# PL/SQL

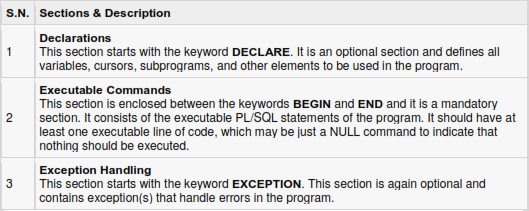
* PL/SQL stands for Procedural Language/Structured Query Language
* It is used to overcome limitations of SQL.
* It is superset of SQL

## Advantages of PL/SQL

* + **Block structure**: PL/SQL consist of block 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 consist of procedural constructs such as conditional statements (if, if else, nested if, else if ladder) and loops (for, while, do while).
  + **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 PL/SQL program. Once an exception is c aught, specific action can be taken depending upon the type of the exception or it can be displayed to the user with message.

## PL/SQL Block Structure

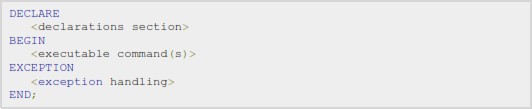
PL/SQL is a block-structured language, meaning that PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts:



Every PL/SQL statement ends with a semicolon (;).

PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END.

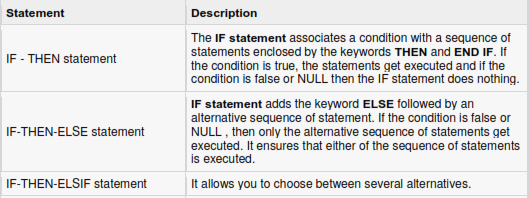
Here is the basic structure of a PL/SQL block:



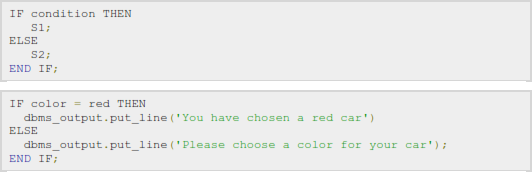
## To create SQL file : ed filename.sql To run it: @filename.sql

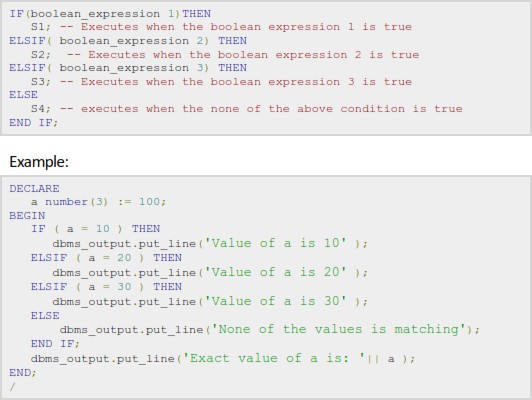
***/***

# Conditional Statements

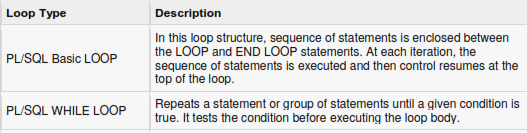


## IF-THEN statement:

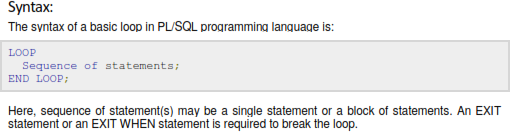
**IF-THEN-ELSE statement:**

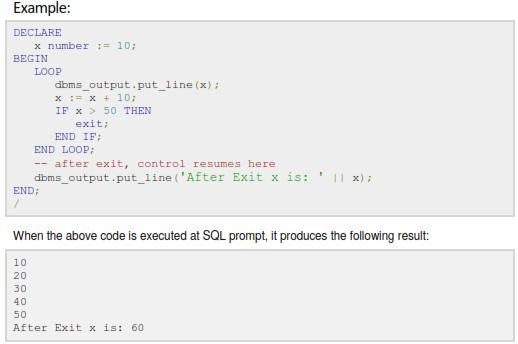
**IF-THEN-ELSIF statement:**

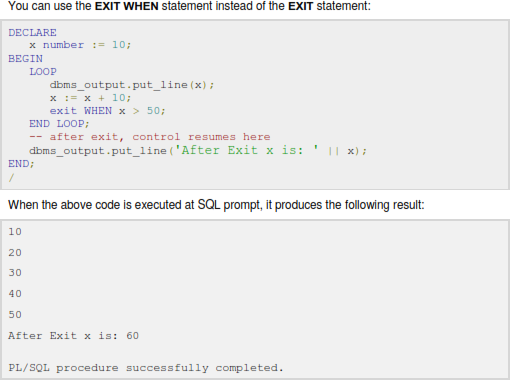
**Looping in PL/SQL**



**PL/SQL Basic LOOP:**







**PL/SQL Subprogram**

* A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'.
* A subprogram can be invoked by another subprogram or program which is called the calling program.
* PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms:
* **Functions**: these subprograms return a single value, mainly used to compute and return a value.
* **Procedures**: these subprograms do not return a value directly, mainly used to perform an action.

## Procedure (Stored Procedure)

* A stored procedure (proc) is a group of PL/SQL statements that performs specific task.
* A procedure has two parts, header and body.
* The header consists of the name of the procedure and the parameters passed to the procedure.
* The body consists of declaration section, execution section and exception section.
* A procedure may or may not return any value. A procedure may return more than one value.

General Syntax to create a procedure

CREATE [OR REPLACE] PROCEDURE proc\_name [list of parameters]

IS

Declaration section

BEGIN

Execution section

EXCEPTION

Exception section

END;

Explanation

Create:-It will create a procedure.

Replace:- It will re-create a procedure if it already exists. We can pass parameters to the procedures in three ways.

IN-parameters: - These types of parameters are used to send values to stored procedures

OUT-parameters: - These types of parameters are used to get values from stored procedures. This is similar to a return type in functions but procedure can return values for more than one parameters.

IN OUT-parameters: - This type of parameter allows us to pass values into a procedure and get output values from the procedure.

**IS** indicates the beginning of the body of the procedure. The code between IS and BEGIN forms the Declaration section.

**Begin:-**It contains the executable statement.

**Exception:-** It contains exception handling part. This section is optional.

**End:-** It will end the procedure.

The syntax within the brackets [ ] indicates that they are optional.

By using CREATE OR REPLACE together the procedure is created if it does not exist and if it exists then it is replaced with the current code.

* + There are two ways to execute a procedure.
  1. From the SQL prompt.

Syntax: EXECUTE [or EXEC] procedure\_name (parameter);

* 1. Within another procedure – simply use the procedure name. Syntax: procedure\_name (parameter);

## *Advantages of procedure*

* + **Security:-** We can improve security by giving rights to selected persons only.
  + **Faster Execution:-** It is precompiled so compilation of procedure is not required every time you call it.
  + **Sharing of code:-** Once procedure is created and stored, it can be used by more than one user.
  + **Productivity:-** Code written in procedure is shared by all programmers. This eliminates redundant coding by multiple programmers so overall improvement in productivity.

## Function

Where,

* 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 parameters. 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.

## Difference between Function and Stored Procedure

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **User Defined Function** | **Stored Procedure** |
| 1 | Function must return a value. | Stored Procedure may or not return values. |
| 2 | Will allow only Select statements, it will not allow us to use DML statements. | Can have select statements as well as DML statements such as insert, update, delete and so on |
| 3 | It will allow only input parameters, doesn't support output parameters. | It can have both input and output parameters. |
| 4 | Stored Procedures can't be called from a function. | Function can be called from Stored Procedures. |

**Cursor**

* + Cursors are database objects used to traverse the results of a select SQL query.
  + It is a temporary work area created in the system memory when a select SQL statement is executed.
  + This temporary work area is used to store the data retrieved from the database, and manipulate this data.
  + It points to a certa in location within a record set and allow the operator to move forward (and sometimes backward, depending upon the cursor type).
  + We can process only one record at a time.
  + The set of rows the cursor holds which is called the *active* set (active data set).
  + Cursors are often criticized for their high overhead.
  + There are two types of cursors in PL/SQL:

### Implicit cursors:

* + These are created by default by ORACLE itself 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.
  + We cannot use implicit cursors for user defined work.

### Explicit cursors:

* + Explicit cursors are user defined cursors written by the developer.
  + They can be created when a SELECT statement th at returns more than one row is executed.
  + 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.

|  |  |  |
| --- | --- | --- |
| *Attributes* | *Return Value* | *Example* |
| %FOUND | It will return TRUE, if the DML statements like INSERT,  DELETE, UPDATE and SELECT will affect at least one row else return FALSE | SQL%FOUND |
| %NOTFOUND | It will return FALSE, if the DML statements like INSERT, DELETE, UPDATE and SELECT will affect at least one row  else return TRUE | SQL%NOTFOUND |
| %ROWCOUNT | Return the number of rows affected by the DML  operations INSERT, DELETE, UPDATE, SELECT | SQL%ROWCOUNT |
| %ISOPEN | It will return true if cursor is open else return false. | SAL%OPEN |

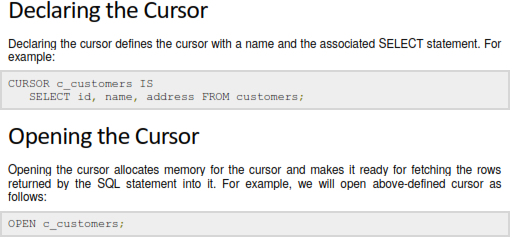
## Explicit Cursor

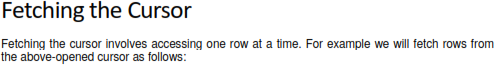
* Explicit cursors are programmer defined cursors for gaining more control over the context area.
* An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

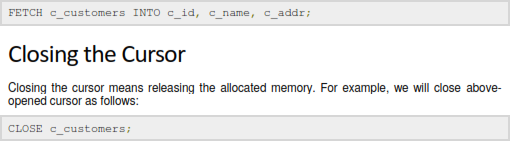
The syntax for creating an explicit cursor is



* Working with an explicit cursor involves four steps:
* Declaring the cursor for initializing in the memory
* Opening the cursor for allocating memory
* Fetching the cursor for retrieving data
* Closing the cursor to release allocated memory







# Trigger

* Triggers are stored programs, which are automatically executed or fired when some events occur.

## Advantages of Trigger

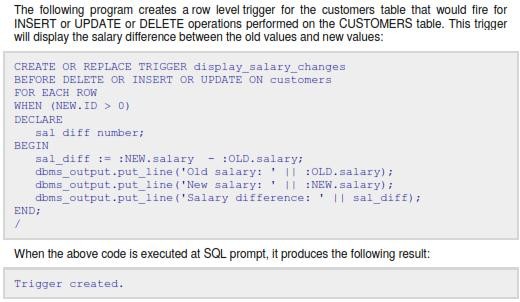
Triggers can be written for the following purposes:

* Generating some derived column values automatically
* Enforcing referential integrity
* Event logging and storing information on table access
* Auditing
* Synchronous replication of tables
* Imposing security authorizations
* Preventing invalid transactions



Where,

* CREATE [OR REPLACE] TRIGGER trigger\_name: 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.



Triggering a Trigger