**Assignment No:3**

**Title:** Write a procedure and function that takes in a zip code, city, and state and inserts the values into the zip code table. It should check to see if the zip code is already in the database. If it is, an exception should be raised, and an error message should be displayed. Write an anonymous block that uses the procedure and inserts your zip code.

**Problem Statement:** Create a procedure and a function that accept a zip code, city, and state as input and add these values to the zip\_code\_table. The function should verify if the zip code already exists in the database. If it does, an exception must be raised, displaying an appropriate error message. Additionally, write an anonymous block to demonstrate the use of the procedure by inserting a sample zip code.

**Objective:** • To acquire knowledge of database query languages.

**Outcome:** Implement different SQL and PLSQL operations using suitable databases.

**Tools Required:** Ubuntu OS, Mysql.

**Theory:**

**1. Introduction to PL/SQL**

PL/SQL (Procedural Language/Structured Query Language) is an extension of SQL developed by Oracle Corporation. It combines the power of SQL for data manipulation with procedural constructs such as loops, conditions, and exception handling, enabling the development of robust and scalable database applications.

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

* PL/SQL is tightly integrated with SQL.
* It offers extensive error checking.
* It offers numerous data types.
* It offers a variety of programming structures.
* It supports structured programming through functions and procedures.
* It supports object-oriented programming.
* It supports the development of web applications and server pages.

**1.1 Advantages:**

* SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL block. In Dynamic SQL, SQL allows embedding DDL statements in PL/SQL blocks.
* PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.
* PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.
* PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
* Applications written in PL/SQL are fully portable.
* PL/SQL provides high security level.
* PL/SQL provides access to predefined SQL packages.
* PL/SQL provides support for developing Web Applications and Server Pages.

**1.23 Syntax:**

**Declarations:** This section starts with the keyword **DECLARE**. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.

**Executable Commands:** This section is enclosed between the keywords BEGIN and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.

**Exception Handling:**This section starts with the keyword **EXCEPTION**. This optional section contains **exception(s)** that handle errors in the program.

|  |  |
| --- | --- |
| **Syntax**  DECLARE  <declarations section>  BEGIN  <executable command(s)>  EXCEPTION  <exception handling>  END; | **Example**  DECLARE  message varchar2(20):= 'Hello, Golu!’;  BEGIN  dbms\_output.put\_line(message);  END; / |

**2. PL/SQL Basic:**

**2.1 Local and Global variables**

**Local variables −** Variables declared in an inner block and not accessible to outer blocks.

**Global variables −** Variables declared in the outermost block or a package.

|  |
| --- |
| DECLARE  -- Global variables  num1 number := 95;  num2 number := 85;  BEGIN  dbms\_output.put\_line('Outer Variable num1: ' || num1);  dbms\_output.put\_line('Outer Variable num2: ' || num2);  DECLARE  -- Local variables  num1 number := 195;  num2 number := 185;  BEGIN  dbms\_output.put\_line('Inner Variable num1: ' || num1);  dbms\_output.put\_line('Inner Variable num2: ' || num2);  END;  END;  / |

**2.2 Assigning SQL Query Results to PL/SQL Variables**

You can use the SELECT INTO statement of SQL to assign values to PL/SQL variables. For each item in the SELECT list, there must be a corresponding, type-compatible variable in the INTO list.

|  |
| --- |
| DECLARE  c\_id customers.id%type := 1;  c\_name customers.name%type;  c\_addr customers.address%type;  c\_sal customers.salary%type;  BEGIN  SELECT name, address, salary INTO c\_name, c\_addr, c\_sal  FROM customers  WHERE id = c\_id;  dbms\_output.put\_line('Customer ' ||c\_name || ' from ' || c\_addr || ' earns ' || c\_sal);  END;  / |

**2.3 Loop**

Basic loop structure encloses sequence of statements in between the LOOP and END LOOP statements. With each iteration, the sequence of statements is executed and then control resumes at the top of the loop.

|  |  |
| --- | --- |
| **Syntax :**  LOOP  Sequence of statements;  END LOOP; | **Example**  DECLARE  x number := 10;  BEGIN  LOOP  dbms\_output.put\_line(x);  x := x + 10;  IF x > 50 THEN  exit;  END IF;  END LOOP;  -- after exit, control resumes here  dbms\_output.put\_line('After Exit x is: ' || x);  END;  / |

**2.4 IF statement**

The **IF statement** associates a condition with a sequence of statements enclosed by the keywords THEN and END IF.

|  |
| --- |
| IF (a <= 20) THEN  c:= c+1;  END IF; |

**2.5 WHILE LOOP**

**A WHILE LOOP** statement in PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.

|  |  |
| --- | --- |
| **Syntax:**  WHILE condition LOOP  sequence\_of\_statements  END LOOP; | **Example:**  DECLARE  a number(2) := 10;  BEGIN  WHILE a < 20 LOOP  dbms\_output.put\_line('value of a: ' || a);  a := a + 1;  END LOOP;  END;  / |

**2.6 FOR LOOP**

**A FOR LOOP** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

|  |
| --- |
| DECLARE  a number(2);  BEGIN  FOR a in 10 .. 20 LOOP  dbms\_output.put\_line('value of a: ' || a);  END LOOP;  END;  / |

**3. PL/SQL Procedure**

A **PL/SQL Procedure** is a named block of PL/SQL code that performs a specific task. It is stored in the database and can be reused multiple times. Procedures allow for modular programming, improve code reusability, and simplify maintenance.

**3.1 Syntax**

|  |
| --- |
| CREATE [OR REPLACE] PROCEDURE procedure\_name[(parameter\_name [IN | OUT | IN OUT] type [, ...])]  {IS | AS}  BEGIN  < procedure\_body >  END procedure\_name; |

**Key Components of a Procedure**

1. **Header**: Contains the procedure name and parameter list.
   * **IN**: Passes a value to the procedure.
   * **OUT**: Returns a value to the calling program.
   * **IN OUT**: Passes a value to the procedure and returns an updated value.
2. **IS/AS**: Indicates the start of the procedure body.
3. **Declaration Section**: Declares variables, cursors, etc.
4. **Executable Section**: Contains SQL and PL/SQL statements.
5. **Exception Section**: Handles runtime errors.

**3.2 Example**

|  |
| --- |
| DECLARE  a number;  b number;  c number;  PROCEDURE findMin(x IN number, y IN number, z OUT number) IS  BEGIN  IF x < y THEN  z:= x;  ELSE  z:= y;  END IF;  END;  BEGIN  a:= 23;  b:= 45;  findMin(a, b, c);  dbms\_output.put\_line(' Minimum of (23, 45) : ' || c);  END;  / |

**4. PL/SQL Function**

A **PL/SQL Function** is a named block of code that performs a specific task and returns a single value to the calling program. Functions are commonly used to encapsulate logic that computes and returns a value, making them reusable and efficient.

**4.1 Syntax**

|  |
| --- |
| CREATE [OR REPLACE] FUNCTION function\_name[(parameter\_name [IN | OUT | IN OUT] type [, ...])]  RETURN return\_datatype  {IS | AS}  BEGIN  < function\_body >  END [function\_name]; |

* *function-name* specifies the name of the function.
* [OR REPLACE] option allows the modification of an existing function.
* The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
* The function must contain a **return** statement.
* The *RETURN* clause specifies the 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.

**4.3 Example**

|  |
| --- |
| DECLARE  a number;  b number;  c number;  FUNCTION findMax(x IN number, y IN number) RETURN number  IS  z number;  BEGIN  IF x > y THEN  z:= x;  ELSE  Z:= y;  END IF;  RETURN z;  END;  BEGIN  a:= 23;  b:= 45;  c := findMax(a, b);  dbms\_output.put\_line(' Maximum of (23,45): ' || c);  END;  / |

**Conclusion:**

We have successfully implemented procedure and function that takes in a zip code, city, and state and inserts the values into the zip code table..