CERTIFIED QUIZ

Introduction to PL/SQL

PL/SQL (Procedural Language/Structured Query Language) is Oracle's procedural extension for SQL and the Oracle relational database. It is designed to integrate seamlessly with SQL and offers procedural programming capabilities such as loops, conditions, and exception handling. PL/SQL is used to create stored procedures, functions, packages, triggers, and other program units that reside in the database.

PL/SQL Block Structure

A PL/SQL program consists of a block of code. The block is divided into three sections:

1. Declaration: Used to declare variables, cursors, and types.
2. Execution: Contains the SQL and PL/SQL statements that are executed.
3. Exception Handling: Handles errors that occur during execution.

Example:
sql
DECLARE
v_emp_name VARCHAR2(50);
BEGIN
SELECT first_name INTO v_emp_name FROM employees WHERE employee_id = 100;
DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_emp_name);
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Employee not found.');
END;

Variables and Data Types

PL/SQL supports a variety of data types, including numbers, characters, strings, dates, and booleans. Variables can be declared in the declaration section of a block and used throughout the program.

Example:
sql
DECLARE
v_salary NUMBER(7,2);
v_employee_name VARCHAR2(50);
BEGIN
v_salary := 50000;
v_employee_name := 'John Doe';
END;

Control Structures

PL/SQL provides control structures like conditional statements and loops to control the flow of execution:

- IF-THEN-ELSE: Executes statements based on conditions.
- LOOPS: Repeats statements until a condition is met.
- CASE: Similar to IF-THEN-ELSE, but allows for multiple conditions.

Example:
sql

```
BEGIN
IF v_salary > 10000 THEN
DBMS_OUTPUT.PUT_LINE('High salary');
ELSE
DBMS_OUTPUT.PUT_LINE('Low salary');
END IF;

FOR i IN 1..10 LOOP
DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
END;
```

Cursors

Cursors are used in PL/SQL to process query results row by row. There are two types of cursors:

- Implicit Cursor: Automatically created by Oracle for SQL statements that return a single row.
- Explicit Cursor: Must be explicitly declared for queries that return multiple rows.

Example of an explicit cursor:

```
sql
DECLARE
CURSOR emp_cursor IS SELECT employee_id, first_name FROM employees;
v_emp_id employees.employee_id%TYPE;
v_first_name employees.first_name%TYPE;
BEGIN
OPEN emp_cursor;
LOOP
FETCH emp_cursor INTO v_emp_id, v_first_name;
EXIT WHEN emp_cursor%NOTFOUND;
DBMS_OUTPUT.PUT_LINE(v_emp_id || ' ' || v_first_name);
END LOOP;
CLOSE emp_cursor;
END;
```

Stored Procedures and Functions

Stored procedures and functions are reusable program units that encapsulate business logic. A procedure performs an action, while a function returns a value.

- Procedure Example:

```
sql
CREATE OR REPLACE PROCEDURE update_salary(p_emp_id IN NUMBER, p_new_salary IN NUMBER) IS
BEGIN
UPDATE employees SET salary = p_new_salary WHERE employee_id = p_emp_id;
END update_salary;
/
```

- Function Example:

```
sql
CREATE OR REPLACE FUNCTION get_employee_name(p_emp_id IN NUMBER) RETURN VARCHAR2 IS
v_emp_name VARCHAR2(100);
BEGIN
SELECT first_name INTO v_emp_name FROM employees WHERE employee_id = p_emp_id;
RETURN v_emp_name;
```

```
END get_employee_name;
/
```

Triggers

Triggers are stored PL/SQL blocks that are automatically executed in response to specific database events such as INSERT, UPDATE, or DELETE. Triggers are useful for maintaining data integrity or enforcing business rules.

Example:
sql
```
CREATE OR REPLACE TRIGGER before_employee_insert
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
:NEW.hire_date := SYSDATE;
END;
/
```

Packages

Packages are collections of related procedures, functions, and variables grouped together as a single unit. They provide modularity, easier maintenance, and performance improvements.

Example:
sql
```
CREATE OR REPLACE PACKAGE employee_pkg IS
PROCEDURE add_employee(p_name IN VARCHAR2, p_salary IN NUMBER);
FUNCTION get_employee_count RETURN NUMBER;
END employee_pkg;
/

CREATE OR REPLACE PACKAGE BODY employee_pkg IS
PROCEDURE add_employee(p_name IN VARCHAR2, p_salary IN NUMBER) IS
BEGIN
INSERT INTO employees (first_name, salary) VALUES (p_name, p_salary);
END add_employee;

FUNCTION get_employee_count RETURN NUMBER IS
v_count NUMBER;
BEGIN
SELECT COUNT(*) INTO v_count FROM employees;
RETURN v_count;
END get_employee_count;
END employee_pkg;
/
```

Exception Handling

PL/SQL provides a robust exception-handling mechanism to handle runtime errors. You can define custom error-handling routines or use predefined exceptions.

Example:
sql
```
BEGIN
SELECT salary INTO v_salary FROM employees WHERE employee_id = 100;
```

```
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Employee not found');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
END;
```

Bulk Collect

PL/SQL allows for bulk data processing using BULK COLLECT and FORALL. These statements are used to improve performance by reducing the context switch between PL/SQL and SQL.

Example:
sql
```
DECLARE
TYPE t_emp_names IS TABLE OF employees.first_name%TYPE;
v_emp_names t_emp_names;
BEGIN
SELECT first_name BULK COLLECT INTO v_emp_names FROM employees WHERE department_id = 10;
FOR i IN 1..v_emp_names.COUNT LOOP
DBMS_OUTPUT.PUT_LINE(v_emp_names(i));
END LOOP;
END;
```