Introduction to PL/SQL

Introduction to PL/SQL

- What is PL/SQL
- Why PL/SQL
- Advantage of PL/SQL
- Kinds of PL/SQL BLOCKS
  - Anonymous or Named Blocks
- Named Blocks (Stored procedures, Funtions, Triggers)

What is PL/SQL

- PL/SQL is a sophistical programming language used to access an database from a various environments.

- PL/SQL stands for Procedural Language/SQL.

- It extends SQL by adding constructs found in other procedural languages such as:-
  - **loops**,
  - **conditional statements,**
  - **declared variables,**
  - **accessing individual records one at a time**,
  - and many others.

Why use PL/SQL

- Compared to SQL, PL/SQL has the procedural constructs that are useful **to express a desired process from start to end.**

- One block of PL/SQL code can **bundled several SQL statements together** as a single unit.
- Making less network traffic and improving application performance.

- PL/SQL can be **integrated with other languages**, such as Java, to take advantage of the strongest features of both languages.

Advantages of PL/SQL

These are the advantages of PL/SQL.

- ***Block Structures*:**
- PL SQL consists of blocks of code, which can be nested within each other.
- Each block forms a unit of a task or a logical module.
- PL/SQL Blocks can be stored in the database and reused.

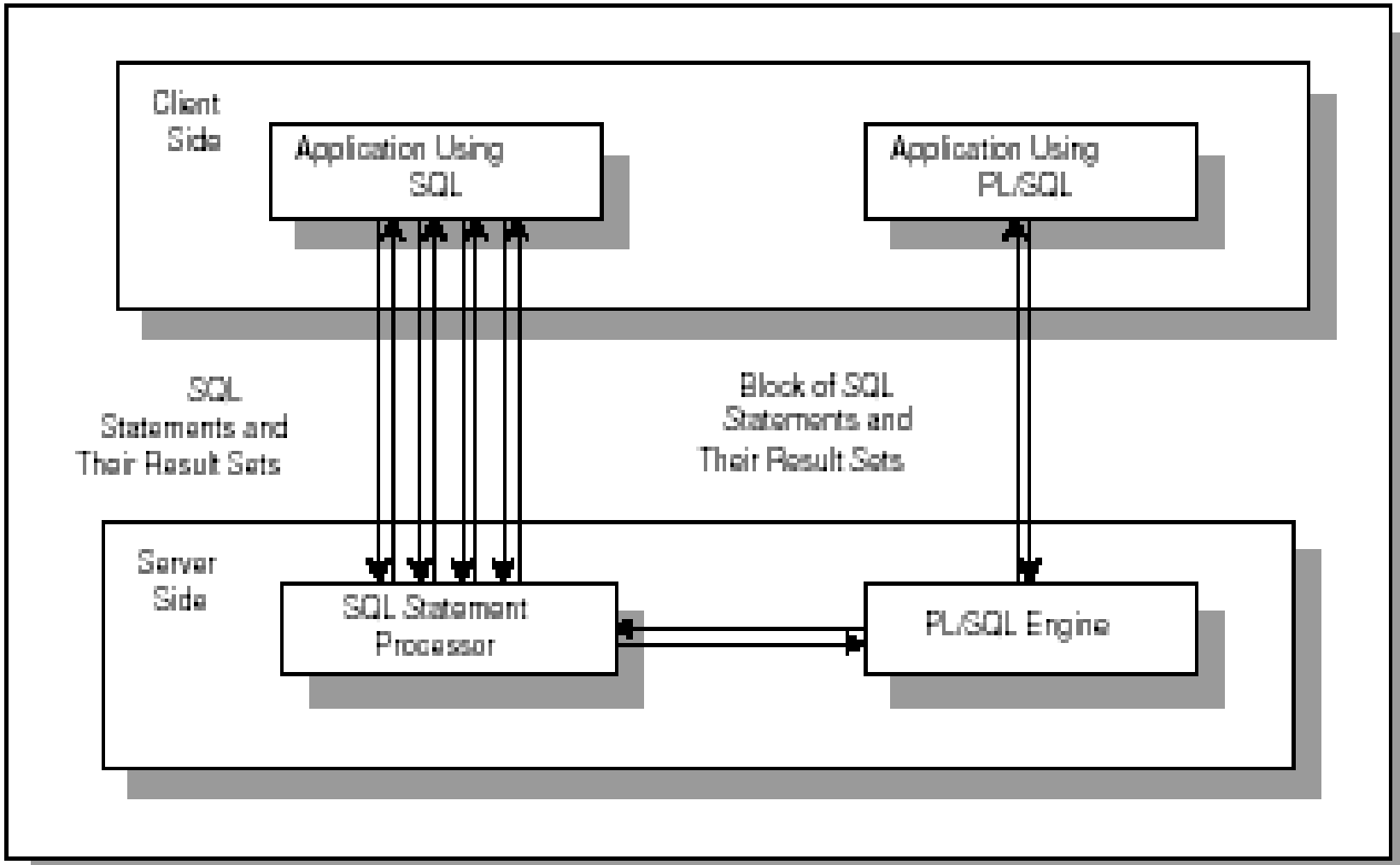
- ***Procedural Language Capability*:**
- PL SQL consists of procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).

- ***Better Performance*:**
- PL SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.

- ***Error Handling*:**
- PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program.
- Once an exception is caught, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

- DIFFERENCE BETWEEN PL/SQL AND SQL

- When a SQL statement is issued on the client computer, the request is made to the database on the server, and the result set is sent back to the client.
- As a result, a single SQL statement causes two trips on the network.
- If multiple SELECT statements are issued, the network traffic increase significantly very fast.
- For example, four SELECT statements cause eight network trips.
- If these statements are part of the PL/SQL block, they are sent to the server as a single unit.
- The SQL statements in this PL/SQL program are executed at the server and the result set is sent back as a single unit.
- There is still only one network trip made as is in case of a single SELECT statement.



PL/SQL in client server Architecture

Kinds of PL/SQL BLOCKS

- The basic unit in any PL/SQL PROGRAM is a **BLOCK.**
- All PL/SQL programs are composed of a single block or blocks that occur either sequentially or nested within another block.

- There are two kinds of blocks:

- **Anonymous blocks** are generally constructed dynamically and **executed only once** by the user.
- It is sort of a complex SQL statement.

- **Named blocks** are blocks that have a name associated with them.
- are stored in the database, and can be **executed again and again.**
- can take in parameters, and can modify and existing database.

PL/SQL BLOCK STRUCTURE

- PL/SQL blocks contain three sections

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

Optional.

Mandatory

- The executable section is the only mandatory section of the block.

- Both the declaration and exception-handling sections are optional.

Structure of Anonymous Block

```
DECLARE
/* Declare section (optional).  */

BEGIN
/* Executable section (required).  */

EXCEPTION
/* Exception handling section
  (optional).  */

END;  -- end the block (do not forget
  the " ; " in the end.)
/
```

Example of Anonymous Block

- **Example 1.**

- An anonymous block without declaration and exception sections
- *BEGIN*
- *dbms_output.put_line('Hello Wold');*
- *END;*

- Out put:-
- *Hello Wold*
- *Statement processed.*

- An anonymous block with declaration and without exception sections

- **Example 2**

```
DECLARE
hiredate date;
BEGIN
hiredate := SYSDATE;
dbms_output.put_line('System date is:-'||hiredate);
END;
```

OUT PUT:-
System date is:-02-AUG-12

- **Example 3**
- An anonymous block with declaration and exception sections
- Declare
- x number;
- y number;
- z number;
- Begin
- x := 100;
- y := 0;
- z := x/y;
- dbms_output.put_line('The answer is ' || z);
- Exception
- When ZERO_DIVIDE then
- dbms_output.put_line('Cannot divide by zero!!!');
- End;

Out put:-

Cannot divide by zero!!!

- An anonymous block with declaration and exception sections
- Declare
- x number;
- y number;
- z number;
- Begin
- x := 100;
- y := 10;
- z := x/y;
- dbms_output.put_line('The answer is ' || z);
- Exception
- When ZERO_DIVIDE then
- dbms_output.put_line('Cannot divide by zero!!!');
- End;

Out put:-

The answer is 10

DECLARATION SECTION

- The *declaration section* is the first section of the PL/SQL block.
- It contains definitions of PL/SQL identifiers such as variables, constants, cursors and so on.

- Example

```
DECLARE
    v_first_name VARCHAR2(35) ;
    v_last_name VARCHAR2(35) ;
    v_counter NUMBER := 0 ;
```

EXECUTABLE SECTION

- The executable section is the next section of the PL/SQL block.
- This section contains executable statements that allow you to manipulate the variables that have been declared in the declaration section.

```
BEGIN
    SELECT first_name, last_name
        INTO v_first_name, v_last_name
        FROM student
        WHERE student_id = 123 ;
    DBMS_OUTPUT.PUT_LINE
    ('Student name :' || v_first_name ||'  '||
v_last_name);
END;
```

EXCEPTION-HANDLING SECTION

- The *exception-handling section* is the last section of the PL/SQL block.
- This section contains statements that are executed when a runtime error occurs within a block.
- Runtime errors occur while the program is running and cannot be detected by the PL/SQL compiler.

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE
  (‘ There is no student with student id 123 ’);
END;
```

- SQL*Plus is an interactive tool that allows you to type SQL or PL/SQL statements at the command prompt.

- These statements are then sent to the database.
- Once they are processed, the results are sent back from the database and displayed on the screen.

- There are some differences between entering SQL and PL/SQL statements.

What are Cursors?

- A cursor is a temporary work area created in the system memory when a SQL statement is executed.
- A cursor contains information on a select statement and the rows of data accessed by it.
- This temporary work area is used to store the data retrieved from the database, and manipulate this data.
- A cursor can hold more than one row, **but can process only one row at a time.**
- The set of rows the cursor holds is called the *active* set.

- There are two types of cursors in PL/SQL:
- **Implicit cursors:**
- These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed.
- They are also created when a SELECT statement that returns just one row is executed.

- **Explicit cursors:**
- 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.
- Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed.

Implicit Cursors:

- When you execute DML statements like DELETE, INSERT, UPDATE and SELECT statements, implicit statements are created to process these statements.
- Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations.
- The cursor attributes available are %FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN.
- For example, When you execute INSERT, UPDATE, or DELETE statements the cursor attributes tell us whether any rows are affected and how many have been affected.
- When a SELECT... INTO statement is executed in a PL/SQL Block, implicit cursor attributes can be used to find out whether any row has been returned by the SELECT statement.
- PL/SQL returns an error when no data is selected.

The status of the cursor for each of these attributes are defined in the below table.

Attributes	Return Value	Example
%FOUND	The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row and if SELECT ….INTO statement return at least one row.	SQL%FOUND
	The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE do not affect row and if SELECT….INTO statement do not return a row.	
%NOTFOUND	The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE at least one row and if SELECT ….INTO statement return at least one row.	SQL%NOTFOUND
	The return value is TRUE, if a DML statement like INSERT, DELETE and UPDATE do not affect even one row and if SELECT ….INTO statement does not return a row.	
%ROWCOUNT	Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE, SELECT	SQL%ROWCOUNT

- For Example: Consider the PL/SQL Block that uses implicit cursor attributes as shown below:
- *DECLARE*
- *var_rows number(5);*
- *BEGIN*
- *UPDATE employee SET salary = salary + 1000;*
- *IF SQL%NOTFOUND THEN*
- *dbms_output.put_line('None of the salaries where updated');*
- *ELSIF SQL%FOUND THEN*
- *var_rows := SQL%ROWCOUNT;*
- *dbms_output.put_line('Salaries for ' || var_rows || 'employees are updated');*
- *END IF;*
- *END;*
- In the above PL/SQL Block, the salaries of all the employees in the ‘employee’ table are updated.
- If none of the employee’s salary are updated we get a message 'None of the salaries where updated'.
- Else we get a message like for example, 'Salaries for 1000 employees are updated' if there are 1000 rows in ‘employee’ table.

Explicit Cursors

- An explicit cursor is defined in the declaration section of the PL/SQL Block.
- It is created on a SELECT Statement which returns more than one row.
- We can provide a suitable name for the cursor.
- **The General Syntax for creating a cursor is as given below:**

  DECLARE cursor_name CURSOR FOR select_statement;

- *cursor_name – A suitable name for the cursor.*

- *select_statement – A select query which returns multiple rows.*

- How to use Explicit Cursor?
- There are four steps in using an Explicit Cursor.
- DECLARE the cursor in the declaration section.
- OPEN the cursor in the Execution Section.
- FETCH the data from cursor into PL/SQL variables or records in the Execution Section.
- CLOSE the cursor in the Execution Section before you end the PL/SQL Block.

- 1) Declaring a Cursor in the Declaration Section:

- *DECLARE emp_cur CURSOR FOR*
- *SELECT * FROM emp_tbl WHERE salary > 5000;*

- In the above example we are creating a cursor ‘emp_cur’ on a query which returns the records of all the employees with salary greater than 5000.

- Here ‘emp_tbl’ in the table which contains records of all the employees.

- 2) Accessing the records in the cursor:

- Once the cursor is created in the declaration section we can access the cursor in the execution section of the PL/SQL program.

- **How to access an Explicit Cursor?**

- These are the three steps in accessing the cursor.

  1) Open the cursor.

  2) Fetch the records in the cursor one at a time.

  3) Close the cursor.

- General Syntax to open a cursor is:
- *OPEN cursor_name;*

- General Syntax to fetch records from a cursor is:
- *FETCH cursor_name INTO record_name;*
- OR
- FETCH cursor_name INTO variable_list;

- General Syntax to close a cursor is:
- *CLOSE cursor_name;*

- When a cursor is opened, the first row becomes the current row.
- When the data is fetched it is copied to the record or variables and the logical pointer moves to the next row and it becomes the current row.
- On every fetch statement, the pointer moves to the next row.
- If you want to fetch after the last row, the program will throw an error.
- When there is more than one row in a cursor we can use loops along with explicit cursor attributes to fetch all the records.

General Form of using an explicit cursor is:

- DECLARE
  - variables;
  - records;
  - create a cursor;
- BEGIN
  - OPEN cursor;
  - FETCH cursor;
  - process the records;
  - CLOSE cursor;
- END;

- Lets Look at the example below

Example 1:

- *DECLARE*
  - *emp_rec emp_tbl%rowtype;*
  - *CURSOR emp_cur IS*
  - *SELECT * FROM emp_tbl*
  - *WHERE salary > 10;*
- *BEGIN*
  - *OPEN emp_cur;*
  - *FETCH emp_cur INTO emp_rec;*
  - *dbms_output.put_line (emp_rec.first_name || ' ' || emp_rec.last_name);*
  - *CLOSE emp_cur;*
- *END;*

creating a record 'emp_rec' of the same structure as of table 'emp_tbl'

declaring a cursor 'emp_cur' from a select query

opening the cursor

fetching the cursor to the record

displaying the first_name and last_name of the employee in the record emp_rec

closing the cursor

- **What are Explicit Cursor Attributes?**

- Oracle provides some attributes known as Explicit Cursor Attributes to control the data processing while using cursors.

- We use these attributes to avoid errors while accessing cursors through OPEN, FETCH and CLOSE Statements.

- **When does an error occur while accessing an explicit cursor?**

- a) When we try to open a cursor which is not closed in the previous operation.
- b) When we try to fetch a cursor after the last operation.

- These are the attributes available to check the status of an explicit cursor.

Attributes	Return values	Example
%FOUND	TRUE, if fetch statement returns at least one row.	Cursor_name%FOUND
	FALSE, if fetch statement doesn't return a row.	
%NOTFOUND	TRUE, , if fetch statement doesn't return a row.	Cursor_name%NOTFOUND
	FALSE, if fetch statement returns at least one row.	
%ROWCOUNT	The number of rows fetched by the fetch statement	Cursor_name%ROWCOUNT
	If no row is returned, the PL/SQL statement returns an error.	
%ISOPEN	TRUE, if the cursor is already open in the program	Cursor_name%ISNAME
	FALSE, if the cursor is not opened in the program.	