OUTPUT

Enter value for num: 5 old 6: num:=&num;

new 6: num:=5;

120

PL/SQL procedure successfully completed.

## For Loop

* + - A FOR LOOP is used to execute a set of statements for a predetermined number of times.
    - Iteration occurs between the start and end integer values given.

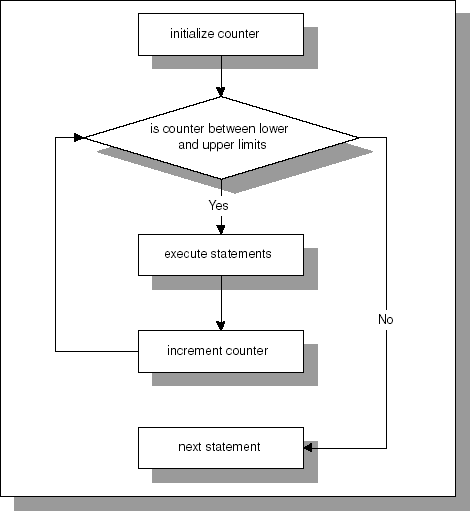
#### Syntax:

FOR counter IN [reverse] val1..val2 LOOP statements;

END LOOP;

#### val1 - Start integer value. val2 - End integer value.

FLOWCHART of FOR LOOP



#### Program to print numbers from 1 to 5

SQL> declare

1. a number :=1;
2. begin
3. for a in 1..5 loop
4. dbms\_output.put\_line(a);
5. end loop;
6. end;
7. /

### OUTPUT

1

2

3

4

5

PL/SQL procedure successfully completed.

#### Program to print sum of numbers from 1 to 10.

SQL> declare

1. i number(10);
2. ans number(20):=0;
3. begin
4. for i in 1..10 loop
5. ans:= ans + i;
6. end loop;
7. dbms\_output.put\_line(ans);
8. end;
9. /

### OUTPUT

55

PL/SQL procedure successfully completed.

#### Program to print numbers from 1 to 10 in reversed order.

SQL> declare

1. i number(10);
2. begin
3. for i in reverse 1..10 loop
4. dbms\_output.put\_line(i);
5. end loop;
6. end;
7. /

### OUTPUT

10

9

8

7

6

5

4

3

2

1

## Sequential Control

* The Goto to statement immediately transfer Program control unconditionally ( Without checking Conditon ) to Label( Jump Location) Statement .
* Goto to statement can not Branch into IF Statement, Loop statement.
* Changes the flow of control within the block.
* Entry point is marked using tag:

#### <<userdefined name>

* **GOTO** Statement is used to jump to this label.

#### Syntax:

Begin Statements;

#### GOTO Lablel\_1;

Statements;

**<<Lablel\_1>>** Statements; END;

#### Write a program to print the numbers 1 to 5 , and then print ,message as number is 5 label is displayed

SQL> declare

1. i number;
2. begin
3. for i in 1..10 loop
4. dbms\_output.put\_line(i);

#### if (i=5) then

1. **goto label1;**
2. else
3. null;
4. end if;
5. end loop;

#### <<label1>>

1. **dbms\_output.put\_line('As i=5, label1 is displayed');**
2. end;
3. / **OUTPUT** 1

2

3

4

5

As i=5, label1 is displayed

# EXCEPTION HANDLING

* Exception is nothing but an error.
* When the system throws a warning or has an error it can lead to an exception.
* Such exception needs to be handled and can be defined internally or user defined.
* The Exception section start with the keyword „EXCEPTION‟ and it is optional.
* Any ERROR in the program are handled in this section.

**Syntax:**

### DECLARE

Declaration section

#### Begin

Executable statement;

### EXCEPTION

WHEN ex\_name1 THEN

-Error handling statements/user defined action to be carried out;

### END;

#### Types of Exception

1. Predefined Exception Or System Defined Exceptions
2. User Defined Exception

## Predefined Exception Or System Defined Exceptions

* + - System exception are automatically raised by Oracle, when a Program violates a RDBMS rules.
    - The most common errors that can occurs during the execution of PL/SQL i.e. no data found, zero divide and too many rows etc.

## ZERO\_DIVIDE Exception

* + The ZERO\_DIVIDE Exception (ORA-01476) occurs when a program attempts to divide a number by zero

## Handling of Zero Divide Exception

SQL> declare

1. a number;
2. b number;
3. c number;
4. begin
5. a:=&a;
6. b:=&b;
7. c:= a/b;
8. dbms\_output.put\_line(c);

#### exception

1. when zero\_divide then
2. dbms\_output.put\_line('divisor is zero');
3. end;
4. /

### OUTPUT

Enter value for a: 6 old 6: a:=&a;

new 6: a:=6; Enter value for b: 0 old 7: b:=&b;

new 7: b:=0; divisor is zero

PL/SQL procedure successfully completed.

## Table ‘course’ for demonstration of further examples.

|  |  |  |  |
| --- | --- | --- | --- |
| **CID** | **SUB** | **DU48RATION** | **SID** |
| C001 | RDM | 48 | 1 |
| C002 | VB | 45 | 2 |
| C003 | DSU | 34 | 3 |
| C004 | PIC | 40 | 4 |

1. **NO\_DATA\_FOUND Exception**
   * It is raised when a SELECT INTO statement returns no rows.

## Handling NO\_DATA\_FOUND exception.

SQL> declare

1. c\_id course.cid%type;
2. begin
3. select cid **into** c\_id from course where sub='VBS';
4. Exception
5. when **NO\_DATA\_FOUND** then

7dbms\_output.put\_line(' **Data is No Found by Select Query**');

1. end;
2. /

### OUTPUT

Data is No Found by Select Query PL/SQL procedure successfully completed.

## TOO\_MANY\_ROWS Exception

* + It is raised when s SELECT INTO statement returns more than one row.

## Handling TOO\_MANY\_ROWS exception.

SQL> declare

1. c\_id course.cid%type;
2. begin
3. select cid into c\_id from course;
4. exception
5. when TOO\_MANY\_ROWS then
6. dbms\_output.put\_line(' Many Rows are Fetched. ');
7. end;
8. /

### OUTPUT

Many Rows are Fetched....

PL/SQL procedure successfully completed.

# USER DEFINED EXCEPTIONS

* + A user Define exception has to be defined by the Programmer.
  + User Define exception is declared in the Declaration section with their type as
  + Two Method to Define user define Exception .
    1. Raise
    2. RAISE\_APPLICATION\_ERROR

**Syntax: DECLARE**

### BEGIN

**EXCEPTION**

#### <exception\_name> exception ;

<SQL Sentences>;

if <condition> Then

#### Raise <exception\_name>

End if;

#### WHEN <exception\_name> THEN

-Error handling statements/user defined action to be carried out;

### END;

#### E.g. Program to Check the Duration if it is Less than 40 then raise user defined exception

SQL> declare

1. my\_exe Exception;
2. D1 course.duration%type;
3. begin
4. select duration into d1 from course where sid=3;
5. if d1 <40 then
6. raise my\_exe;
7. end if;
8. exception
9. when my\_exe then
10. dbms\_output.put\_line(' Duration is less than 40');
11. end;
12. /

### OUTPUT

Duration is less than 40

PL/SQL procedure successfully completed.

# RAISE\_APPLICATION\_ERROR

* + To display your own error message one can use the bulit in RAISE\_APPLICATION\_ERROR.
  + Use negative number between **-20000 to -20999** for the Error\_number and the error message should not exceed **512** character.

**Syntax :**

**RAISE\_APPLICATION\_ERROR( error\_number,’error message**‟);

**Example- IF data is not found for SID=6 then use Raise\_application\_error to Display message**

SQL >declare

1. dur course.duration%type;
2. begin
3. select duration into dur from course where sid=6;
4. Exception
5. when NO\_DATA\_FOUND then

#### Raise\_application\_error(-20100,'NO Data Found');

1. **end;**
2. /

#### Output:

ERROR at line 1:

ORA-20100: NO Data Found ORA-06512: at line 6

PL/SQL procedure successfully completed.

#### Example- IF divisor is zero then use Raise\_application\_error to Display message .

SQL> declare

b number;

#### begin

b:= 5/0;

#### exception

when zero\_divide then

#### raise\_application\_error(-20202,'divisor is zero');

end;

/

ERROR at line 1:

#### ORA-20202: divisor is zero

ORA-06512: at line 4

PL/SQL procedure successfully completed.

### CURSORS

* PL/SQL Cursor is Pointer that Points the results set of the SQL Query Against database Table
* Cursor is a temporary work area created in the system memory when SQL statement is executed.
* It is used to store the data retrieved from the database and manipulate this data.
* The data stored in the cursor is called as Active Data Set.
* A cursor can hold more than one row but can process only one row at a time.
* Cursor is used with select Statement and store result.

**Types of Cursor. 1)Implicit**

* 1. **Explicit**

1. **Implicit Curcors**
   * Implicit cursors are created for every query made in Oracle Like Delete ,Update, Insert

,Select etc .

* + When we write command ,we use implicit cursors.
  + They are system generated cursors.
  + Oracle will perform the open, fetches, and close for you automatically.
  + Implicit cursors are used in statements that return only one row.
    - The most recently opened cursor is called the “SQL%” Cursor.

### ATTRIBUTES OF IMPLICIT CURSOR

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Format** |
| **%FOUND** | If SELECT statement return one or more rows or DML statement (INSERT, UPDATE, DELETE) affect one or more rows . IT return TRUE otherwise return FALSE. | SQL%FOUND |
| **%NOTFOUND** | No rows found then Return TRUE Else Return False . | SQL%NOTFOUND |
| **%ISOPEN** | Oracle engine automatically open the cursor If cursor open return TRUE otherwise return FALSE. | SQL%ISOPEN |
| **%ROWCOUNT** | Count the Row Affected by DML operations | SQL%ROWCOUNT |

|  |  |  |
| --- | --- | --- |
| **Consider table** | **student1;** |  |
| **NO** | **NAME** | **MARKS** |
| **----------** | **--------------------** | **----------** |
| **1** | **akash** | **85** |
| **2** | **nikita** | **65** |
| **3** | **puja** | **75** |
| **4** | **nisha** | **35** |

#### Count the No of Row Deleted from table using rowcount.

SQL> begin

1. delete from student1 where name='puja';
2. dbms\_output.put\_line('Number of rows deleted:='||sql%rowcount);
3. end;
4. /

**Number of rows deleted:=1**

**PL/SQL procedure successfully completed.**

1. **Implicit cursors with update statement**

SQL> begin

1. update student1 set marks=55 where no=4;
2. dbms\_output.put\_line('Number of rows updated:='||sql%rowcount);
3. end;
4. /

**Number of rows updated:=1**

**PL/SQL procedure successfully completed.**

1. **Use of sql%found and sql%notfound attribute**

SQL> declare

1. rowcnt number(11);
2. begin
3. update student1 set marks=marks+1;
4. if (sql%found) then
5. rowcnt:=sql%rowcount;
6. dbms\_output.put\_line('Number of rows updated:='||rowcnt);
7. else if (sql%notfound) then
8. dbms\_output.put\_line('no rows updated');
9. end if;
10. end if;
11. end;
12. /

**Number of rows updated:=4**

**PL/SQL procedure successfully completed**.

1. **Explicit Cursor.**
   * Cursors can be declared by a programmer within PL/SQL
   * They must be created when you are executing a SELECT statement that returns more than one row.
   * Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row.
   * When you fetch a row the current row position moves to next row.
   * Cursor Is First Declared then Open, Once the Cursor open Fetch Data From Cursor, Finally Close the Cursor.

#### Steps for Creating Explicit Cursors

1. **Declare Cursor**
   * Defines name and structure of cursor.

#### Syntax:

**Example:**

Cursor <cursor\_name> IS Select statement ; Cursor c1 is select name,marks from student1;

#### Open Syntax:

**Example:**

Open <Cursor\_name>; open c1;

#### Fetch Data from cursor Syntax:

**Example:**

fetch <Cursor\_name>into <Variable\_name>; fetch c1 into var1;

#### Close Syntax:

**Example:**

close <Cursor\_name>; close c1;

#### General Syntax :

**declare**

Variable Declaration;

**cursor <name\_cursor> IS select statement ; begin**

**open <name\_cursor>;**

**fetch <name\_cursor > into <Variable\_name>; Close <name\_cursor>**

**end;**

**/**

Consider table student1;

|  |  |  |
| --- | --- | --- |
| NO | NAME | MARKS |
| 1 | akash | 85 |
| 2 | nikita | 65 |
| 3 | puja | 75 |
| 4 | nisha | 35 |

#### Display name and marks from student1 table using cursors

SQL> declare

* 1. cursor cs1 is select name,marks from student1;
  2. info cs1%rowtype;
  3. begin
  4. open cs1;
  5. loop
  6. fetch cs1 into info;
  7. exit when cs1%notfound;
  8. dbms\_output.put\_line(info.name || info.marks);
  9. end loop;
  10. close cs1;
  11. end;
  12. /

akash 85

nikita 65

puja 75

nisha 35

PL/SQL procedure successfully completed.

### PROCEDURE

* A procedure is **named** PL/SQL block perform one or more specific task .
* Procedure consist of **Header** and **Body .**

**Header** – Contain **name of Procedure** and **Parameter Passed** to it.

**Body** - Contain **Declaration ,Execution, Exception section** Same AS PL/SQL

#### Syntax :

**Create OR Replace Procedure < name\_Procedure>**

[ Parameter name [IN | OUT | IN OUT] Data\_type ] [ IS | AS ]

Variable declaration;

Constant declarations;

#### Begin

Pl/sql subprogram;

#### END < name\_Procedure> ;

**Passing Parameter to Procedure by Following Way :**

1. **IN** - This Parameter are used to send the Value to Procedure .
2. **Out** - This Parameter are used to Get the Value From Procedure .
3. **IN OUT** - This Parameter are used to get and send the Value to Procedure if IN and OUT data type is same

#### Write a program to insert the record into table through procedure

SQL> create or replace procedure ins21(no1 in number,name1 in varchar2,marks1 in number) as

1. begin
2. insert into student1 values(no1,name1,marks1);
3. dbms\_output.put\_line(„record inserted‟);
4. end ins21;
5. /

#### Procedure created.

**Calling of procedure**

SQL> exec ins21(12,'punam',75); record inserted

PL/SQL procedure successfully completed

#### Write a program to delete the record from table through procedure

SQL> create or replace procedure select3(name1 in varchar) as

1. begin
2. delete from student1 where name=name1;
3. end select3;
4. /

#### Procedure created.

**Calling of procedure**

SQL> exec select3('puja');

PL/SQL procedure successfully completed

#### Write PL/SQL procedure to calculate factorial of a given number.

SQL> create or replace procedure Factorial(n in number) is

v number :=1; begin

for i in 1..n loop

v :=v \* i; end loop;

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

Procedure created

#### Execution of procedure

SQL> Begin Factorial(5); End;

#### Deleteting a Proceduer

* To delete Procedure use Following Command DROP Procedure <Procedure\_name>;

### FUNCTION

* Function is named PL/SQL Block that accept the value and Return only one value .
* It is used for code reusability.

#### Syntax:

**create or replace function <Function\_name>(Parameter in Data type )**

return Data Type [IS | AS]

Variable declaration; Constant declarations;

#### Begin

Pl/sql subprogram;

**return Value ;**

**end <Funcation\_name >;**

1. **Write a program to Find Square of a number using function in PL/SQL**

SQL> create or replace function sqr( no in number ) return number

1. AS
2. begin
3. return no\*no;
4. end sqr;
5. /

Function created.

#### Calling of function through program

SQL> declare

1. val1 number;
2. begin
3. val1 :=&val1;
4. dbms\_output.put\_line(„Square‟ || sqr(val1));
5. end;
6. /

Enter value for val1: 4 Square 16

#### Write a program area of circle using function in PL/SQL

SQL> create or replace function area20(r1 in number) return number as

1. pi number :=3.14;
2. area number(10);
3. begin
4. area:=pi\*r1\*r1;
5. return area;
6. end area20;
7. /

Function created.

#### Calling of function through program

SQL> declare

1. a number(10);
2. begin
3. a:=area20(2);
4. dbms\_output.put\_line('area is'||a);
5. end;
6. / area is13

PL/SQL procedure successfully completed.

#### Write a program for addition of two numbers using function in PL/SQL

SQL> create or replace function add1(a in number,b in number) return number as

1. c number(10);
2. begin
3. c:=a+b;
4. return c;
5. end add1;
6. /

Function created.

#### Calling of function through program

SQL> declare

1. result number(20);
2. begin
3. result:=add1(10,30);
4. dbms\_output.put\_line('addition is'||result);
5. end;
6. /

addition is 40

PL/SQL procedure successfully completed**.**

#### Write a function which will count the number of rows deleted

SQL> create or replace function del(name1 in varchar2)return number as

1. no number(10);
2. begin
3. delete from mp1 where name=name1;
4. no:=sql%rowcount;
5. return no;
6. end del;
7. /

Function created.

#### Calling of function through program

SQL> declare

1. a number;
2. begin
3. a:=del(„manas');
4. dbms\_output.put\_line(„number of rows deleted :‟||a);
5. end;
6. /

number of rows deleted :1

PL/SQL procedure successfully completed.

## Difference between procedures and functions

|  |  |
| --- | --- |
| **Functions** | **Procedures** |
| A function must return a value back to caller. | Procedure may or may not be return the value. |
| A function return only one value to the calling PL/SQL block | By defining the OUT parameters multiple values can be passed to the procedure. |
| Will compiled at run time | Stored in database in compiled format. |

### ADVANTAGES OF USING A PROCEDURE AND FUNCTION

#### Advantages of Function:

1. Code reusability feature can be used.
2. It saves time and cost.
3. Increases flexibility of the program.
4. Memory space required is less.
5. It can return value to the calling program.

#### Advantages of Procedure:

1. Security: Procedures offer more security.
2. Productivity: Avoids redundant code for common procedures in multiple applications.
3. Memory savings: Requires only one copy of the code for multiple users.
4. Performance: Precompiled code hence no compilation is required to execute a code.
5. Integrity and accuracy: As procedure is needed to be tested only once hence guarantee of accurate result.

### DATABASE TRIGGER

* + **Database Trigger** is PL/SQL Block that is executed on an Event in the Database.
  + It is fired when DML statement like insert,update,delete executed.
  + **Event** - Data Manipulation of a Table such as Inserting, Deleting or Updating a Row of Table.
  + Trigger is automatically fire (executed) before or after an INSERT, DELETE, and UPDATE etc.

#### Types of PL/SQL Triggers

There are two types of triggers based on the which level it is triggered.

1. **Row level trigger** - An event is triggered for each row upated, inserted or deleted.
2. **Statement level trigger** - An event is triggered for each sql statement executed.

**Syntax for Creating a Trigger**

**CREATE [OR REPLACE ] TRIGGER <trigger\_name >**

### { BEFORE | AFTER}

**{ INSERT [OR] | UPDATE [OR] | DELETE }**

**[OF col\_name] ON <table\_name >**

### [FOR EACH ROW]

#### WHEN (condition) BEGIN

**--- sql statements END;**

**/**

* **CREATE [OR REPLACE ] TRIGGER trigger\_name** - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
* **{BEFORE | AFTER }** - This clause indicates at what time should the trigger get fired.

i.e for example : before(validation purpose) or after updating a table.

* **{INSERT [OR] | UPDATE [OR] | DELETE}** - This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword.
* **[OF col\_name]** - This clause is used when you want to trigger an event only when a

**specific column** is updated.

* **[ON table\_name]** - This clause identifies the name of the **table** to which the trigger is associated.
* **[FOR EACH ROW]** - This clause is used to determine whether a trigger must fire when each row gets affected ( i.e. a Row Level Trigger)
* **WHEN (condition)** - This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

#### 1. Write a program which display pass or fail after insert, update of new marks into student table using trigger

SQL> create trigger studtrigger

1. after insert or update on student1
2. for each row
3. begin
4. if(:new.marks<40) then
5. dbms\_output.put\_line('fail');
6. else
7. dbms\_output.put\_line('pass');
8. end if;
9. end;
10. /

Trigger created

SQL> insert into student1 values(10,'neha',30);

#### fail

1. row created.

SQL> insert into student1 values(10,'neeta',80);

#### pass

1. row created.

SQL> update student1 set marks=46 where marks=20;

**pass**

### EXAMPLE:

The price of a product changes constantly. It is important to maintain the ***history of the prices*** of the products.

So we can create a trigger to Maintain the price history into **'product\_price\_history'** table when the price of the **product** is updated in the **'product'** table.

#### Create 'product' table in which new value is update

*CREATE TABLE product*

*(product\_id number(5), product\_name varchar2(32), supplier\_name varchar2(32), unit\_price number(7,2) );*

#### Create the 'product\_price\_history' table which maintain the history of Price from ‘Product ‘ table ......

*CREATE TABLE product\_price\_history (product\_id number(5), product\_name varchar2(32), supplier\_name varchar2 (32), unit\_price number(7,2) );*

1. **Create the price\_history\_trigger and execute it.** *CREATE or REPLACE TRIGGER price\_history\_trigger BEFORE UPDATE OF unit\_price*

*ON product*

*FOR EACH ROW BEGIN*

*INSERT INTO product\_price\_history VALUES*

*(:old.product\_id,*

*:old.product\_name,*

*:old.supplier\_name,*

*:old.Unit\_price); END;*

***/ <- Used to execute the Trigger***

#### Lets update the price of a product.

UPDATE PRODUCT SET unit\_price = 25 WHERE product\_id = 1;

Once the above *update query* is executed, the trigger fires automatically and updates the '**product\_price\_history**' table.

#### Use of database triggers

* 1. Triggers can be used to prevent invalid transaction and to apply complex security authorization.
  2. It can be used to various integrity constraint.
  3. It can be used to check data modifications**.**

#### Comparison of procedures and triggers

|  |  |
| --- | --- |
| **Database Trigger** | **Procedure** |
| Triggers are fired when particular SQL  command(DML) is executed | Procedures are executed when they are  called |
| Triggers have event and actions | Procedures do not have event and  actions |
| Triggers are called implicitly | Procedures are called explicitly |

**Remove the Trigger**

SQL> Drop Trigger <Trigger\_name>;

#### Example

SQL> DROP TRIGGER studtrigger ; Trigger dropped.