PL/SQL Concept

PL/SQL (Procedural Language/Structured Query Language) is a procedural programming language that is designed specifically for use with the Oracle database management system. It is a powerful language that provides developers with the ability to create complex and highly optimized database applications.

PL/SQL allows developers to write stored procedures, functions, triggers, and packages that can be stored in the database and executed on demand. These stored programs can be written in a high-level programming language, which allows for the development of complex business logic that can be executed within the database.

Some of the key features of PL/SQL include:

1. **Stored procedures:** PL/SQL allows developers to write stored procedures that can be called from within SQL statements or other PL/SQL blocks.
2. **Data types:** PL/SQL provides a wide range of data types, including user-defined types, that can be used to store and manipulate data within the database.
3. **Exception handling:** PL/SQL provides developers with a powerful exception-handling mechanism that allows for the trapping and handling of errors that occur during program execution.
4. **Cursors:** PL/SQL provides the ability to define cursors, which are used to retrieve data from the database and process it in a row-by-row fashion.
5. **Triggers:** Triggers are database objects that are automatically executed in response to certain events, such as changes to data in a table. A trigger consists of a PL/SQL block that is executed when the trigger event occurs. Triggers can be used to enforce business rules, audit changes to data, and perform complex data validation.
6. **Functions:** Functions, on the other hand, are PL/SQL subprograms that return a single value. They can be called from within SQL statements, other PL/SQL blocks, or even other functions. Functions are useful for encapsulating complex calculations and data manipulation operations, and for reusing code in multiple places within an application.

DECLARE -- declaration section (optional)

The DECLARE section of the block is optional and is used to declare variables, constants, cursors, and other program objects that will be used within the executable section of the block. The syntax for declaring variables is as follows:

DECLARE variable_name data_type := value; BEGIN -- executable section END;

BEGIN -- executable section (required)

-- contains one or more SQL or PL/SQL statements -- such as SELECT, UPDATE, INSERT, DELETE, IF, LOOP, etc. EXCEPTION

The BEGIN keyword signals the start of the executable section of the block. This section contains the main body of the PL/SQL code, which consists of one or more SQL or PL/SQL statements that perform some processing, manipulation or retrieval of data. This section is mandatory, and at least one statement is required.

The EXCEPTION section is also optional, and it is used to handle exceptions that may occur in the executable section of the block. This section contains one or more exception handlers that define how to handle a particular exception when it occurs.

The END keyword signals the end of the block. The semi-colon at the end of the END keyword is also mandatory.

Example ,

```
Declare
a int;
b int;
c int;
begin
a:=&a;
b:=&b;
c:=a+b;
dbms_output.put_line('Addition = ' || c);
end; /
```

Create, alter, drop, execute stored procedure

1. **Creating a stored procedure:**

CREATE PROCEDURE procedure_name [parameter1 datatype, parameter2 datatype, ...]
AS
BEGIN -- SQL statements within the stored procedure END;

Replace **procedure_name** with the desired name for your stored procedure. You can optionally define parameters by specifying **parameter1 datatype, parameter2 datatype, ...** within the parentheses after the procedure name.

2. **Altering a stored procedure:**

ALTER PROCEDURE procedure_name [parameter1 datatype, parameter2 datatype, ...]
AS
BEGIN -- Updated SQL statements within the stored procedure END;

Use the **ALTER PROCEDURE** statement followed by the procedure name to modify the existing stored procedure. You can redefine parameters or update the SQL statements within the procedure.

3. **Dropping a stored procedure:**

DROP PROCEDURE procedure_name;

The **DROP PROCEDURE** statement followed by the procedure name allows you to remove a stored procedure from the database.

4. **Executing a stored procedure:**

EXEC procedure_name [parameter_values];

To execute a stored procedure, use the **EXEC** statement followed by the procedure name. If the stored procedure has parameters, provide the corresponding parameter values within the square brackets.

Trigger

A trigger in a database is a stored program that is automatically executed in response to certain events or actions. These events, known as triggering events, can include data manipulation operations (such as INSERT, UPDATE, DELETE) or database-related events (such as system startup or shutdown). Triggers are defined on database tables and are associated with specific events that occur on those tables.

The main purpose of a trigger is to enable the database to automatically take action or enforce specific business rules whenever a predefined event occurs. Triggers are commonly used to implement complex integrity constraints, perform data validation, enforce business rules, maintain audit trails, and synchronize data between tables. They can be used to automate repetitive tasks or implement custom business logic within the database.

Triggers can be classified based on the timing of their execution:

1. Before triggers (or pre-triggers): These triggers are fired before the triggering event occurs. They can be used to modify data values or cancel the event based on certain conditions.
2. After triggers (or post-triggers): These triggers are fired after the triggering event occurs. They are typically used for auditing, logging, or performing additional actions that are not directly related to the event itself.

Triggers can also be classified based on the level at which they are defined:

1. Row-level triggers: These triggers operate on each affected row individually. They can access the old and new values of the row being modified.
2. Statement-level triggers: These triggers operate on a set of rows affected by a single SQL statement. They cannot access the individual row values and are typically used for performing actions that affect multiple rows.

Create, Disable Trigger

Create a trigger:

```
CREATE OR REPLACE TRIGGER trigger_name
 BEFORE
INSERT ON table_name
 FOR EACH
ROW
BEGIN -- Trigger logic
END; /
```

Replace **trigger_name** with the desired name for your trigger and **table_name** with the name of the table on which the trigger should be applied. The **BEFORE INSERT** clause specifies that the trigger will fire before an insert operation on the specified table. Adjust the trigger logic inside the **BEGIN** and **END** blocks as per your requirements.

Disable a trigger:

```
ALTER TRIGGER trigger_name DISABLE;
```