# PL/SQL Block

PL/SQL is a superset of SQL. PL/SQL is a block structured language that enables developers to combine the power of SQL with procedural statements. PL/SQL bridges the gap between database technology and procedural programming language.
PL/SQL permits the creation of structured logical blocks of code that describe processes, which have to be applied to data.

The sections of PL/SQL block are:
1) The declare section
2) The master begin and end section that also (optionally) contains an Exception section.

Each of these is explained below:

| | |
|---|---|
| DECLARE | Declaration of memory variables, constants, cursors etc in PL/SQL. |
| BEGIN | SQL executable statements, PL/SQL executable statements. |
| EXCEPTION | SQL or PL/SQL code to handle errors that may arise during execution of the code block between BEGIN and EXCEPTION section. |
| END | This marks the end of a PL/SQL block. |

## Displaying user message on screen:

Programming tools require a method through which messages can be displayed on screen.

DBMS_OUTPUT is a package that includes a number of procedures and functions that accumulate information in a buffer so that it can be retrived later. These functions can also be used to display message.

PUT_LINE puts a piece of information in the package buffer followed by an end-of line marker. It can be used to display message.

To display messages, the SERVEROUTPUT should be SET ON.

Control Structure

1) conditional control
2) Iterative control
3) Sequential control

Conditional control :

```
IF <condition> THEN
    <action>
ELSEIF <condition> THEN
    <action>
ELSE
```

<ACTION>
END IF;

Iterative control:

Simple loop:

LOOP
<Sequence of statements>
END LOOP;

While loop:

WHILE <condition>
LOOP
<ACTION>
END LOOP

For loop:

FOR variable IN [REVERSE] start..end
LOOP
<ACTION>
END LOOP;

Sequential control:

Goto statement:

GOTO <codeblock name>

## Error Handling in PL/SQL :

Exception

```
WHEN <ExceptionName> THEN
    <User Defined Action To Be Carried Out>
```

Oracle Named Exception Handlers :

| | |
|---|---|
| LOGIN_DENIED | Raised when an invalid username/password was used to log onto oracle. |
| NO_DATA_FOUND | Raised when a select statement return zero rows. |
| PROGRAM_ERROR | Raised when PL/SQL has an internal problem. |
| TOO_MANY_ROWS | Raised when a select statement returns more than one row. |
| VALUE_ERROR | Raised when the data type or data size is invalid. |

# Cursor

The oracle engine uses a work area for its internal processing in order to execute an SQL Statement. This work area is private to SQL's operations and is called a Cursor.

## Types of cursors:

Cursors are classified depending on the circumstances under which they are opened. If the oracle engine opened a cursor for its internal processing it is known as Implicit cursor.

A cursor can also be opened for processing data through a PL/SQL block, on demand. Such a user-defined cursor is known as an Explicit cursor.

## Implicit cursor attributes:

| | |
|---|---|
| %ISOPEN | Returns TRUE if cursor is open, FALSE otherwise. |
| %FOUND | Return TRUE if record was fetched successfully, FALSE otherwise. |
| %NOTFOUND | Return TRUE if record was not fetched successfully. FALSE otherwise. |
| %ROWCOUNT | Returns number of records processed from the cursor. |

## Explicit Cursor :

A cursor is defined in the declarative part of a PL/SQL block.
The three commands used to control the cursor subsequently are open, fetch and close.

Syntax :

```
CURSOR cursor_name IS SELECT statement;
```

Opening a cursor :

```
OPEN cursorName;
```

Fetching a record from the cursor :

```
FETCH cursorName INTO Variable1,
Variable2, ... ;
```

Closing a cursor :

```
CLOSE cursorName;
```

Cursor for loops :

```
FOR memory_variable IN cursorName
```

Parameterized cursors :

Declaring a parameterized cursor :

CURSOR cursor_name (VariableName Datatype)
IS <SELECT statement ... >

Opening a parameterized cursor

OPEN cursorName (value/variable / Expression)

# Procedures

A procedure is a logically grouped set of SQL and PL/SQL statements that perform a specific task.

Procedure are made up of :
A declarative part
An executable part
An optional exception-handling part

Syntax to show the status of a procedure :

```
SELECT <object Name>, <object Type>, <Status>
FROM <User objects>
WHERE <objectType> = <'PROCEDURE'>;
```

creating Stored procedure

```
CREATE OR REPLACE PROCEDURE [schema]
<procedure Name>
(<argument> {IN, OUT, IN OUT} <Datatype>, )
{IS, AS}
<Variable> declarations;
<constant> declarations;

BEGIN
    <PL/SQL subprogram body>;
```

EXCEPTION
    <EXCEPTION PL/SQL block>;
END;

The keywords and the parameters used for creating database procedure are explained below:

REPLACE — Recreates a procedure if it already exists.
This option is used to change the definition of an existing procedure without dropping, recreating and re-generating object privileges previously granted on it.
If a procedure is redefined the Oracle engine recompile it.

Schema — Is the schema to contain the procedure.
The Oracle engine takes the default schema to be the current schema, if it is omitted.

Procedure — Is the name of the procedure to be created.

Argument — Is the name of an argument to the procedure. Parentheses

can be omitted if no arguments are present.

IN — Indicates that the parameter will accept a value from the user.

OUT — Indicates that the parameter will return a value to the user.

IN OUT — Indicates that the parameter will either accept a value from the user or return a value to the user.

Data type — Is the data type of an argument. It supports any data type supported by PL/SQL.

PL/SQL Subprogram body — Is the definition of procedure consisting of PL/SQL statements.

Deleting a stored Procedure :

```
Drop PROCEDURE procedure_name;
```

# Function

A function is a logically grouped set of SQL and PL/SQL statements that perform a specific task.

Functions are made up of :
A declarative part
An executable part
An optional exception-handling part.

Creating a Function :

```
CREATE OR REPLACE FUNCTION [schema..]
<Function Name>
(<Argument> IN <Data type>,...)
RETURN <Data type> {IS, AS}
<variable> declaration;
<constant> declaration;

BEGIN
   <PL/SQL subprogram body>;

EXCEPTION
   <Exception PL/SQL block>;

END;
```

The Keywords and the parameters used for creating database functions are explained below:

REPLACE — Recreates the function if it already exists. This option is used to change the definition of an existing function without dropping, recreating and re-granting object privileges previously granted on it. If a function is redefined, Oracle recompiles it.

Schema — Is the Schema to contain the function. Oracle takes the default Schema to be the current schema, if it is omitted.

Function — Is the name of the function to be created.

Argument — Is the name of an argument to the function. Parenthesis can be omitted if no arguments are present.

IN — Indicates that the parameter will accept a value from the user.

RETURN data type : Is the data type of the function's return value. Because every function must return a value, this clause is required. It supports any data type supported by PL/SQL.

PL/SQL Subprogram body — Is the definition of function consisting of PL/SQL statements

Deleting a Stored function :

```
DROP FUNCTION Function_name;
```

# Packages

A package is an oracle object, which holds other objects within it.
objects commonly held within a package are procedures, functions, variables, constants, cursors and exceptions.

## Package Specification:

The package Specification contains:
→ Name of the package
→ Name of the data types of any arguments.
→ This declaration is local to database and global to the package.

Creating package:

```
CREATE or REPLACE PACKAGE <Package_name>
as <declaration of variable, function,
Procedure> ;
...
END <package_names> ;
```

After specification is created, the body of the package needs to be created.
The body of the package is a collection of detailed definitions of the objects that were declared in the specification.

CREATE OR REPLACE PACKAGE BODY
<package name> AS
<PROCEDURE, Function definition> IS

BEGIN
<Body of PLISQL procedure>

END package procedure name ;

Invoking Package :

When a package is invoked, the oracle engine performs three steps to execute it:
1) Verify user access
2) Verify procedure validity
3) Execute

To reference a package subprogram and objects, use dot notation.

The syntax for Dot notation :

PackageName . TypeName
PackageName . ObjectName
PackageName . Subprogram Name

In this syntax,
PackageName is the name of declared package
Type_name is name of user defined type.

- Objectname is the name of the constant or variable that is declared by the user.
- Subprogram is the name of Procedure or function contained in the package body.

Alterations to an existing package:

ALTER PACKAGE <packageName> COMPILE PACKAGE;

# Triggers

Database triggers are database objects created via the SQL* plus tool on the client and stored on the server in the Oracle engine's System Table.

These database objects consist of the following distinct sections:

A named database event

A PL/SQL block that will execute when the event occurs

Syntax for creating a Trigger:

```
CREATE OR REPLACE TRIGGER [schema.] <TriggerName>
    {BEFORE, AFTER}
    {DELETE, INSERT, UPDATE [OF column,...]}
ON [schema] <Table Name>
    [REFERENCING {OLD AS old, NEW AS new}]
    [FOR EACH ROW [WHEN condition]]
DECLARE
    <variable declarations>
    <constant declarations>
BEGIN
    <PL/SQL subprogram body>;
EXCEPTION
    <Exception PL/SQL block>;
END;
```

The keywords and the parameters used for creating database triggers are explained below:

| | |
|---|---|
| OR REPLACE | Recreates the trigger if it is already exists. |
| Schema | Is the schema, which contains the trigger. |
| TriggerName | Is the name of the trigger to be created. |
| BEFORE | Indicates that the oracle engine fires the trigger before executing the triggering statement. |
| AFTER | Indicates that the oracle engine fires the trigger after executing the triggering statement. |
| DELETE | Indicates that oracle engine fires trigger when DELETE Statement removes a row from the table. |
| INSERT | Fires trigger when INSERT Statement adds a row to table. |

UPDATE — fires a trigger when UPDATE statement changes a value.

ON — Specifies the schema and name of the table, which the trigger is to be created.

REFERENCING — Specifies correlation names.

FOR EACH ROW — Designates a trigger to be a row trigger.

WHEN — Specifies the trigger restriction.

PL/SQL BLOCK — Block that the oracle engine executes when the trigger is fired.

Deleting a Trigger :

```
DROP TRIGGER <TriggerName>;
```