UNIT 4: PL/SQL

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

Advantages of PL/SQL

1. It is a standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and also dynamic SQL. Static SQL is said to support DML operations and also the transaction control from PL/SQL block. In dynamic SQL, SQL allows embedding the DDL statements in the PL/SQL blocks.
2. Also, It then allows sending an entire block of statements to the database at one time. It reduces network traffic and also provides high performance for the applications.
3. It gives high productivity to programmers as it can query, transform and also update the data in the database.
4. This is said to save time on the design and also the debugging by strong features, like the exception handling, encapsulation, data hiding and also object-oriented data types.
5. Applications that are written in PL/SQL languages are portable.
6. this provides high security level.
7. It also provides access to the predefined SQL packages.
8. It also supports for Object-oriented programming.
9. It provides support for developing web applications and server pages.

PL/SQL Execution Environment:

The PL/SQL engine resides in the Oracle engine.The Oracle engine can process not only single SQL statement but also block of many statements.The call to Oracle engine needs to be made only once to execute any number of SQL statements if these SQL statements are bundled inside a PL/SQL block.

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.

Orcale 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 soecifies 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 or 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 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 a statements.
%ROWCOUNT	It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDAT statement.

PL/SQL Implicit Cursor Example

Create customers table and have records:

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

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:

```
1.  DECLARE
2.    total_rows number(2);
3.  BEGIN
4.    UPDATE  customers
5.    SET salary = salary + 5000;
6.    IF sql%notfound THEN
7.       dbms_output.put_line('no customers updated');
8.    ELSIF sql%found THEN
9.       total_rows := sql%rowcount;
10.      dbms_output.put_line( total_rows || ' customers updated ');
11.   END IF;
12. END;
13. /
```

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.

1. **select** * **from** customers;

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

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:

1. **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

```
1. CURSOR name IS
2.  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:

```
1. 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:

```
1. 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:

```
1. 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
1	Ramesh	23	Allahabad
2	Suresh	22	Kanpur
3	Mahesh	24	Ghaziabad
4	Chandan	25	Noida
5	Alex	21	Paris
6	Sunita	20	Delhi

Create procedure:

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

```
1. DECLARE
2.    c_id customers.id%type;
3.    c_name customers.name%type;
4.    c_addr customers.address%type;
5.    CURSOR c_customers is
6.       SELECT id, name, address FROM customers;
7. BEGIN
```

```
8.    OPEN c_customers;
9.    LOOP
10.     FETCH c_customers into c_id, c_name, c_addr;
11.     EXIT WHEN c_customers%notfound;
12.     dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
13.   END LOOP;
14.   CLOSE c_customers;
15. END;
16. /
```

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 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 an 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 and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once 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
1	Ramesh	23	Allahabad
2	Suresh	22	Kanpur
3	Mahesh	24	Ghaziabad
4	Chandan	25	Noida
5	Alex	21	Paris

6	Sunita	20	Delhi

Create trigger:

Let's take a program to create a row level trigger for the CUSTOMERS table that would 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 values:

```
1.  CREATE OR REPLACE TRIGGER display_salary_changes
2.  BEFORE DELETE OR INSERT OR UPDATE ON customers
3.  FOR EACH ROW
4.  WHEN (NEW.ID > 0)
5.  DECLARE
6.    sal_diff number;
7.  BEGIN
8.    sal_diff := :NEW.salary  - :OLD.salary;
9.    dbms_output.put_line('Old salary: ' || :OLD.salary);
10.   dbms_output.put_line('New salary: ' || :NEW.salary);
11.   dbms_output.put_line('Salary difference: ' || sal_diff);
12. END;
13. /
```

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

```
Trigger created.
```

PL/SQL Transactions

Write a PL/SQL code that will accept an account number from user. Check if user balance is less than minimum balance than only deduct Rs 100 from balance. The process is fired on the acct_mstr.

Example – Consider the following scenario,

```
acct_master (acct_no number(5) primary key,
                        acct_name varchar2(10),
                           balance number(10));
```

First, you need to create table acct_master,

```
# CREATING table acct_master
create table acct_master(acct_no number(5) primary key,
                                 acct_name varchar2(10),
                                    balance number(10));
```

Insert these data into the table,

```
# INSERTING data in acct_mstr
insert into acct_master values(1, 'aaa', 1000)
insert into acct_master values(2, 'bbb', 100)
insert into acct_master values(3, 'ccc', 1100)
insert into acct_master values(4, 'ddd', 700)
insert into acct_master values(5, 'eee', 1700)
```

Approach used –

- **Step-1:** Declare variables and set minimum balance in one of variable.
- **Step-2:** Take account number input from user.
- **Step-3:** Select the balance of that user into one variable.
- **Step-4:** Check if balance is less than minimum balance or not.
- **Step-5:** If balance is less, then update the balance in table with balance = balance - 100, and display the balance after deducting 100 from that variable in which balance was stored. Then, display the output.
- **Step-6:** Else simply print the value.

Note that all text in green colour are comments.

Below is the required implementation:

```
-- DECLARING VARIABLES

DECLARE

xacct_no number(5);



-- here, minimum balance is set to 1000;
```

```
xmin_bal number(5):=1000;

xbalance number(5);



BEGIN



-- taking input from user

xacct_no:=&xacct_no;



-- selecting balance of that user INTO "xbalance";

select balance into xbalance

from acct_master

where acct_no=xacct_no;



-- if condition true, updating balance

-- with balance = balance - 100

IF(xbalance < xmin_bal) THEN --condition check

update acct_master

set balance=balance-100

where acct_no=xacct_no;
```

```
-- remaining
amount

xbalance:=xbalance-100;

dbms_output.put_line('Rs 100 is deducted

               and current balance is '||xbalance);



-- if condition is false

ELSE

dbms_output.put_line('Current balance is '||xbalance);



--ENDING IF

END IF;



-- ENDING OF BEGIN

END;



/     -- FOR DISPLAYING OUTPUT IN SCREEN
```

Output:

```
Enter value for xacct_no: 2

old 6: xacct_no:=&xacct_no;

new 6: xacct_no:=2;

Rs 100 is deducted and current balance is 0
```

```
PL/SQL procedure successfully completed.


SQL> /
Enter value for xacct_no: 3
old   6: xacct_no:=&xacct_no;
new   6: xacct_no:=3;
Current balance is 1100

PL/SQL procedure successfully completed.
```

Concurrency Control

Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

The concurrency control protocols ensure the *atomicity, consistency, isolation, durability* and *serializability* of the concurrent execution of the database transactions. Therefore, these protocols are categorized as:

- Lock Based Concurrency Control Protocol
- Time Stamp Concurrency Control Protocol
- Validation Based Concurrency Control Protocol

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

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

```
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 result. For a demo example: here, we are using the id 5.

```
1.  DECLARE
2.    c_id customers.id%type := 5;
3.    c_name  customers.name%type;
4.    c_addr customers.address%type;
5.  BEGIN
6.    SELECT  name, address INTO  c_name, c_addr
7.    FROM customers
8.    WHERE id = c_id;
9.  DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
10.  DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
11. EXCEPTION
12.   WHEN no_data_found THEN
13.     dbms_output.put_line('No such customer!');
14.   WHEN others THEN
15.     dbms_output.put_line('Error!');
16. END;
17. /
```

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

```
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;
```