ADBMS

UNIT -4

PL/SQL (Procedural Language/Structured Query Language)

Introduction:

- PL/SQL is a procedural extension of SQL used for writing programmatic logic within Oracle databases.
- It combines SQL for data manipulation with procedural constructs for control flow and logic execution.
- PL/SQL allows users to create functions, procedures, triggers, and packages to manage database operations.

Advantages:

1. Integration with SQL:
    - PL/SQL seamlessly integrates with SQL, enabling efficient database operations and manipulation of data.
2. Procedural Constructs:
    - Offers procedural constructs such as loops, conditionals, and exception handling for implementing complex logic and business rules.
3. Modularity:
    - Supports modular programming through the use of procedures, functions, and packages, promoting code reusability and maintainability.
4. Performance:
    - Optimized execution within the database engine leads to improved performance compared to executing SQL statements individually.
5. Security:
    - Provides fine-grained access control and security features, ensuring secure access to database objects and sensitive data.

Blocks:

- A PL/SQL block is a unit of code containing declarations, executable statements, and exception handling.
- It begins with the `DECLARE` keyword for declaring variables and ends with the `END` keyword.

Character Set:

- PL/SQL supports the character set defined by the underlying database.
- Unicode character sets like UTF-8 are commonly used for internationalization and multilingual support.

Literals:

- Literals are constant values directly represented in PL/SQL code.
- Examples include string literals ('Hello'), numeric literals (123), and date literals (DATE '2024-05-23').

Data Types:

- PL/SQL supports various data types for representing different kinds of data, including:
    1. Scalar Data Types: INTEGER, VARCHAR2, NUMBER, DATE, etc.
    2. Composite Data Types: RECORD, TABLE, etc.
    3. Reference Data Types: CURSOR, REF CURSOR, etc.

Variables:

- Variables are named storage locations used to hold data values within PL/SQL programs.
- They must be declared with a specific data type before use and can be assigned values using the assignment operator :=.

Example (Acedemia Context):

```
DECLARE
 student_name VARCHAR2(100);
 student_id NUMBER;
 course_code VARCHAR2(10);
BEGIN
 -- Prompt user for student details
 student_name := 'Samridhi Jha';
 student_id := 20242001; -- Assuming student ID format: year (last 2 digits) + roll number
 course_code := 'CSE101';

 -- Enroll student in the course
 INSERT INTO student_courses (student_id, course_code)
 VALUES (student_id, course_code);

 -- Display success message
 DBMS_OUTPUT.PUT_LINE('Student ' || student_name || ' enrolled in course ' || course_code);
```

```
EXCEPTION
 WHEN OTHERS THEN
 -- Handle errors
 DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

This PL/SQL block enrolls a student named Samridhi Jha with a student ID of 20242001 in the 'CSE101' course at Jawaharlal Nehru University, Delhi. It inserts a record into the `student_courses` table and outputs a success message. Exception handling is included to catch and handle any errors that may occur during execution.

Constants:

Definition:

- Constants are fixed values that remain unchanged during the execution of a program.
- They are declared using the `CONSTANT` keyword and assigned a value during declaration.
- Constants are useful for storing values that are not expected to change throughout the program's execution.

Example:

```
DECLARE
 PI CONSTANT NUMBER := 3.14159;
BEGIN
 DBMS_OUTPUT.PUT_LINE('Value of PI: ' || PI);
END;
/
```

In this example, `PI` is declared as a constant with the value of π (approximately 3.14159). The value of `PI` remains constant throughout the execution of the program.

Attributes:

Definition:

- Attributes provide information about various program constructs such as variables, exceptions, and cursors.
- They are accessed using predefined syntax and provide metadata about the associated program construct.

Example:

```
DECLARE
 emp_name VARCHAR2(100);
BEGIN
 emp_name := 'Samridhi Jha';

 -- Using attributes to get the length of a string
 DBMS_OUTPUT.PUT_LINE('Length of emp_name: ' || LENGTH(emp_name));
END;
/
```

In this example, the `LENGTH` function is used to retrieve the length of the `emp_name` string variable. `LENGTH` is an attribute that returns the length of the specified string.

Control Structures:

Conditional Control:

Definition:

- Conditional control structures in PL/SQL allow for executing different blocks of code based on specified conditions.
- Common conditional statements include `IF-THEN`, `IF-THEN-ELSE`, and `CASE` statements.

Example:

```
DECLARE
 num INTEGER := 10;
BEGIN
 IF num > 0 THEN
 DBMS_OUTPUT.PUT_LINE('Positive');
 ELSIF num = 0 THEN
 DBMS_OUTPUT.PUT_LINE('Zero');
 ELSE
 DBMS_OUTPUT.PUT_LINE('Negative');
 END IF;
END;
/
```

In this example, the program checks the value of `num` and prints whether it is positive, negative, or zero based on the condition.

Iterative Control:

Definition:

- Iterative control structures in PL/SQL allow for executing a block of code repeatedly until a specified condition is met.
- Common iterative statements include `LOOP`, `WHILE`, and `FOR` loops.

Example:

```
DECLARE
 counter INTEGER := 1;
BEGIN
 WHILE counter <= 5 LOOP
 DBMS_OUTPUT.PUT_LINE('Iteration: ' || counter);
 counter := counter + 1;
 END LOOP;
END;
/
```

In this example, a `WHILE` loop is used to print the value of `counter` from 1 to 5 in each iteration.

Sequential Control:

Definition:

- Sequential control structures in PL/SQL define the order of execution of statements within a block of code.
- Statements are executed sequentially from top to bottom unless control flow statements alter the default execution order.

Example:

```
DECLARE
 num1 INTEGER := 10;
 num2 INTEGER := 20;
 result INTEGER;
BEGIN
 result := num1 + num2;
 DBMS_OUTPUT.PUT_LINE('Result: ' || result);
END;
/
```

In this example, `num1` and `num2` are added together, and the result is stored in the `result` variable. The result is then printed using `DBMS_OUTPUT.PUT_LINE`.

Cursors:

Definition:

- Cursors in PL/SQL are named control structures used to retrieve and process multiple rows of data returned by a SQL query.
- Cursors provide a mechanism for iterative processing of query results row by row.

Example:

```
DECLARE
 emp_name VARCHAR2(100);
 CURSOR emp_cursor IS
 SELECT employee_name FROM employees;
BEGIN
 OPEN emp_cursor;
 LOOP
 FETCH emp_cursor INTO emp_name;
 EXIT WHEN emp_cursor%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_name);
 END LOOP;
 CLOSE emp_cursor;
END;
/
```

In this example, a cursor named `emp_cursor` is declared to fetch the `employee_name` from the `employees` table. The cursor is opened, data is fetched row by row into the `emp_name` variable, and each employee name is printed using `DBMS_OUTPUT.PUT_LINE`.

Exception Handling:

Definition:

- Exception handling in PL/SQL provides a mechanism to gracefully handle errors or exceptional conditions that may occur during program execution.
- It allows developers to detect, handle, and recover from errors, ensuring the robustness and reliability of the application.

Components of Exception Handling:

1. Exception Declaration: Define custom exceptions or use predefined exceptions provided by PL/SQL.
2. Exception Handling Block: Surround code that may raise exceptions with a `BEGIN...EXCEPTION...END` block.

3. Exception Handlers: Specify actions to be taken when specific exceptions occur using `EXCEPTION WHEN` clauses.
4. Logging and Error Reporting: Log error messages or perform error reporting using built-in functions like `DBMS_OUTPUT.PUT_LINE` or logging frameworks.

Example:

```
DECLARE
 num1 INTEGER := 10;
 num2 INTEGER := 0;
 result INTEGER;
BEGIN
 result := num1 / num2; -- Division by zero
 DBMS_OUTPUT.PUT_LINE('Result: ' || result);
EXCEPTION
 WHEN ZERO_DIVIDE THEN
 DBMS_OUTPUT.PUT_LINE('Error: Division by zero');
END;
/
```

In this example, an exception is raised when attempting to divide `num1` by `num2`, resulting in a division by zero error. The exception is caught using a `WHEN ZERO_DIVIDE` clause, and an error message is printed.

Triggers:

Definition:

- Triggers in PL/SQL are named blocks of code that automatically execute in response to specified events occurring in the database.
- They can be triggered by DML (Data Manipulation Language) events such as INSERT, UPDATE, DELETE, or DDL (Data Definition Language) events such as CREATE, ALTER, DROP.

Types of Triggers:

1. Row-Level Triggers: Fired once for each row affected by the triggering event.
2. Statement-Level Triggers: Fired once for each triggering event, regardless of the number of rows affected.

Example:

```
CREATE OR REPLACE TRIGGER student_insert_trigger
BEFORE INSERT ON students
```

```
FOR EACH ROW
BEGIN
 -- Perform actions before inserting into the students table
 DBMS_OUTPUT.PUT_LINE('Trigger: New student inserted');
END;
/
```

This trigger fires before each row is inserted into the `students` table. It prints a message indicating that a new student has been inserted.

Procedures:

Definition:

- Procedures in PL/SQL are named blocks of code that perform a specific task or set of tasks.
- They encapsulate a series of SQL and procedural statements, allowing for code reusability and modularity.

Components of Procedures:

1. Procedure Declaration: Define the procedure name, parameters, and return type (if any).
2. Procedure Body: Implement the logic and statements to be executed.
3. Procedure Invocation: Call the procedure from other PL/SQL blocks or SQL statements.

Example:

```
CREATE OR REPLACE PROCEDURE calculate_salary (
 emp_id IN NUMBER,
 hours_worked IN NUMBER
)
IS
 hourly_rate NUMBER := 25; -- Assuming hourly rate
 total_salary NUMBER;
BEGIN
 total_salary := hours_worked * hourly_rate;
 DBMS_OUTPUT.PUT_LINE('Employee ' || emp_id || ' earned $' || total_salary);
END calculate_salary;
/
```

This procedure calculates the total salary earned by an employee based on their `emp_id` and the `hours_worked`. It multiplies the `hours_worked` by the `hourly_rate` and prints the result.

Packages:

Definition:

- Packages in PL/SQL are schema objects that group related procedures, functions, variables, and cursors together.
- They provide a way to organize and encapsulate code for better modularity, encapsulation, and performance.

Components of Packages:

1. Package Specification: Declares the public elements (procedures, functions, types) that can be accessed by other PL/SQL blocks.
2. Package Body: Implements the logic and private elements (variables, cursors) used internally within the package.

Example:

```
CREATE OR REPLACE PACKAGE student_package AS
 PROCEDURE enroll_student (student_id IN NUMBER, course_code IN VARCHAR2);
END student_package;
/

CREATE OR REPLACE PACKAGE BODY student_package AS
 PROCEDURE enroll_student (student_id IN NUMBER, course_code IN VARCHAR2)
 IS
 BEGIN
 INSERT INTO student_courses (student_id, course_code)
 VALUES (student_id, course_code);
 DBMS_OUTPUT.PUT_LINE('Student enrolled in course: ' || course_code);
 END enroll_student;
END student_package;
/
```

This package defines a procedure `enroll_student` in the package specification, which inserts a record into the `student_courses` table when called. The package body implements the logic for the `enroll_student` procedure