UNIT – 4

PL sql

PL/SQL is a block structured language that can have multiple blocks in it.

What is PL/SQL

PL/SQL is a block structured language.

The programs of PL/SQL are logical blocks that can contain any number of nested sub-blocks.

Pl/SQL stands for "Procedural Language extension of SQL" that is used in Oracle.

PL/SQL is integrated with Oracle database (since version 7).

The functionalities of PL/SQL usually extended after each release of Oracle database.

Although PL/SQL is closely integrated with SQL language, yet it adds some programming constraints that are not available in SQL.

PL/SQL Functionalities

PL/SQL includes procedural language elements like conditions and loops.

It allows declaration of constants and variables, procedures and functions, types and variable of those types and triggers.

It can support Array and handle exceptions (runtime errors).

After the implementation of version 8 of Oracle database have included features associated with object orientation.

You can create PL/SQL units like procedures, functions, packages, types and triggers, etc.

which are stored in the database for reuse by applications.

With PL/SQL, you can use SQL statements to manipulate Oracle data and

flow of control statements to process the data.

The PL/SQL is known for its combination of data manipulating power of SQL with data processing power of procedural languages.

It inherits the robustness, security, and portability of the Oracle Database.

PL/SQL is not case sensitive so you are free to use lower case letters or upper case letters except within string and character literals.

A line of PL/SQL text contains groups of characters known as lexical units. It can be classified as follows:

- Delimeters
- Identifiers

- Literals
- Comments

PL/SQL Variables

A variable is a meaningful name which facilitates a programmer to store data temporarily during the execution of code.

It helps you to manipulate data in PL/SQL programs. It is nothing except a name given to a storage area. Each variable in the PL/SQL has a specific data type which defines the size and layout of the variable's memory.

A variable should not exceed 30 characters. Its letter optionally followed by more letters, dollar signs, numerals, underscore etc.

1. It needs to declare the variable first in the declaration section of a PL/SQL block before using it.

2. By default, variable names are not case sensitive. A reserved PL/SQL keyword cannot be used as a variable name.

How to declare variable in PL/SQL

You must declare the PL/SQL variable in the declaration section or in a package as a global variable. After the declaration, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

Syntax for declaring variable:

Following is the syntax for declaring variable:

1. variable_name [CONSTANT] datatype [NOT NULL] [:= | DEF AULT initial_value]

   Here, variable_name is a valid identifier in PL/SQL and datatype must be valid PL/SQL data type. A data type with size, scale or precision limit is called a constrained declaration. The constrained declaration needs less memory than unconstrained declaration.

   Naming rules for PL/SQL variables

   The variable in PL/SQL must follow some naming rules like other programming languages.

   - The variable_name should not exceed 30 characters.
   - The name of the variable must begin with ASCII letter. The PL/SQL is not case sensitive so it could be either lowercase or uppercase. For example: v_data and V_DATA refer to the same variables.
   - You should make your variable easy to read and understand, after the first character, it may be any number, underscore (_) or dollar sign ($).
   - NOT NULL is an optional specification on the variable.

   Initializing Variables in PL/SQL

   Every time you declare a variable, PL/SQL defines a default value NULL to it. If you want to initialize a variable with other value than NULL value, you can do so during the declaration, by using any one of the following methods.

   - The DEFAULT keyword

- The assignment operator

```
1. counter binary_integer := 0;
2. greetings varchar2(20) DEFAULT 'saket-nitish-lavish';
```

You can also specify NOT NULL constraint to avoid NULL value. If you specify the NOT NULL constraint, you must assign an initial value for that variable.

You must have a good programming skill to initialize variable properly otherwise, sometimes program would produce unexpected result.

Example of initilizing variable

Let's take a simple example to explain it well:

```
1. DECLARE
2. a integer := 30;
3. b integer := 40;
4. c integer;
5. f real;
6. BEGIN
7. c := a + b;
8. dbms_output.put_line('Value of c: ' || c);
9. f := 100.0/3.0;
10.     dbms_output.put_line('Value of f: ' || f);
11.     END;
```

After the execution, this will produce the following result:

```
1. Value of c: 70
```

2. Value of f: 33.33333333333333
3.
4. PL/SQL procedure successfully completed.

Variable Scope in PL/SQL:

PL/SQL allows nesting of blocks. A program block can contain another inner block. If you declare a variable within an inner block, it is not accessible to an outer block. There are two types of variable scope:

- Local Variable: Local variables are the inner block variables which are not accessible to outer blocks.
- Global Variable: Global variables are declared in outermost block.

Example of Local and Global variables

Let's take an example to show the usage of Local and Global variables in its simple form:

```
1. DECLARE
2. -- Global variables
3. num1 number := 95;
4. num2 number := 85;
5. BEGIN
6. dbms_output.put_line('Outer Variable num1: ' || num1);
7. dbms_output.put_line('Outer Variable num2: ' || num2);
8. DECLARE
9. -- Local variables
10.      num1 number := 195;
11.      num2 number := 185;
```

```
12.     BEGIN
13.     dbms_output.put_line('Inner Variable num1: ' || num1);
14.     dbms_output.put_line('Inner Variable num2: ' || num2);
15.     END;
16.     END;
17.     /
```

After the execution, this will produce the following result:

```
1. Outer Variable num1: 95
2. Outer Variable num2: 85
3. Inner Variable num1: 195
4. Inner Variable num2: 185
5.
6. PL/SQL procedure successfully completed.
```

PL/SQL Constants

A constant is a value used in a PL/SQL block that remains unchanged throughout the program. It is a user-defined literal value. It can be declared and used instead of actual values.

Let's take an example to explain it well:

Suppose, you have to write a program which will increase the salary of the employees upto 30%, you can declare a constant and use it throughout the program. Next time if you want to increase the salary again you can change the value of constant than the actual value throughout the program.

Syntax to declare a constant:

1. constant_name CONSTANT datatype := VALUE;
    - Constant_name:it is the name of constant just like variable name. The constant word is a reserved word and its value does not change.
    - VALUE: it is a value which is assigned to a constant when it is declared. It can not be assigned later.

Example of PL/SQL constant

Let's take an example to explain it well:

```
1. DECLARE
2. -- constant declaration
3. pi constant number := 3.141592654;
4. -- other declarations
5. radius number(5,2);
6. dia number(5,2);
7. circumference number(7, 2);
8. area number (10, 2);
9. BEGIN
10.     -- processing
11.     radius := 9.5;
12.     dia := radius * 2;
13.     circumference := 2.0 * pi * radius;
14.     area := pi * radius * radius;
15.     -- output
16.     dbms_output.put_line('Radius: ' || radius);
17.     dbms_output.put_line('Diameter: ' || dia);
18.     dbms_output.put_line('Circumference: ' || circumference);
19.     dbms_output.put_line('Area: ' || area);
20.     END;
```

21. /

After the execution of the above code at SQL prompt, it will produce the following result:.

1. Radius: 9.5
2. Diameter: 19
3. Circumference: 59.69
4. Area: 283.53
5.
6. Pl/SQL procedure successfully completed.

PL/SQL Literals

Literals are the explicit numeric, character, string or boolean values which are not represented by an identifier. For example: TRUE, NULL, etc. are all literals of type boolean. PL/SQL literals are case-sensitive. There are following kinds of literals in PL/SQL:

- Numeric Literals
- Character Literals
- String Literals

Literals	Examples
Numeric	75125, 3568, 33.3333333 etc.
Character	'A' '%' '9' ' ' 'z' '('
String	Hello!

Boolean	TRUE, FALSE, NULL etc.
Date and Time	'26-11-2002' , '2012-10-29 12:01:01'

- BOOLEAN Literals
- Date and Time Literals

Example of these different types of Literals:

Control statements

PL/SQL If

PL/SQL supports the programming language features like conditional statements and iterative statements. Its programming constructs are similar to how you use in programming languages like Java and C++.

Syntax for IF Statement:

There are different syntaxes for the IF-THEN-ELSE statement.

Syntax: (IF-THEN statement):

1. IF condition
2. THEN
3. Statement: {It is executed when condition is true}
4. END IF;

This syntax is used when you want to execute statements only when condition is TRUE.

Syntax: (IF-THEN-ELSE statement):

1. IF condition
2. THEN
3. {...statements to execute when condition is TRUE...}
4. ELSE
5. {...statements to execute when condition is FALSE...}
6. END IF;

This syntax is used when you want to execute one set of statements when condition is TRUE or a different set of statements when condition is FALSE.

Syntax: (IF-THEN-ELSIF statement):

1. IF condition1
2. THEN
3. {...statements to execute when condition1 is TRUE...}
4. ELSIF condition2
5. THEN
6. {...statements to execute when condition2 is TRUE...}
7. END IF;

This syntax is used when you want to execute one set of statements when condition1 is TRUE or a different set of statements when condition2 is TRUE.

Syntax: (IF-THEN-ELSIF-ELSE statement):

1. IF condition1
2. THEN
3. {...statements to execute when condition1 is TRUE...}
4. ELSIF condition2

5. THEN
6. {...statements to execute when condition2 is TRUE...}
7. ELSE
8. {...statements to execute when both condition1 and condition2 a re FALSE...}
9. END IF;

It is the most advance syntax and used if you want to execute one set of statements when condition1 is TRUE, a different set of statement when condition2 is TRUE or a different set of statements when both the condition1 and condition2 are FALSE.

When a condition is found to be TRUE, the IF-THEN-ELSE statement will execute the corresponding code and not check the conditions any further.

If there no condition is met, the ELSE portion of the IF-THEN-ELSE statement will be executed.

ELSIF and ELSE portions are optional.

Example of PL/SQL If Statement

Let's take an example to see the whole concept:

```
1. DECLARE
2. a number(3) := 500;
3. BEGIN
4. -- check the boolean condition using if statement
5. IF( a < 20 ) THEN
6. -- if condition is true then print the following
7. dbms_output.put_line('a is less than 20 ' );
```

```
8. ELSE
9. dbms_output.put_line('a is not less than 20 ' );
10.     END IF;
11.     dbms_output.put_line('value of a is : ' || a);
12.     END;
```

After the execution of the above code in SQL prompt, you will get the following result:

```
a is not less than 20
value of a is : 500
PL/SQL procedure successfully completed.
```

PL/SQL Case Statement

The PL/SQL CASE statement facilitates you to execute a sequence of statements based on a selector. A selector can be anything such as variable, function or an expression that the CASE statement checks to a boolean value.

The CASE statement works like the IF statement, only using the keyword WHEN. A CASE statement is evaluated from top to bottom. If it get the condition TRUE, then the corresponding THEN calause is executed and the execution goes to the END CASE clause.

Syntax for the CASE Statement:

```
1. CASE [ expression ]
2. WHEN condition_1 THEN result_1
3. WHEN condition_2 THEN result_2
4. ...
5. WHEN condition_n THEN result_n
6. ELSE result
7. END
```

Example of PL/SQL case statement

Let's take an example to make it clear:

```
1. DECLARE
2. grade char(1) := 'A';
3. BEGIN
4. CASE grade
5. when 'A' then dbms_output.put_line('Excellent');
6. when 'B' then dbms_output.put_line('Very good');
7. when 'C' then dbms_output.put_line('Good');
```

```
8. when 'D' then dbms_output.put_line('Average');
9. when 'F' then dbms_output.put_line('Passed with Grace');
10.    else dbms_output.put_line('Failed');
11.    END CASE;
12.    END;
```

After the execution of above code, you will get the following result:

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

PL/SQL Loop

The PL/SQL loops are used to repeat the execution of one or more statements for specified number of times. These are also known as iterative control statements.

Syntax for a basic loop:

```
LOOP
Sequence of statements;
END LOOP;
```

Types of PL/SQL Loops

There are 4 types of PL/SQL Loops.

1. Basic Loop / Exit Loop
2. While Loop
3. For Loop
4. Cursor For Loop

PL/SQL Exit Loop (Basic Loop)

PL/SQL exit loop is used when a set of statements is to be executed at least once before the termination of the loop. There must be an EXIT condition specified in the loop, otherwise the loop will get into an infinite number of iterations. After the occurrence of EXIT condition, the process exits the loop.

Syntax of basic loop:

```
LOOP
Sequence of statements;
END LOOP;
```

Syntax of exit loop:

```
LOOP
statements;
EXIT;
{or EXIT WHEN condition;}
END LOOP;
```

Example of PL/SQL EXIT Loop

Let's take a simple example to explain it well:

```
DECLARE
i NUMBER := 1;
BEGIN
LOOP
EXIT WHEN i>=10;
DBMS_OUTPUT.PUT_LINE(i);
i := i+1;
END LOOP;
```

```
END;
```

After the execution of the above code, you will get the following result:

```
1
2
3
4
5
6
7
8
9
10
```

Note: You must follow these steps while using PL/SQL Exit Loop.

- Initialize a variable before the loop body
- Increment the variable in the loop.
- You should use EXIT WHEN statement to exit from the Loop. Otherwise the EXIT statement without WHEN condition, the statements in the Loop is executed only once.

PL/SQL EXIT Loop Example 2

```
DECLARE
VAR1 NUMBER;
VAR2 NUMBER;
BEGIN
VAR1:=100;
VAR2:=1;
LOOP
```

```
DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
IF (VAR2=10) THEN
.       EXIT;
.       END IF;
.       VAR2:=VAR2+1;
.       END LOOP;
.       END;
```

PL/SQL While Loop

PL/SQL while loop is used when a set of statements has to be executed as long a
condition is true, the While loop is used. The condition is decided at the beginni
iteration and continues until the condition becomes false.

Syntax of while loop:

```
WHILE <condition>
LOOP statements;
END LOOP;
```

Example of PL/SQL While Loop

Let's see a simple example of PL/SQL WHILE loop.

```
DECLARE
i INTEGER := 1;
BEGIN
WHILE i <= 10 LOOP
DBMS_OUTPUT.PUT_LINE(i);
i := i+1;
END LOOP;
END;
```

After the execution of the above code, you will get the following result:

```
1
2
3
4
5
6
7
8
9
10
```

Note: You must follow these steps while using PL/SQL WHILE Loop.

- Initialize a variable before the loop body.
- Increment the variable in the loop.
- You can use EXIT WHEN statements and EXIT statements in While loop not done often.

PL/SQL WHILE Loop Example 2

```
DECLARE
VAR1 NUMBER;
VAR2 NUMBER;
BEGIN
VAR1:=200;
VAR2:=1;
WHILE (VAR2<=10)
LOOP
DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
```

```
.       VAR2:=VAR2+1;
.       END LOOP;
.       END;
```

Output:

```
200
400
600
800
1000
1200
1400
1600
1800
2000
```

PL/SQL FOR Loop

PL/SQL for loop is used when when you want to execute a set of statements for a predetermined number of times. The loop is iterated between the start and end integer values. The counter is always incremented by 1 and once the counter reaches the value of end integer, the loop ends.

Syntax of for loop:

```
FOR counter IN initial_value .. final_value LOOP
LOOP statements;
END LOOP;
```

- initial_value : Start integer value
- final_value : End integer value

PL/SQL For Loop Example 1

Let's see a simple example of PL/SQL FOR loop.

```
BEGIN
FOR k IN 1..10 LOOP
-- note that k was not declared
DBMS_OUTPUT.PUT_LINE(k);
END LOOP;
END;
```

After the execution of the above code, you will get the following result:

```
1
2
3
4
5
6
7
8
9
10
```

Note: You must follow these steps while using PL/SQL WHILE Loop.

- You don't need to declare the counter variable explicitly because it is declared implicitly in the declaration section.
- The counter variable is incremented by 1 and does not need to be incremented explicitly.
- You can use EXIT WHEN statements and EXIT statements in FOR Loops but it is not done often.

PL/SQL For Loop Example 2

```
DECLARE
VAR1 NUMBER;
BEGIN
VAR1:=10;
FOR VAR2 IN 1..10
LOOP
DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
END LOOP;
END;
```

Output:

```
10
20
30
40
50
60
70
80
90
100
```

PL/SQL For Loop REVERSE Example 3

Let's see an example of PL/SQL for loop where we are using REVERSE keyword.

```
DECLARE
VAR1 NUMBER;
```

```
BEGIN
VAR1:=10;
FOR VAR2 IN REVERSE 1..10
LOOP
DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
END LOOP;
END;
```

Output:

```
100
90
80
70
60
50
40
30
20
10
```

PL/SQL Continue Statement

The continue statement is used to exit the loop from the reminder if its body either conditionally or unconditionally and forces the next iteration of the loop to take place, skipping any codes in between.

The continue statement is not a keyword in Oracle 10g. It is a new feature

encorporated in oracle 11g.

For example: If a continue statement exits a cursor FOR LOOP prematurely then it exits an inner loop and transfer control to the next iteration of an outer loop, the cursor closes (in this context, CONTINUE works like GOTO).

Syntax:

```
continue;
```

Example of PL/SQL continue statement

Let's take an example of PL/SQL continue statement.

```
DECLARE
x NUMBER := 0;
BEGIN
LOOP -- After CONTINUE statement, control resumes here
DBMS_OUTPUT.PUT_LINE ('Inside loop:  x = ' || TO_CHAR(x));
x := x + 1;
IF x < 3 THEN
CONTINUE;
END IF;
.       DBMS_OUTPUT.PUT_LINE
.       ('Inside loop, after CONTINUE:  x = ' || TO_CHAR(x));
.       EXIT WHEN x = 5;
.       END LOOP;
.
.       DBMS_OUTPUT.PUT_LINE (' After loop:  x = ' || TO_CHAR(x));
.       END;
.       /
```

After the execution of above code, you will get the following result:

```
Inside loop:  x = 0
Inside loop:  x = 1
Inside loop:  x = 2
Inside loop, after CONTINUE:  x = 3
Inside loop:  x = 3
Inside loop, after CONTINUE:  x = 4
Inside loop:  x = 4
Inside loop, after CONTINUE:  x = 5
After loop:  x = 5
```

Note: The continue statement is not supported in Oracle 10g. Oracle 11g suppo
a new feature.

PL/SQL GOTO Statement

In PL/SQL, GOTO statement makes you able to get an unconditional jump from the GOTO to a specific executable statement label in the same subprogram of the PL/SQL block.

Here the label declaration which contains the label_name encapsulated within the << >> symbol and must be followed by at least one statement to execute.

Syntax:

```
GOTO label_name;
```

Here the label declaration which contains the label_name encapsulated within th
symbol and must be followed by at least one statement to execute.

```
GOTO label_name;
..
..
<<label_name>>
Statement;
```

Example of PL/SQL GOTO statement

Let's take an example of PL/SQL GOTO statement.

```
DECLARE
a number(2) := 30;
BEGIN
<<loopstart>>
-- while loop execution
WHILE a < 50 LOOP
dbms_output.put_line ('value of a: ' || a);
a := a + 1;
IF a = 35 THEN
.       a := a + 1;
.       GOTO loopstart;
.       END IF;
.       END LOOP;
.       END;
.       /
```

After the execution of above code, you will get the following result:

```
value of a: 30
value of a: 31
value of a: 32
value of a: 33
```

```
value of a: 34
value of a: 36
value of a: 37
value of a: 38
value of a: 39
value of a: 40
value of a: 41
value of a: 42
value of a: 43
value of a: 44
value of a: 45
value of a: 46
value of a: 47
value of a: 48
value of a: 49

Statement processed.
```

Restriction on GOTO statement

Following is a list of some restrictions imposed on GOTO statement.

- Cannot transfer control into an IF statement, CASE statement, LOOP statement or sub-block.
- Cannot transfer control from one IF statement clause to another or from one CASE statement WHEN clause to another.
- Cannot transfer control from an outer block into a sub-block.
- Cannot transfer control out of a subprogram.
- Cannot transfer control into an exception handler.

Procedure

PL/SQL Procedure

The PL/SQL stored procedure or

simply a procedure is a PL/SQL block which performs one or

more specific tasks.

It is just like procedures in other programming languages.

The procedure contains a header and a body.

- Header: The header contains the name of the procedure

and the parameters or variables passed to the procedure.

- Body: The body contains a declaration section,

  execution section and exception section similar to

  a general PL/SQL block.

How to pass parameters in procedure:

When you want to create a procedure or function,

you have to define parameters .

There is three ways to pass parameters in procedure:

1. IN parameters: The IN parameter can be referenced by

   the procedure or function.

   The value of the parameter cannot be overwritten by the

   procedure or the function.

2. OUT parameters: The OUT parameter cannot be referenced

   by the procedure or function, but the value of the parameter

   can be overwritten by the procedure or function.

3. INOUT parameters: The INOUT parameter can be referenced

   by the procedure or function and the value of the parameter

   can be overwritten by the procedure or function.

A procedure may or may not return any value.

PL/SQL Create Procedure

Syntax for creating procedure:

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

Create procedure example

In this example, we are going to insert record in user table. So you need to creat
table first.

Table creation:

```
create table user(id number(10) primary key,
name varchar2(100));
```

Now write the procedure code to insert record in user table.

Procedure Code:

```
create or replace procedure "INSERTUSER"
(id IN NUMBER,
```

```
name IN VARCHAR2)
is
begin
insert into user values(id,name);
end;
```

```
Procedure created.
```

PL/SQL program to call procedure

Let's see the code to call above created procedure.

```
BEGIN
insertuser(101,'Rahul');
dbms_output.put_line('record inserted successfully');
END;
/
```

Now, see the "USER" table, you will see one record is inserted.

ID	Name
101	Rahul

PL/SQL Drop Procedure

Syntax for drop procedure

```
DROP PROCEDURE procedure_name;
```

Example of drop procedure

```
DROP PROCEDURE pro1;
```

==========

PL/SQL Function

The PL/SQL Function is very similar to PL/SQL Procedure.

The main difference between procedure and a function is,

a function must always return a value, and on the other hand

a procedure may or may not return a value. Except this,

all the other things of PL/SQL procedure are true for

PL/SQL function too.

Syntax to create a function:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
< function_body >
```

```
END [function_name];
```

Here:

- Function_name: specifies the name of the function.
- [OR REPLACE] option allows modifying an existing function.
- The optional parameter list contains name, mode and types of the paramete
- IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

The function must contain a return statement.

- RETURN clause specifies that data type you are going to return from the function.
- Function_body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

PL/SQL Function Example

Let's see a simple example to create a function.

```
create or replace function adder(n1 in number, n2 in number)
return number
is
n3 number(8);
begin
n3 :=n1+n2;
return n3;
end;
/
```

Now write another program to call the function.

```
DECLARE
n3 number(2);
BEGIN
n3 := adder(11,22);
dbms_output.put_line('Addition is: ' || n3);
END;
/
```

Output:

```
Addition is: 33
Statement processed.
0.05 seconds
```

Another PL/SQL Function Example

Let's take an example to demonstrate Declaring, Defining and Invoking a simple PL/SQL function which will compute and return the maximum of two values.

```
DECLARE
a number;
b number;
c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
z number;
BEGIN
IF x > y THEN
```

```
z:= x;
   ELSE
Z:= y;
END IF;

RETURN z;
END;
BEGIN
a:= 23;
 b:= 45;

c := findMax(a, b);
dbms_output.put_line(' Maximum of (23,45): ' || c);
END;
/
```

Output:

```
Maximum of (23,45): 45
Statement processed.
0.02 seconds
```

PL/SQL function example using table

Let's take a customer table. This example illustrates creating and calling a standalone function. This function will return the total number of CUSTOMERS in the customers table.

Create customers table and have records in it.

Customers

Id	Name	Department	Salary
1	alex	web developer	35000
2	ricky	program developer	45000
3	mohan	web designer	35000
4	dilshad	database manager	44000

Create Function:

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
total number(2) := 0;
BEGIN
SELECT count(*) into total
FROM customers;
RETURN total;
END;
/
```

After the execution of above code, you will get the following result.

```
Function created.
```

Calling PL/SQL Function:

While creating a function, you have to give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. Once the function is called, the program control is transferred to the called function.

After the successful completion of the defined task, the call function returns program control back to the main program.

To call a function you have to pass the required parameters along with function name and if function returns a value then you can store returned value. Following program calls the function totalCustomers from an anonymous block:

```
DECLARE
c number(2);
BEGIN
c := totalCustomers();
dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
```

After the execution of above code in SQL prompt, you will get the following result.

```
Total no. of Customers: 4
PL/SQL procedure successfully completed.
```

PL/SQL Recursive Function

a subprogram can call another subprogram. When a subprogram calls itself, it is called recursive call and the process is known as recursion.

Example to calculate the factorial of a number

Let's take an example to calculate the factorial of a number. This example calculates the factorial of a given number by calling itself recursively.

```
DECLARE
num number;
factorial number;

FUNCTION fact(x number)
RETURN number
IS
f number;
BEGIN
.       IF x=0 THEN
.       f := 1;
.       ELSE
.       f := x * fact(x-1);
.       END IF;
.       RETURN f;
.       END;
.
.       BEGIN
.       num:= 6;
.       factorial := fact(num);
.       dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
.       END;
.       /
```

After the execution of above code at SQL prompt, it produces the following result.

Factorial 6 is 720
PL/SQL procedure successfully completed.

PL/SQL Drop Function

Syntax for removing your created function:

If you want to remove your created function from the database, you should use the following syntax.

DROP FUNCTION function_name;

PL/SQL Cursor

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

- Implicit Cursors
- Explicit Cursors

1) PL/SQL Implicit Cursors

The implicit cursors are automatically generated by

Oracle while an SQL statement is executed,

if you don't use an explicit cursor for the statement.

These are created by default to process the statements

when DML statements like INSERT, UPDATE, DELETE etc.

are executed.

Oracle provides some attributes known as Implicit cursor's

attributes to check the status of DML operations.

Some of them are: %FOUND, %NOTFOUND, %ROWCOUNT

and %ISOPEN.

For example: When you execute the SQL statements like

INSERT, UPDATE, DELETE then the cursor attributes tell

whether any rows are affected and how many have been affected. If you run a SELECT INTO statement in PL/SQL block, the implicit cursor attribute can be used to find out whether any row has been returned by the SELECT statement. It will return an error if there no data is selected.

The following table specifies the status of the cursor with each of its attribute.

Attribute	Description
%FOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect at least one row or more rows or a SELECT INTO statement returned one or more rows. Otherwise it returns FALSE.

%NOTFOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect no row, or a SELECT INTO statement return no rows. Otherwise it returns FALSE. It is a just opposite of %FOUND.
%ISOPEN	It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements.
%ROWCOUNT	It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDATE or returned by a SELECT INTO statement.

PL/SQL Implicit Cursor Example

Create customers table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000

6	Sunita	20	Delhi	30000

Let's execute the following program to update the table and increase salary of each customer by 5000. Here, SQL%ROWCOUNT attribute is used to determine the number of rows affected:

Create procedure:

```
DECLARE
total_rows number(2);
BEGIN
UPDATE  customers
SET salary = salary + 5000;
IF sql%notfound THEN
dbms_output.put_line('no customers updated');
ELSIF sql%found THEN
total_rows := sql%rowcount;
.       dbms_output.put_line( total_rows || ' customers updated ');
.       END IF;
.       END;
.       /
```

Output:

```
6 customers updated
PL/SQL procedure successfully completed.
```

Now, if you check the records in customer table, you will find that the rows are updated.

```
select * from customers;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	25000
2	Suresh	22	Kanpur	27000
3	Mahesh	24	Ghaziabad	29000
4	Chandan	25	Noida	31000
5	Alex	21	Paris	33000
6	Sunita	20	Delhi	35000

2) PL/SQL Explicit Cursors

The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Following is the syntax to create an explicit cursor:

Syntax of explicit cursor

Following is the syntax to create an explicit cursor:

CURSOR cursor_name IS select_statement;;

Steps:

You must follow these steps while working with an explicit cursor.

1. Declare the cursor to initialize in the memory.
2. Open the cursor to allocate memory.
3. Fetch the cursor to retrieve data.
4. Close the cursor to release allocated memory.

1) Declare the cursor:

It defines the cursor with a name and the associated SELECT statement.

Syntax for explicit cursor decleration

```
CURSOR name IS
SELECT statement;
```

2) Open the cursor:

It is used to allocate memory for the cursor and make it easy to fetch the rows returned by the SQL statements into it.

Syntax for cursor open:

```
OPEN cursor_name;
```

3) Fetch the cursor:

It is used to access one row at a time. You can fetch rows from the above-opened cursor as follows:

Syntax for cursor fetch:

```
FETCH cursor_name INTO variable_list;
```

4) Close the cursor:

It is used to release the allocated memory. The following syntax is used to close the above-opened cursors.

Syntax for cursor close:

```
Close cursor_name;
```

PL/SQL Explicit Cursor Example

Explicit cursors are defined by programmers to gain more control over the context area. It is defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Let's take an example to demonstrate the use of explicit cursor. In this example, we are using the already created CUSTOMERS table.

Create customers table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000

5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create procedure:

Execute the following program to retrieve the customer name and address.

```
DECLARE
c_id customers.id%type;
c_name customers.name%type;
c_addr customers.address%type;
CURSOR c_customers is
SELECT id, name, address FROM customers;
BEGIN
OPEN c_customers;
LOOP
.       FETCH c_customers into c_id, c_name, c_addr;
.       EXIT WHEN c_customers%notfound;
.       dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
.       END LOOP;
.       CLOSE c_customers;
.       END;
.       /
```

Output:

```
1  Ramesh  Allahabad
2  Suresh  Kanpur
3  Mahesh  Ghaziabad
```

```
4  Chandan  Noida
5  Alex  Paris
6  Sunita  Delhi
PL/SQL procedure successfully completed.
```

PL/SQL Exception Handling

What is Exception

An error occurs during the program execution is called

Exception in PL/SQL.

PL/SQL facilitates programmers to catch such conditions

using exception block in the program and an appropriate

action is taken against the error condition.

There are two type of exceptions:

- System-defined Exceptions
- User-defined Exceptions

PL/SQL Exception Handling

Syntax for exception handling:

Following is a general syntax for exception handling:

```
DECLARE
<declarations section>
BEGIN
<executable command(s)>
EXCEPTION
<exception handling goes here >
WHEN exception1 THEN
exception1-handling-statements
WHEN exception2  THEN
.       exception2-handling-statements
.       WHEN exception3 THEN
.       exception3-handling-statements
.       ........
.       WHEN others THEN
.       exception3-handling-statements
.       END;
```

Example of exception handling

Let's take a simple example to demonstrate the concept of

exception handling.

Here we are using the already created CUSTOMERS table.

SELECT* FROM COUSTOMERS;

ID	NAME	AGE	ADDRESS	SALARY

1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

```
DECLARE
c_id customers.id%type := 8;
c_name  customers.name%type;
c_addr customers.address%type;
BEGIN
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);
.       EXCEPTION
.       WHEN no_data_found THEN
.       dbms_output.put_line('No such customer!');
.       WHEN others THEN
.       dbms_output.put_line('Error!');
.       END;
.       /
```

After the execution of above code at SQL Prompt,

it produces the following result:

```
No such customer!
PL/SQL procedure successfully completed.
```

The above program should show the name and address of a customer as result whose ID is given. But there is no customer with ID value 8 in our database, so the program raises the run-time exception NO_DATA_FOUND, which is captured in EXCEPTION block.

Note: You get the result "No such customer" because the customer_id used in the above example is 8 and there is no cutomer having id value 8 in that table.

If you use the id defined in the above table (i.e. 1 to 6), you will get a certain res
demo example: here, we are using the id 5.

```
DECLARE
c_id customers.id%type := 5;
c_name  customers.name%type;
c_addr customers.address%type;
BEGIN
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);
.       EXCEPTION
.       WHEN no_data_found THEN
```

```
.       dbms_output.put_line('No such customer!');
.       WHEN others THEN
.       dbms_output.put_line('Error!');
.       END;
.       /
```

After the execution of above code at SQL prompt, you will get the following res

```
Name: alex
Address: paris
PL/SQL procedure successfully completed.
```

Raising Exceptions

In the case of any internal database error, exceptions are raised by the database server automatically. But it can also be raised explicitly by programmer by using command RAISE.

Syntax for raising an exception:

```
DECLARE
exception_name EXCEPTION;
BEGIN
IF condition THEN
RAISE exception_name;
END IF;
EXCEPTION
WHEN exception_name THEN
statement;
.       END;
```

PL/SQL User-defined Exceptions

PL/SQL facilitates their users to define their own exceptions according to the need of the program. A user-defined exception can be raised explicitly, using either a RAISE statement or the procedure DBMS_STANDARD.RAISE_APPLICATION_ERROR.

Syntax for user define exceptions

```
DECLARE
my-exception EXCEPTION;
```

PL/SQL Pre-defined Exceptions

There are many pre-defined exception in PL/SQL which are executed when any database rule is violated by the programs.

For example: NO_DATA_FOUND is a pre-defined exception which is raised when a SELECT INTO statement returns no rows.

Following is a list of some important pre-defined exceptions:

Exception	Oracle Error	SQL Code	Description
ACCESS_INTO_NULL	06530	-6530	It is raised when a NULL object is automatically assigned a value.
CASE_NOT_FOUND	06592	-6592	It is raised when none of the

			choices in the "WHEN" clauses of a CASE statement is selected, and there is no else clause.
COLLECTION_IS_NULL	06531	-6531	It is raised when a program attempts to apply collection methods other than exists to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray.
DUP_VAL_ON_INDEX	00001	-1	It is raised when duplicate values are attempted to be stored in a column with unique index.

INVALID_CURSOR	01001	-1001	It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.
INVALID_NUMBER	01722	-1722	It is raised when the conversion of a character string into a number fails because the string does not represent a valid number.
LOGIN_DENIED	01017	-1017	It is raised when s program attempts to log on to the database with an invalid username or password.
NO_DATA_FOUND	01403	+100	It is raised when a select into statement returns no rows.

NOT_LOGGED_ON	01012	-1012	It is raised when a database call is issued without being connected to the database.
PROGRAM_ERROR	06501	-6501	It is raised when PL/SQL has an internal problem.
ROWTYPE_MISMATCH	06504	-6504	It is raised when a cursor fetches value in a variable having incompatible data type.
SELF_IS_NULL	30625	-30625	It is raised when a member method is invoked, but the instance of the object type was not initialized.
STORAGE_ERROR	06500	-6500	It is raised when PL/SQL ran out of memory or memory was corrupted.

TOO_MANY_ROWS	01422	-1422	It is raised when a SELECT INTO statement returns more than one row.
VALUE_ERROR	06502	-6502	It is raised when an arithmetic, conversion, truncation, or size-constraint error occurs.
ZERO_DIVIDE	01476	1476	It is raised when an attempt is made to divide a number by zero.

PL/SQL Trigger

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any

of the following events.

- A database manipulation (DML) statement
- (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement
- (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF,
- STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or

database with which the event is associated.

Advantages of Triggers

These are the following advantages of Triggers:

- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating a trigger:

Syntax for creating trigger:

CREATE [OR REPLACE] TRIGGER trigger_name

{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE

. Declaration-statements
. BEGIN
. Executable-statements
. EXCEPTION
. Exception-handling-statements
. END;

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new an values for various DML statements, like INSERT, UPDATE, and DELETE
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger wou

executed for each row being affected. Otherwise the trigger will execute ju
when the SQL statement is executed, which is called a table level trigger.

- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

PL/SQL Trigger Example

Let's take a simple example to demonstrate the trigger. In this example, we are using the following CUSTOMERS table:

Create table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create trigger:

Let's take a program to create a row level trigger for the CUSTOMERS table tha

fire for INSERT or

UPDATE or DELETE operations performed on the CUSTOMERS table.

This trigger will display the salary difference between the old values and new v

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
sal_diff number;
BEGIN
sal_diff := :NEW.salary  - :OLD.salary;
dbms_output.put_line('Old salary: ' || :OLD.salary);
.       dbms_output.put_line('New salary: ' || :NEW.salary);
.       dbms_output.put_line('Salary difference: ' || sal_diff);
.       END;
.       /
```

After the execution of the above code at SQL Prompt, it produces the following

```
Trigger created.
```

Check the salary difference by procedure:

Use the following code to get the old salary, new salary and salary difference af
trigger created.

```
DECLARE
total_rows number(2);
BEGIN
UPDATE  customers
```

```
SET salary = salary + 5000;
IF sql%notfound THEN
dbms_output.put_line('no customers updated');
ELSIF sql%found THEN
total_rows := sql%rowcount;
.       dbms_output.put_line( total_rows || ' customers updated ');
.       END IF;
.       END;
.       /
```

Output:

```
Old salary: 20000
New salary: 25000
Salary difference: 5000
Old salary: 22000
New salary: 27000
Salary difference: 5000
Old salary: 24000
New salary: 29000
Salary difference: 5000
Old salary: 26000
New salary: 31000
Salary difference: 5000
Old salary: 28000
New salary: 33000
Salary difference: 5000
Old salary: 30000
New salary: 35000
Salary difference: 5000
6 customers updated
```

Note: As many times you executed this code, the old and new both salary is incremented by 5000 and hence the salary difference is always 5000.

After the execution of above code again, you will get the following result.

```
Old salary: 25000
New salary: 30000
Salary difference: 5000
Old salary: 27000
New salary: 32000
Salary difference: 5000
Old salary: 29000
New salary: 34000
Salary difference: 5000
Old salary: 31000
New salary: 36000
Salary difference: 5000
Old salary: 33000
New salary: 38000
Salary difference: 5000
Old salary: 35000
New salary: 40000
Salary difference: 5000
6 customers updated
==========================================
```

Important Points

Following are the two very important point and should be noted carefully.

- OLD and NEW references are used for record level triggers these are

not avialable for table level triggers.

- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.

Questions and answers for preparation of interview :

1) What is PL/SQL?

PL/SQL stands for procedural language extension to SQL. It supports procedura
of programming language and SQL both. It was developed by Oracle Corporati
of 90's to enhance the capabilities of SQL.

2) What is the purpose of using PL/SQL?

PL/SQL is an extension of SQL. While SQL is non-procedural, PL/SQL is a pro
language designed by Oracle. It is invented to overcome the limitations of SQL.

3) What are the most important characteristics of PL/SQL?

A list of some notable characteristics:

- PL/SQL is a block-structured language.
- It is portable to all environments that support Oracle.
- PL/SQL is integrated with the Oracle data dictionary.

- Stored procedures help better sharing of application.

4) What is PL/SQL table? Why it is used?

Objects of type tables are called PL/SQL tables that are modeled as database tab
can also say that PL/SQL tables are a way to providing arrays. Arrays are like te
tables in memory that are processed very quickly. PL/SQL tables are used to mo
data. They simplifies moving collections of data.

5) What are the datatypes available in PL/SQL?

There are two types of datatypes in PL/SQL:

1. Scalar datatypes Example are NUMBER, VARCHAR2, DATE, CHAR, L
   BOOLEAN etc.
2. Composite datatypes Example are RECORD, TABLE etc.

6) What is the basic structure of PL/SQL?

PL/SQL uses BLOCK structure as its basic structure. Each PL/SQL program co
SQL and PL/SQL statement which form a PL/SQL block.

PL/SQL block contains 3 sections.

1. The Declaration Section (optional)
2. The Execution Section (mandatory)
3. The Exception handling Section (optional)

7) What is the difference between FUNCTION, PROCEDURE AND PACKAG
PL/SQL?

Function: The main purpose of a PL/SQL function is generally to compute and return a single value. A function has a return type in its specification and must return a value specified in that type.

Procedure: A procedure does not have a return type and should not return any value but it can have a return statement that simply stops its execution and returns to the caller. A procedure is used to return multiple values otherwise it is generally similar to a function.

Package: A package is schema object which groups logically related PL/SQL types , items and subprograms. You can also say that it is a group of functions, procedure, variables and record type statement. It provides modularity, due to this facility it aids application development. It is used to hide information from unauthorized users.

8) What is exception? What are the types of exceptions?

Exception is an error handling part of PL/SQL. There are two type of exceptions pre_defined exception and user_defined exception.

9) How to write a single statement that concatenates the words ?Hello? and ?W assign it in a variable named Greeting?

Greeting := 'Hello' || 'World';

10) Does PL/SQL support CREATE command?

No. PL/SQL doesn't support the data definition commands like CREATE.

11) Write a unique difference between a function and a stored procedure.

A function returns a value while a stored procedure doesn?t return a value.

12) How exception is different from error?

Whenever an Error occurs Exception arises. Error is a bug whereas exception is or error condition.

13) What is the main reason behind using an index?

Faster access of data blocks in the table.

14) What are PL/SQL exceptions? Tell me any three.

1. Too_many_rows
2. No_Data_Found
3. Value_error
4. Zero_error etc.

15) How do you declare a user-defined exception?

You can declare the User defined exceptions under the DECLARE section, with keyword EXCEPTION.

Syntax:

<exception_name> EXCEPTION;

16) What are some predefined exceptions in PL/SQL?

A list of predefined exceptions in PL/SQL:

- DUP_VAL_ON_INDEX
- ZERO_DIVIDE
- NO_DATA_FOUND
- TOO_MANY_ROWS
- CURSOR_ALREADY_OPEN
- INVALID_NUMBER
- INVALID_CURSOR
- PROGRAM_ERROR
- TIMEOUT_ON_RESOURCE
- STORAGE_ERROR
- LOGON_DENIED
- VALUE_ERROR
- etc.

17) What is a trigger in PL/SQL?

A trigger is a PL/SQL program which is stored in the database. It is executed im

before or after the execution of INSERT, UPDATE, and DELETE commands.

18) What is the maximum number of triggers, you can apply on a single table?

12 triggers.

19) How many types of triggers exist in PL/SQL?

There are 12 types of triggers in PL/SQL that contains the combination of BEF
AFTER, ROW, TABLE, INSERT, UPDATE, DELETE and ALL keywords.

- BEFORE ALL ROW INSERT
- AFTER ALL ROW INSERT
- BEFORE INSERT
- AFTER INSERT etc.

20) What is the difference between execution of triggers and stored procedures?

A trigger is automatically executed without any action required by the user, whi
stored procedure is explicitly invoked by the user.

21) What happens when a trigger is associated to a view?

When a trigger is associated to a view, the base table triggers are normally enab

22) What is the usage of WHEN clause in trigger?

A WHEN clause specifies the condition that must be true for the trigger to be tri

23) How to disable a trigger name update_salary?

ALTER TRIGGER update_salary DISABLE;

24) Which command is used to delete a trigger?

DROP TRIGGER command.

25) what are the two virtual tables available at the time of database trigger execu

Table columns are referred as THEN.column_name and NOW.column_name.

For INSERT related triggers, NOW.column_name values are available only.

For DELETE related triggers, THEN.column_name values are available only.

For UPDATE related triggers, both Table columns are available.

26) What is stored Procedure?

A stored procedure is a sequence of statement or a named PL/SQL block which
one or more specific functions. It is similar to a procedure in other programming
languages. It is stored in the database and can be repeatedly executed. It is store

schema object. It can be nested, invoked and parameterized.

27) What are the different schemas objects that can be created using PL/SQL?

- Stored procedures and functions
- Packages
- Triggers
- Cursors

28) What do you know by PL/SQL Cursors?

Oracle uses workspaces to execute the SQL commands. When Oracle processes
command, it opens an area in the memory called Private SQL Area. This area is
by the cursor. It allows programmers to name this area and access it?s informati

29) What is the difference between the implicit and explicit cursors?

Implicit cursor is implicitly declared by Oracle. This is a cursor to all the DDL a
commands that return only one row.

Explicit cursor is created for queries returning multiple rows.

30) What will you get by the cursor attribute SQL%ROWCOUNT?

The cursor attribute SQL%ROWCOUNT will return the number of rows that ar
processed by a SQL statement.

31) What will you get by the cursor attribute SQL%FOUND?

It returns the Boolean value TRUE if at least one row was processed.

32) What will you get by the cursor attribute SQL%NOTFOUND?

It returns the Boolean value TRUE if no rows were processed.

33) What do you understand by PL/SQL packages?

A PL/SQL package can be specified as a file that groups functions, cursors, stor
procedures, and variables in one place.

34) What are the two different parts of the PL/SQL packages?

PL/SQL packages have the following two parts:

Specification part: It specifies the part where the interface to the application is d

Body part: This part specifies where the implementation of the specification is d

35) Which command is used to delete a package?

The DROP PACKAGE command is used to delete a package.

36) How to execute a stored procedure?

There are two way to execute a stored procedure.

From the SQL prompt, write EXECUTE or EXEC followed by procedure_name

EXECUTE or [EXEC] procedure_name;

Simply use the procedure name

procedure_name;

37) What are the advantages of stored procedure?

Modularity, extensibility, reusability, Maintainability and one time compilation.

38) What are the cursor attributes used in PL/SQL?

%ISOPEN: it checks whether the cursor is open or not.

%ROWCOUNT: returns the number of rows affected by DML operations: INSERT,DELETE,UPDATE,SELECT.

%FOUND: it checks whether cursor has fetched any row. If yes - TRUE.

%NOTFOUND: it checks whether cursor has fetched any row. If no - TRUE.

39) What is the difference between syntax error and runtime error?

A syntax error can be easily detected by a PL/SQL compiler. For example: inco
spelling etc. while, a runtime error is handled with the help of exception-handli

in a PL/SQL block. For example: SELECT INTO statement, which does not ret
rows.

40) Explain the Commit statement.

Following conditions are true for the Commit statement:

- Other users can see the data changes made by the transaction.
- The locks acquired by the transaction are released.
- The work done by the transaction becomes permanent.

41) Explain the Rollback statement?

The Rollback statement is issued when the transaction ends. Following conditio
for a Rollback statement:

- The work done in a transition is undone as if it was never issued.
- All locks acquired by transaction are released.

42) Explain the SAVEPOINT statement.

With SAVEPOINT, only part of transaction can be undone.

43) What is mutating table error?

Mutating table error is occurred when a trigger tries to update a row that it is cu

using. It is fixed by using views or temporary tables.

44) What is consistency?

Consistency simply means that each user sees the consistent view of the data.

Consider an example: there are two users A and B. A transfers money to B's acc
Here the changes are updated in A's account (debit) but until it will be updated t
account (credit), till then other users can't see the debit of A's account. After the
and credit of B, one can see the updates. That?s consistency.

45) What is cursor and why it is required?

A cursor is a temporary work area created in a system memory when an SQL st
executed.

A cursor contains information on a select statement and the row of data accesse
This temporary work area stores the data retrieved from the database and manip
data. A cursor can hold more than one row, but can process only one row at a ti
are required to process rows individually for queries.

46) How many types of cursors are available in PL/SQL?

There are two types of cursors in PL/SQL.

1. Implicit cursor, and
2. explicit cursor