Parul® University

CHAPTER-4

PL/SQL Cursor & Trigger

PL/SQL

PL/SQL (Procedural Language/Structured Query Language) is a **block-structured language** developed by **Oracle** that allows developers to combine the power of **SQL** with procedural programming constructs.
The PL/SQL language enables efficient data manipulation and control-flow logic, all within the **Oracle Database**.

•PL/SQL stands for Procedural Language extensions to the Structured Query Language (SQL).

•PL/SQL is a combination of SQL along with the procedural features of programming languages.

•Oracle uses a PL/SQL engine to process the PL/SQL statements.

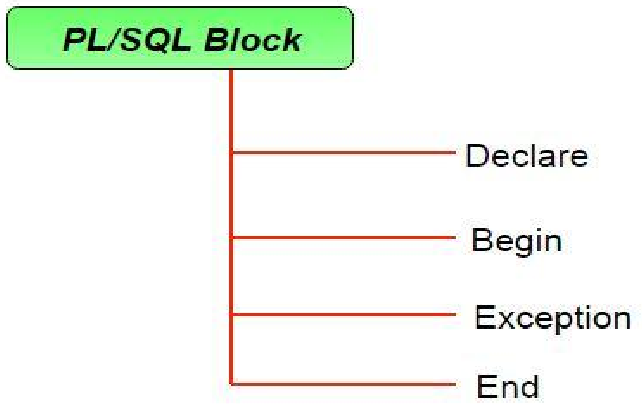
PL/SQL

•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.

Structure of PL/SQL

PL/SQL extends SQL by adding constructs found in **procedural languages**, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other.

PL/SQL



```
DECLARE declaration statements;
BEGIN executable statements
EXCEPTIONS exception handling statements
END;
```

PL/SQL

•Declare section starts with **DECLARE** keyword in which variables, constants, records as cursors can be declared which stores data temporarily. It basically consists definition of PL/SQL identifiers. This part of the code is optional.

•Execution section starts with **BEGIN** and ends with **END** keyword.This is a mandatory section and here the program logic is written to perform any task like loops and conditional statements. It supports all DML commands, DDL commands and SQL*PLUS built-in functions as well.

•Exception section starts with **EXCEPTION** keyword.This section is optional which contains statements that are executed when a run-time error occurs. Any exceptions can be handled in this section.

Parul® University

DIGITAL LEARNING CONTENT

PL/SQL EXAMPLE

```
DECLARE
 var varchar2(40) := ‘Hello World’;
 BEGIN
 dbms_output.put_line(var);
 END;
 /
```

```
OUTPUT:
Hello World
```

PL/SQL

dbms_output.put_line **:** This command is used to direct the PL/SQL output to a screen.

2.<u>Using Comments</u>: Like in many other programming languages, in PL/SQL also, comments can be put within the code which has no effect in the code. There are two syntaxes to create comments in PL/SQL :

2. **<u>Single Line Comment:</u>** To create a single line comment , the symbol – – is used.
3. **<u>Multi Line Comment:</u>** To create comments that span over several lines, the symbol /* and */ is used.

Introduction to Cursor

Relational To handle a result set inside a stored procedure, you use a cursor. A cursor allows you to iterate a set of rows returned by a query and process each row accordingly.

MySQL cursor is read only, non-scrollable and asensitive.

Read only: you cannot update data in the underlying table through the cursor.

Non-scrollable: you can only fetch rows in the order determined by the SELECT statement. You cannot fetch rows in the reversed order. In addition, you cannot skip rows or jump to a specific row in the result set.

Introduction to Cursor

Asensitive: there are two kinds of cursors: asensitive cursor and insensitive cursor.
An asensitive cursor points to the actual data,
whereas an **insensitive cursor** uses a temporary copy of the data.
An asensitive cursor performs faster than an insensitive cursor because it does not have to make a temporary copy of data. However, any change that made to the data from other connections will affect the data that is being used by an asensitive cursor. MySQL cursor is asensitive.
You can use MySQL cursors in stored procedures, stored functions and triggers

Introduction to Cursor

First, you have to declare a cursor by using the DECLARE statement:
DECLARE cursor_name CURSOR FOR SELECT_statement;

The cursor declaration must be after any variable declaration. If you declare a cursor before variables declaration, MySQL will issue an error. A cursor must always be associated with a SELECT statement.
Next, you open the cursor by using the OPEN statement. The OPEN statement initializes the result set for the cursor therefore you must call the OPEN statement before fetching rows from the result set.

OPEN cursor_name

Working with cursor

Then, you use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.

FETCH cursor_name **INTO variables list;**

After that, you can check to see if there is any row available before fetching it.

Finally, you call the CLOSE statement to deactivate the cursor and release the memory associated with it as follows:

CLOSE cursor_name;

When the cursor is no longer used, you should close it.

Working with cursor

When working with MySQL cursor, you must also declare a NOT FOUND handler to handle the situation when the cursor could not find any row. Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set. When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.

To declare a NOT FOUND handler, you use the following syntax:

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
```

Where finished is a variable to indicate that the cursor has reached the end of the result set.

Notice that the handler declaration must appear after variable and cursor declaration inside the stored procedures.

Parul® University

DIGITAL LEARNING CONTENT

Working with cursor

The following diagram illustrates how cursor works.

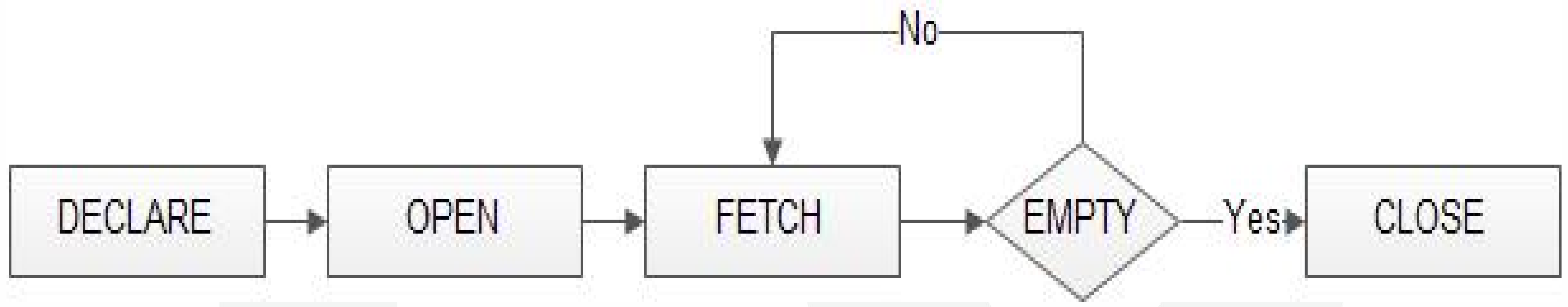


*Fig :*Draw Self

Parul® University

DIGITAL LEARNING CONTENT

Cursor Examples:

We are going to develop a stored procedure that builds an email list of all employees in the employees table in the MySQL sample database.
First, we declare some variables, a cursor for looping over the emails of employees, and a NOT FOUNDhandler:

we declare some variables, a cursor for looping over the emails of employees, and a NOT FOUNDhandler:

```
DECLARE finished INTEGER DEFAULT 0;

DECLARE email varchar(255) DEFAULT "";
```

Cursor Examples:

```
-- declare cursor for employee email
DECLARE email_cursor CURSOR FOR
SELECT email FROM employees;

-- declare NOT FOUND handler
DECLARE CONTINUE HANDLER
FOR NOT FOUND SET finished = 1;
```

Next, we open the email_cursor by using the OPEN statement:

```
OPEN email_cursor;
```

Cursor Examples:

Then, we iterate the email list, and concatenate all emails where each email is separated by a semicolon(;):

```
get_email: LOOP
FETCH email_cursor INTO v_email;
IF v_finished = 1 THEN
LEAVE get_email;
END IF;
-- build email list
SET email_list = CONCAT(v_email,";",email_list);
END LOOP get_email;
```

Cursor Examples:

inside the loop we used the v_finished variable to check if there is any email in the list to terminate the loop.

inside the loop we used the v_finished variable to check if there is any email in the list to terminate the loop.

Cursor Examples:

The build_email_list stored procedure is as follows:

```
DELIMITER $$
 CREATE PROCEDURE build_email_list (INOUT email_list varchar(4000))
BEGIN
DECLARE v_finished INTEGER DEFAULT 0;
DECLARE v_email varchar(100) DEFAULT "";
 -- declare cursor for employee email
DEClARE email_cursor CURSOR FOR
SELECT email FROM employees;
-- declare NOT FOUND handler
DECLARE CONTINUE HANDLER
FOR NOT FOUND SET v_finished = 1;
```

Cursor Examples:

```
OPEN email_cursor;
 get_email: LOOP
 FETCH email_cursor INTO v_email;
IF v_finished = 1 THEN
LEAVE get_email;
END IF;
 -- build email list
SET email_list = CONCAT(v_email,";",email_list);
 END LOOP get_email;
 CLOSE email_cursor;
 END$$
 DELIMITER ;
```

To execute a procedure

To execute a procedure

```
SET @email_list = "";

CALL build_email_list(@email_list);

SELECT @email_list;
```

Triggers

By definition, a trigger or database trigger is a stored program that is executed automatically to respond to a specific event associated with table e.g., insert, update or delete.
Database trigger is powerful tool for protecting the integrity of the data in your MySQL databases. Database triggers are very useful to automate some database operations such as audit logging

A SQL trigger is a special type of stored procedure. It is special because it is not called directly like a stored procedure.
The main difference between a trigger and a stored procedure is that a trigger is called automatically when a data modification event is made against a table whereas a stored procedure must be called explicitly

Advantages of Triggers

SQL triggers provide an alternative way to check the integrity of data.

SQL triggers can catch errors in business logic in the database layer.

SQL triggers provide an alternative way to run scheduled tasks. By using SQL triggers, you don’t have to wait to run the scheduled tasks because the triggers are invoked automatically before or after a change is made to the data in the tables.

SQL triggers are very useful to audit the changes of data in tables.

Disadvantages of Triggers

SQL triggers only can provide an extended validation and they cannot replace all the validations. Some simple validations have to be done in the application layer. For example, you can validate user's inputs in the client side by using JavaScript or in the server side using server side scripting languages such as JSP, PHP, ASP.NET, Perl, etc.

SQL triggers are invoked and executed invisibly from client-applications therefore it is difficult to figure out what happen in the database layer.

SQL triggers may increase the overhead of the database server.

CREATE TRIGGER Syntax:

CREATE TRIGGER Syntax

```
CREATE [DEFINER = { user | CURRENT_USER }] TRIGGER trigger_name
trigger_time trigger_event ON tbl_name FOR EACH ROW trigger_body

trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
```

CREATE TRIGGER Syntax:

This statement creates a new trigger. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. The trigger becomes associated with the table named tbl_name, which must refer to a permanent table. You cannot associate a trigger with a TEMPORARY table or a view. CREATE TRIGGER was added in MySQL 5.0.2.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

CREATE TRIGGER Syntax:

There **cannot be multiple triggers for a given table that have the same trigger event and action time.**

For example, you cannot have two BEFORE UPDATE triggers for a table. But you can have a BEFORE UPDATE and a BEFORE INSERT trigger, or a BEFORE UPDATE and an AFTER UPDATE trigger.

Drop Triggers Syntax

DROP TRIGGER Syntax
DROP TRIGGER [IF EXISTS] **[schema_name.]trigger_name**

This statement drops a trigger. The schema (database) name is optional. If the schema is omitted, the trigger is dropped from the default schema. DROP TRIGGER was added in MySQL 5.0.2. Its use requires the SUPER privilege.

Use IF EXISTS to prevent an error from occurring for a trigger that does not exist