# Chapter 2 : PL/SQL

2.1 Advantages of PL/SQL

2.2 Data types

2.2 Control structure
- Conditional
- Iterative
- Sequential

2.3 Concepts of error handling
- Predefined Exceptions
- User defined Exceptions

2.4 Cursor Management
- Static
- Dynamic

2.5 Procedure,Function and Packages
- Procedure and Function
- Package specification, Package body, Advantage of package

# What is PL/SQL

- PL/SQL stands for Procedural Language extension of SQL.

  PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

- Oracle uses a PL/SQL engine to processes the PL/SQL statements. A PL/SQL code can be stored in the client system (client-side) or in the database (server-side).
- depending upon the type of the exception or it can be displayed to the user with a message.

# 2.1 ADVANTAGES OF PL/SQL

Text Book Name : Ivan Bayross , Ref. plsql-tutorial.com

- ***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.
- **Portability :-** it means , program can be transferred and executed from any other computer, where oracle is operational.
- **Support to variables :-** pl/sql supports variables to store intermediate results of a query.
- **Sharing of Code :-** it allows user to store compiled code in database and it can be executed from other programming languages such as java.

# A Simple PL/SQL Block:

- Each PL/SQL program consists of SQL and PL/SQL statements which from a PL/SQL block.
- A PL/SQL Block consists of three sections:
  - The Declaration section (optional).
  - The Execution section (mandatory).
  - The Exception (or Error) Handling section (optional).
-

```
DECLARE                    --Optional

<Declarations Section>

BEGIN                      --Mandatory

        <Executable Commands>

EXCEPTIONS                 --
Optional

        <Exception Handling>

END;                       --Mandatory
```

# 2.2 DATA TYPES

Text Book Name : Ivan Bayross , Ref. plsql-tutorial.com

| Category | Data type | Sub types/Values |
|---|---|---|
| **Numerical** | NUMBER | BINARY_INTEGER, DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INTEGER, INT, NATURAL, NUMERIC, POSITIVE, REAL, SMALLINT, NATURAL |
| **Character** | CHAR,LONG,VARCHAR2 | CHARACTER, VARCHAR, STRING, NCHAR, NVARCHAR2 |
| **Date** | DATE | |
| **Binary** | RAW,LONG ROW | |
| **Boolean** | BOOLEAN | Can have values like TRUE,FALSE and NULL |
| **RowID** | ROWID | Store values of address location of each record. |

# Anchored Data type :-

- Datatype for variable is determined based on the datatype of other object.
- This object can be other variable,or a column of the table.
- This provides the ability to match the data types of the variables with the data types of the columns defined in the database.

- **SYNTAX :-**

variablename object%type [ not null ] :=initial value;

- **EXAMPLE :-**

no Account.accno%type;

bal Account.balance%type; .....etc.

# PL/SQL Variables

- These are placeholders that store the values that can change through the PL/SQL Block.

## How to declare Variables

variable_name datatype [NOT NULL := value ];

*variable_name* is the name of the variable.
*datatype* is a valid PL/SQL datatype.
NOT NULL is an optional specification on the variable.
*value* or DEFAULT *value*is also an optional specification, where you can initialize a variable.
Each variable declaration is a separate statement and must be terminated by a semicolon.

## How to declare constant

constant_name CONSTANT datatype := VALUE;

# How assigning a Value to variable

## 1.) By using assignment operator :-

variblename :=value;

## 2.) By reading from key-board :-

variablename := &variablename ;

## 3.) Fetching table data value into variables:-

select col1,col2,...coln into var1,var2,...varn from tablename where condition;

# How to Display message on Screen

dbms_output.put_line( message );

- Here, dbms_output is a package, which provides functions to accumulate information in a buffer.
- A put_line is a function which displays messages on the screen.

# How to make a Comment in PL/SQL

1.) –( double hyphen or double dash) Treats single line comment.

2.) /*……..*/ Treats multiple lines comments.

# How to create and Execute a PL/SQL block

An edit command can be used on SQL prompt,

- **SYNTAX :-**

  EDIT FILENAME

- **EXAMPLE :-**

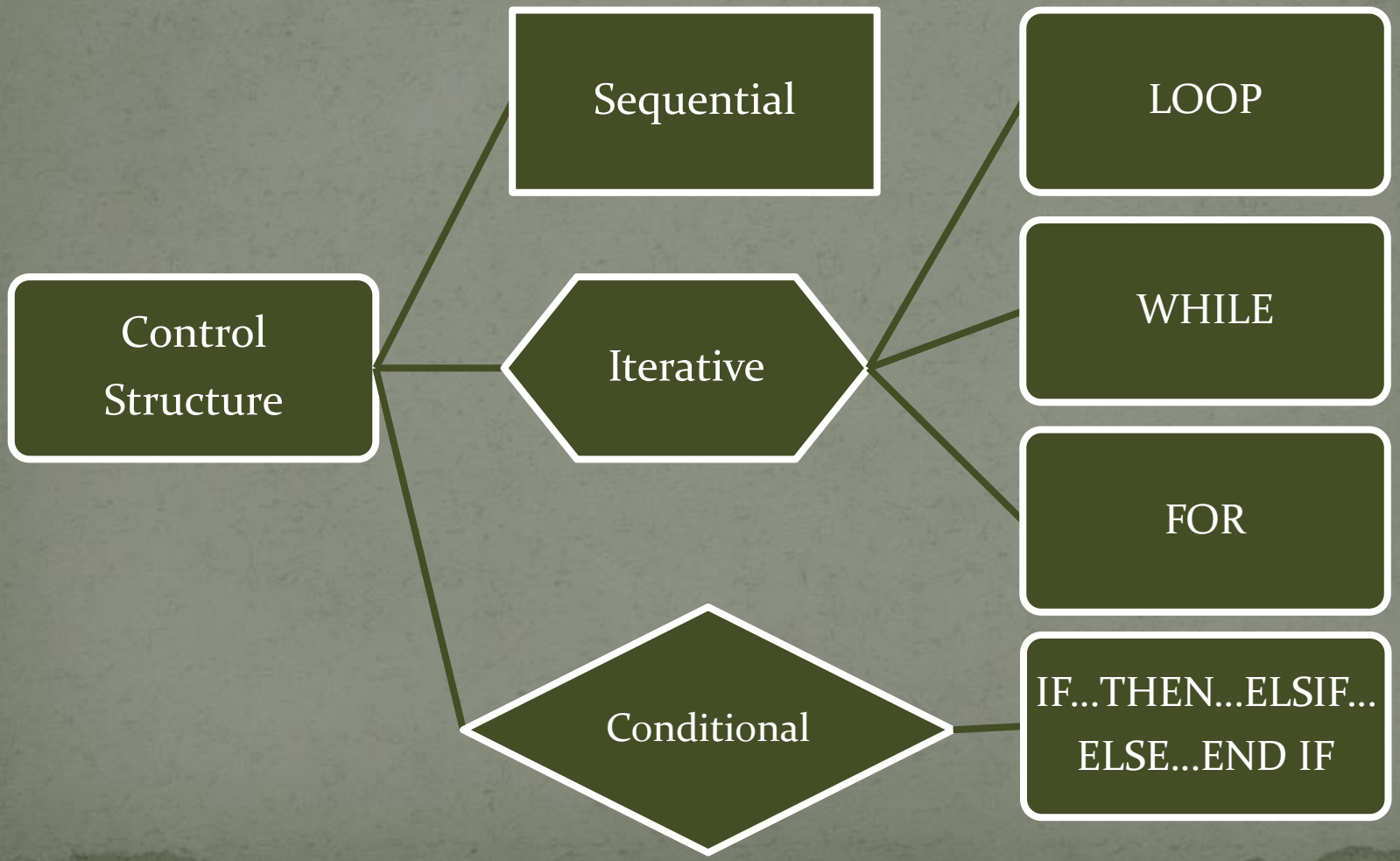
  Edit e:/test/test.sql ;

  This command creates and opens a file named ‘test.sql’.
  To execute this block,

  Run filename;
  start filename; OR@ filename;

# 2.3 CONTROL STRUCTURES

Text Book Name : Ivan Bayross , Ref. plsql-tutorial.com



# Conditional

- To control the execution of block of code based on some conditions, PL/SQL provides the IF statements.

- **SYNTAX :-**

If condition then

----execute commands.....

Elseif condition

----execute commands.....

else

----execute commands.....

End if;

# Iterative

- An iterative control Statements are used when we want to repeat the execution of one or more statements for specified number of times. These are similar to those in

- There are three types of loops in PL/SQL:
  - Simple Loop
  - While Loop
  - For Loop

1) **Simple Loop :-**

```
LOOP
        statements;
EXIT;
{or EXIT WHEN condition;}
END LOOP;
```

Initialize a variable before the loop body.
Increment the variable in the loop.
Use a EXIT WHEN statement to exit from the Loop. If you use a EXIT statement without WHEN condition, the statements in the loop is executed only once.

2) **While Loop :-**

```
WHILE <condition>
LOOP
        statements;
END LOOP;
```

## 3) FOR Loop :-

```
FOR variable in [reverse] start..end
LOOP
      statements;
END LOOP;
```

Variable is a loop control variable.

A start and end specifies the lower and upper bound for the loop control variable.

- The counter variable is implicitly declared in the declaration section, so it's not necessary to declare it explicitly.
- The counter variable is incremented by 1 and does not need to be incremented explicitly.
- EXIT WHEN statement and EXIT statements can be used in FOR loops but it's not done often.

# Sequential Control

- Normally, execution proceeds sequentially with in the block of the code.

But, this can be changed using GOTO statement,

- **SYNTAX :-**

GOTO jumphere;

....

....

<<jumphere>>

This jump is unconditional.

# 2.4 EXCEPTION HANDLING

Text Book Name : Ivan Bayross , Ref. plsql-tutorial.com

- An Exception is a run time unusual condition that a program may encounter while executing.
- It consider as an error arising at run time.
- PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly. When an exception occurs a messages which explains its cause is recieved.

- PL/SQL Exception message consists of three parts.

1) Type of Exception

2) An Error Code

3) A message

**The General Syntax for coding the exception section**

```
DECLARE

    Declaration section

BEGIN

    Exception section

 EXCEPTION

 WHEN ex_name1 THEN
       -Error handling statements
WHEN ex_name2 THEN
       -Error handling statements
WHEN Others THEN
        Error handling statements

END;
```

# Types of Exception

There are 3 types of Exceptions.

a) Named System Exceptions

b) Numbered System Exceptions

c) **User-defined Exceptions**

## a) Named System Exceptions

- They are pre-defined and given a name in Oracle which are known as Named System Exceptions.

  **For example:** NO_DATA_FOUND and ZERO_DIVIDE are called Named System exceptions.

## b) Numbered System Exceptions

- Those system exception for which oracle does not provide a name is known as unamed system exception.

- There are two ways to handle unnamed sysyem exceptions:

  - By using the WHEN OTHERS exception handler, or
  - By associating the exception code to a name and using it as a named exception.

- We can assign a name to unnamed system exceptions using a **Pragma** called **EXCEPTION_INIT.**

- **EXCEPTION_INIT** will associate a predefined Oracle error number to a programmer_defined exception name.
- The general syntax to declare unnamed system exception using EXCEPTION_INIT is:

```
DECLARE
   exception_name EXCEPTION;
   PRAGMA
   EXCEPTION_INIT (exception_name, Err_code);
 BEGIN
   Execution section
EXCEPTION
  WHEN exception_name THEN
   handle the exception
END;
```

## c) User-defined Exceptions

- Apart from system exceptions we can explicitly define exceptions based on business rules. These are known as user-defined exceptions.

- **Steps to be followed to use user-defined exceptions:**

They should be explicitly declared in the declaration section.

They should be explicitly raised in the Execution Section.

They should be handled by referencing the user-defined exception name in the exception section

# In Built Error Handling Function

- **RAISE_APPLICATION_ERROR ( )**

  RAISE_APPLICATION_ERROR is a built-in procedure in oracle which is used to display the user-defined error messages along with the error number whose range is in between -20000 and -20999.

  RAISE_APPLICATION_ERROR raises an exception but does not handle it.

- The General Syntax to use this procedure is:

  RAISE_APPLICATION_ERROR (error_number, error_message);

  - The Error number must be between -20000 and -20999
  - The Error_message is the message you want to display when the error occurs.

# 2.5 Cursors

Text Book Name : Ivan Bayross , Ref. plsql-tutorial.com

- 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.

- **Types of cursors :-**

  - **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.

# 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.

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

| Attributes | Return Value | Example |
|---|---|---|
| **%ISOPEN** | If cursor is open, returns true ; else, returns false. | SQL%ISOPEN |
| **%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 |
| **%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 |
| **%ROWCOUNT** | Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE, SELECT | SQL%ROWCOUNT |

# Explicit Cursor

- 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.

# How To Create Cursor

CURSOR cursor_name IS 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.

1. DECLARE the cursor in the declaration section.
2. OPEN the cursor in the Execution Section.
3. FETCH the data from cursor into PL/SQL variables or records in the Execution Section.
4. CLOSE the cursor in the Execution Section before you end the PL/SQL Block.

1) Declaring a Cursor in the Declaration Section:

*DECLARE*

*CURSOR emp_cur IS SELECT * FROM emp_tbl WHERE salary > 5000 ;*

2) Accessing the records in the cursor:

Open the cursor.

Fetch the records in the cursor one at a time.

Close the cursor.

## How to Open Cursor

```
OPEN cursor_name;
```

## How to Fetch Cursor

```
FETCH cursor_name INTO record_name;
                  OR
FETCH cursor_name INTO variable_list;
```

## How to Close Cursor

```
CLOSE cursor_name;
```

## How to Use Explicit Cursor

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

## Explicit Cursor Attributes

- Oracle provides some attributes known as Explicit Cursor Attributes to control the data processing while using cursors.
  - Cursor_name%FOUND
  - Cursor_name%NOTFOUND
  - Cursor_name%ROWCOUNT
  - Cursor_name%ISOPEN

# Cursor With Simple Loop

```
DECLARE
  CURSOR emp_cur IS 3> SELECT first_name, last_name, salary FROM
 emp_tbl;
  emp_rec emp_cur%rowtype;
BEGIN
  IF NOT sales_cur%ISOPEN THEN
        OPEN sales_cur;
   END IF;
LOOP
FETCH emp_cur INTO emp_rec;
    EXIT WHEN emp_cur%NOTFOUND;
       dbms_output.put_line(emp_cur.first_name || ' ' ||emp_cur.last_name || ' '
 ||emp_cur.salary);
END LOOP;
END;
```

# Cursor With For Loop

```
FOR record_name IN cusror_name
LOOP
    process the row...
END LOOP;
```

# 2.6 PROCEDURES,FUNCTION AND PACKAGES

Text Book Name : Ivan Bayross , Ref. plsql-tutorial.com

## Stored Procedures

- A stored procedure or in simple a proc is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages. A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block. A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

## Advantages Of Function And Procedure

- Security
- Faster execution
- Sharing of code
- Productivity
- Integrity

# How To Create a Procedure

```
CREATE [OR REPLACE] PROCEDURE proc_name
(argument [in,out,in out] data type,..)
 IS
    Declaration section

BEGIN

    Execution section

EXCEPTION
    Exception section

END;
```

# How To Execute a Procedure

There are two ways to execute a procedure.

## 1) From the SQL prompt.

Syn :- EXECUTE [or EXEC] procedure_name(parameter);

ex :- exec debitacc('A01',5555);

## 2) Within another procedure–simply use the procedure name.

Syn :- procedure_name(parameters);

ex :- debitacc('A01',4444);

# PL/SQL Functions

- A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

# How To Create a Functions

```
CREATE [OR REPLACE] FUNCTION function_name
    ( argument in datatype,..)
    RETURN return_datatype;
    IS
    Declaration_section
    BEGIN
        Execution_section
        Return return_variable;
   EXCEPTION
        exception section
        Return return_variable;
   END;
```

A function can be executed in the following ways.

**1) Since a function returns a value we can assign it to a variable.**

```
employee_name := employer_details_func;
```

**2) As a part of a SELECT statement**

```
SELECT employer_details_func FROM dual;
```

**3) In a PL/SQL Statements like**

```
dbms_output.put_line(employer_details_func);
```

## Parameters In Procedure And Functions

- **IN type parameter:** These types of parameters are used to send values to stored procedures.
- **OUT type parameter:** These types of parameters are used to get values from stored procedures. This is similar to a return type in functions.
- **IN OUT parameter:** These types of parameters are used to send values and get values from stored procedures.

## How To Drop Function or Procedure

Drop function functionname ;

Drop procedure procedurename ;

# Package

- A package is a schema object that groups logically related PL/SQL types, items, and subprograms. Packages usually have two parts, a specification and a body, although sometimes the body is unnecessary. The specification (spec for short) is the interface to your applications; it declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The body fully defines cursors and subprograms, and so implements the spec.

# Advantage Of Package

- Modularity
- Easier Application Design
- Information Hiding
- Added Functionality
- Better Performance

# Structure of package

## 1)Package Specification

- **SYNTAX:-**

```
Create or replace package packagename
Is
    --package specification
End packagename;
```

## 2)Package body

- **SYNTAX:-**

```
Create or replace package body packagename
Is
    --package body
End packagename;
```

## Reference A Package Subprogram Using

Packagename.object;

## Destroying package using

Drop package packagename;