PL/SQL Complete Master File – Full Syllabus + Advanced Codes

This document contains **every concept required in PL/SQL**, including: - PL/SQL Basics - Variables & Data Types - Control Structures - Loops - Cursors (Implicit & Explicit) - Procedures - Functions - Packages (optional advanced) - Exception Handling - Records - Triggers - Auditing Triggers - Views - Complex Practice Codes

---

1. Introduction to PL/SQL

PL/SQL (Procedural Language for SQL) is Oracle's extension to SQL. It supports: - Variables - Loops - Conditions - Reusable code (procedures & functions) - Cursors - Exception handling

Structure of a PL/SQL block:

```
DECLARE
    -- Declarations
BEGIN
    -- Executable statements
EXCEPTION
    -- Error handling
END;
```

Always enable output:

```
SET SERVEROUTPUT ON;
```

---

2. Variables & Data Types

Declaring Variables

```
DECLARE
    a NUMBER := 10;
    b NUMBER := 20;
    name VARCHAR2(30);
```

Using %TYPE

```
DECLARE
    emp_salary employees.salary%TYPE;
```

```
BEGIN
    SELECT salary INTO emp_salary FROM employees WHERE employee_id = 100;
END;
```

Using %ROWTYPE

```
DECLARE
    emp_row employees%ROWTYPE;
BEGIN
    SELECT * INTO emp_row FROM employees WHERE
employee_id = 101;
END;
```

3. Conditional Statements

```
IF salary > 20000 THEN
    DBMS_OUTPUT.PUT_LINE('High');
ELSIF salary > 15000 THEN
    DBMS_OUTPUT.PUT_LINE('Medium');
ELSE
    DBMS_OUTPUT.PUT_LINE('Low');
END IF;
```

CASE

```
CASE department_id
    WHEN 10 THEN DBMS_OUTPUT.PUT_LINE('Admin');
    WHEN 20 THEN DBMS_OUTPUT.PUT_LINE('Sales');
    ELSE DBMS_OUTPUT.PUT_LINE('Other');
END CASE;
```

4. Loops

Basic LOOP

```
DECLARE
    i NUMBER := 1;
BEGIN
```

```
    LOOP
        DBMS_OUTPUT.PUT_LINE(i);
        i := i + 1;
        EXIT WHEN i > 5;
    END LOOP;
END;
```

FOR Loop

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

WHILE Loop

```
DECLARE
    x NUMBER := 5;
BEGIN
    WHILE x >= 1 LOOP
        DBMS_OUTPUT.PUT_LINE(x);
        x := x - 1;
    END LOOP;
END;
```

5. Cursors

Implicit Cursor (AUTO by Oracle)

Used in: - INSERT - UPDATE - DELETE - SELECT INTO

Example:

```
DECLARE
    emp_name employees.first_name%TYPE;
BEGIN
    SELECT first_name INTO emp_name FROM employees WHERE employee_id = 101;
END;
```

Explicit Cursor (MANUAL)

Syntax:

```
DECLARE
    CURSOR emp_cursor IS
        SELECT employee_id, first_name, salary
FROM employees;

    emp_row emp_cursor%ROWTYPE;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO emp_row;
        EXIT WHEN emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(emp_row.first_name ||
' - ' || emp_row.salary);
    END LOOP;
    CLOSE emp_cursor;
END;
```

6. Procedures

Procedures perform actions but **do NOT return values**.

Example:

```
CREATE OR REPLACE PROCEDURE Give_Raise(
    p_emp_id IN NUMBER,
    p_amount IN NUMBER
)
IS
BEGIN
    UPDATE employees
    SET salary = salary + p_amount
    WHERE employee_id = p_emp_id;

    DBMS_OUTPUT.PUT_LINE('Salary raised successfully.');
```

```
END;
/
```

Call it:

```
EXEC Give_Raise(101, 5000);
```

---

7. Functions

Functions **return** a single value.

Example:

```
CREATE OR REPLACE FUNCTION Annual_Salary(
    p_emp_id NUMBER
)
RETURN NUMBER
IS
    sal NUMBER;
BEGIN
    SELECT salary * 12 INTO sal
    FROM employees WHERE employee_id = p_emp_id;

    RETURN sal;
END;
/
```

Using the function:

```
SELECT Annual_Salary(101) FROM dual;
```

---

8. Exception Handling

Handled using: - NO_DATA_FOUND - TOO_MANY_ROWS - ZERO_DIVIDE - OTHERS

Example:

```
DECLARE
    emp_sal NUMBER;
BEGIN
    SELECT salary INTO emp_sal FROM employees
WHERE employee_id = 9999;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employee not
found.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An unexpected error
occurred.');
END;
```

9. Records

User-defined record:

```
DECLARE
    TYPE employee_record IS RECORD(
        name VARCHAR2(30),
        salary NUMBER,
        dept NUMBER);

    emp employee_record;
BEGIN
    SELECT first_name, salary, department_id
    INTO emp
    FROM employees
    WHERE employee_id = 100;

    DBMS_OUTPUT.PUT_LINE(emp.name || ' - ' || emp.salary);
END;
```

10. Views

Views are virtual tables created using SELECT.

```
CREATE OR REPLACE VIEW high_salary_view AS
SELECT employee_id, first_name, salary
FROM employees
WHERE salary > 20000;
```

Read-only view:

```
CREATE OR REPLACE VIEW emp_view_readonly AS
SELECT * FROM employees
WITH READ ONLY;
```

---

11. Triggers

Triggers execute **automatically** when DML occurs.

BEFORE INSERT Trigger

```
CREATE OR REPLACE TRIGGER trg_before_insert_student
BEFORE INSERT ON students
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('Inserting Student: ' || :NEW.name);
END;
/
```

BEFORE UPDATE Trigger

```
CREATE OR REPLACE TRIGGER trg_update_student
BEFORE UPDATE ON students
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('Old Name: ' || :OLD.name);
    DBMS_OUTPUT.PUT_LINE('New Name: ' || :NEW.name);
END;
/
```

BEFORE DELETE Trigger

```
CREATE OR REPLACE TRIGGER trg_delete_student
BEFORE DELETE ON students
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('Deleting: ' || :OLD.name);
END;
/
```

---

12. Auditing Trigger (Full Complex Code)

Audit requirements: - Save old/new values - Save user - Save date - Save operation type

Create Audit Table

```
CREATE TABLE emp_audit (
    new_name VARCHAR2(30),
    old_name VARCHAR2(30),
    user_name VARCHAR2(30),
    entry_date VARCHAR2(30),
    operation VARCHAR2(20)
);
```

Audit Trigger

```
CREATE OR REPLACE TRIGGER trg_emp_audit
BEFORE INSERT OR UPDATE OR DELETE ON employees
FOR EACH ROW
DECLARE
    v_user VARCHAR2(30);
    v_date VARCHAR2(30);
BEGIN
    SELECT USER, TO_CHAR(SYSDATE, 'DD-MON-YYYY
HH24:MI:SS')
    INTO v_user, v_date
    FROM dual;

    IF INSERTING THEN
        INSERT INTO emp_audit (new_name, old_name,
user_name, entry_date, operation)
        VALUES (:NEW.emp_name, NULL, v_user,
v_date, 'INSERT');

    ELSIF UPDATING THEN
        INSERT INTO emp_audit (new_name, old_name,
user_name, entry_date, operation)
        VALUES (:NEW.emp_name, :OLD.emp_name,
v_user, v_date, 'UPDATE');

    ELSIF DELETING THEN
        INSERT INTO emp_audit (new_name, old_name,
user_name, entry_date, operation)
        VALUES (NULL, :OLD.emp_name, v_user,
v_date, 'DELETE');
    END IF;
END;
/
```

13. MOST COMPLEX TRIGGER (MULTI-COLUMN AUDIT)

```
CREATE OR REPLACE TRIGGER full_employee_audit
BEFORE INSERT OR UPDATE OR DELETE ON employees
FOR EACH ROW
DECLARE
    v_user VARCHAR2(30);
    v_date VARCHAR2(30);
BEGIN
    SELECT USER, TO_CHAR(SYSDATE, 'DD-MON-YYYY
HH24:MI:SS')
    INTO v_user, v_date
    FROM dual;

    IF INSERTING THEN
        INSERT INTO emp_audit VALUES(
            :NEW.emp_name || ' / ' || :NEW.salary,
            NULL,
            v_user,
            v_date,
            'INSERT'
        );

    ELSIF UPDATING THEN
        INSERT INTO emp_audit VALUES(
            :NEW.emp_name || ' / ' || :NEW.salary,
            :OLD.emp_name || ' / ' || :OLD.salary,
            v_user,
            v_date,
            'UPDATE'
        );

    ELSIF DELETING THEN
        INSERT INTO emp_audit VALUES(
            NULL,
            :OLD.emp_name || ' / ' || :OLD.salary,
            v_user,
            v_date,
            'DELETE'
        );
    END IF;
END;
/
```

14. ADVANCED PRACTICE QUESTIONS

1. Write a trigger that prevents deleting a manager.

2. Write a function that returns number of employees in a dept.

3. Write a procedure that copies data from one table to another.

4. Create a cursor that prints employees with salary > average.

5. Create a master-detail auditing trigger for orders & order_items.

End of Full PL/SQL Master File