PRACTICAL FILE

DATA WAREHOUSING AND MULTI-DIMENSIONAL MODELING LAB

AIEU4123

Submitted By:

Harsh

M.Tech (AIML)

A501144824003

Submitted To:

Dr Sarita Gulia

INDEX

S.NO.	TITLE	DATE	SIGN
1.	Implementation of Basic SQL commands.	09-01-2025	
2.	Introduction to PL/SQL.	09-01-2025	
3.	Creates a simple procedure that displays the message 'Hello World!'	16-01-2025	
4.	Use variables and constants in a PL/SQL block	23-01-2025	
5.	Write a PL/SQL block to add two numbers.	30-01-2025	
6.	Create a procedure that computes the square of value of a passed value	06-02-2025	
7.	Create a PL/SQL block using simple IF-ELSE condition.	13-02-2025	
8.	Implementation of loops in PL/SQL (FOR, WHILE, DO WHILE LOOP).	20-02-2025	
9.	Program to calculates the factorial of a given number	27-02-2025	
10.	Implementation of Operators in PL/SQL.	06-03-2025	
11.	Implementation of Switch Case Statement in PL/SQL.	13-03-2025	
12.	Implementation of Strings in PL/SQL with Various String Function.	20-03-2025	
13.	Implementation of Exception handling in PL/SQL.	27-03-2025	
14.	Implementation of Functions in PL/SQL.	03-04-2025	
15.	Implementation of triggers in PL/SQL.	10-04-2025	
16.	Create a stored procedure to insert and update data.	17-04-2025	
17.	Basics of tableau.	24-04-2025	

Practical No : 1

Implementation of Basic SQL commands

1) Data Definition Language (DDL) Commands
   Used to define and modify database structures.

   a) CREATE
      Definition: Creates a new database object (table, view, index, etc.).
      Example :
      ```
      CREATE TABLE employees (
         employee_id INT PRIMARY KEY,
         first_name VARCHAR(50) NOT NULL,
         last_name VARCHAR(50) NOT NULL,
         email VARCHAR(100) UNIQUE,
         hire_date DATE,
         salary DECIMAL(10,2),
         department_id INT
      );
      ```

   b) ALTER
      Definition: Modifies an existing database structure (add, drop, or modify columns).
      Example :
      ```
      -- Add a new column
      ALTER TABLE employees ADD phone_number VARCHAR(15);

      -- Modify a column
      ALTER TABLE employees MODIFY COLUMN salary DECIMAL(12,2);

      -- Drop a column
      ALTER TABLE employees DROP COLUMN phone_number;
      ```

   c) DROP
      Definition: Deletes an existing database object (table, index, etc.).
      Example :
      ```
      DROP TABLE employees;
      ```

   d) TRUNCATE
      Definition: Removes all records from a table but keeps the structure.
      Example :
      ```
      TRUNCATE TABLE employees;
      ```

2) Data Manipulation Language (DML) Commands
   Used to manipulate data within database tables.

   a) INSERT
      Definition: Adds new records into a table.
      Example :
      ```
      -- Insert single record
      INSERT INTO employees (employee_id, first_name, last_name, email,
      hire_date, salary)
      VALUES (1, 'John', 'Doe', 'john.doe@example.com', '2020-01-15', 75000.00);

      -- Insert multiple records
      INSERT INTO employees VALUES
      ```

```
(2, 'Jane', 'Smith', 'jane.smith@example.com', '2019-05-20', 80000.00, 10),
(3, 'Robert', 'Johnson', 'robert.j@example.com', '2021-03-10', 65000.00, 20);
```

b) SELECT

Definition: Retrieves data from one or more tables.

Example :

```
-- Select all columns
SELECT * FROM employees;

-- Select specific columns
SELECT first_name, last_name, salary FROM employees;

-- Select with condition
SELECT * FROM employees WHERE salary > 70000;

-- Select with sorting
SELECT * FROM employees ORDER BY last_name ASC, first_name ASC;

-- Select with aggregation
SELECT department_id, AVG(salary) as avg_salary
FROM employees
GROUP BY department_id;
```

c) UPDATE

Definition: Modifies existing records in a table.

Example :

```
-- Update single record
UPDATE employees
SET salary = 78000.00
WHERE employee_id = 1;

-- Update multiple recordsUPDATE employees
SET salary = salary * 1.05
WHERE department_id = 10;
```

d) DELETE

Definition: Removes records from a table.

Example :

```
-- Delete specific records
DELETE FROM employees
WHERE employee_id = 3;

-- Delete all records (similar to TRUNCATE but can be rolled back)
DELETE FROM employees;
```

3) Data Control Language (DCL) Commands

Used to manage database security and permissions.

a) GRANT

Definition: Gives specific privileges to users or roles.

Example:

```
-- Grant SELECT and INSERT privileges on the employees table to user
'hr_user'
```

GRANT SELECT, INSERT ON employees TO hr_user;

b) REVOKE

Definition: Removes previously granted privileges from users.

Example:

-- Revoke INSERT privilege on the employees table from user 'hr_user'

REVOKE INSERT ON employees FROM hr_user;

4) Transaction Control Language (TCL) Commands

Used to manage transactions in the database.

a) COMMIT

Definition: Saves all changes made by the current transaction permanently.

Example:

-- Insert a record and save the change

INSERT INTO employees (employee_id, first_name, last_name, email, hire_date, salary)

VALUES (4, 'Alice', 'Brown', 'alice.brown@example.com', '2022-07-01', 70000.00);

COMMIT;

b) ROLLBACK

Definition: Undoes changes made in the current transaction.

Example:

-- Insert a record but undo it

INSERT INTO employees (employee_id, first_name, last_name, email, hire_date, salary)

VALUES (5, 'Charlie', 'Davis', 'charlie.davis@example.com', '2022-08-01', 72000.00);

ROLLBACK;

c) **SAVEPOINT**

Definition: Sets a point within a transaction to which you can later roll back.

Example:

-- Insert multiple records with savepoints

SAVEPOINT before_inserts;

INSERT INTO employees (employee_id, first_name, last_name, email, hire_date, salary)

VALUES (6, 'Emma', 'Wilson', 'emma.wilson@example.com', '2023-01-10', 68000.00);

SAVEPOINT after_first_insert;

INSERT INTO employees (employee_id, first_name, last_name, email, hire_date, salary)

VALUES (7, 'Liam', 'Miller', 'liam.miller@example.com', '2023-02-15', 69000.00);

-- Rollback to the first savepoint (undoes Liam's record, keeps Emma's)

ROLLBACK TO after_first_insert;

Practical No : 2

Introduction to PL/SQL

Definition

PL/SQL (Procedural Language extensions to SQL) is Oracle's procedural extension to SQL. It combines SQL's data manipulation power with procedural programming features like loops, conditions, and exception handling. PL/SQL is used to write stored procedures, functions, triggers, and packages in Oracle databases.

Key Features of PL/SQL

1. **Block Structure** – PL/SQL code is organized into blocks.
2. **Procedural Capabilities** – Supports loops, conditions, and exception handling.
3. **Better Performance** – Reduces network traffic by executing multiple SQL statements in a single block.
4. **Exception Handling** – Provides robust error management.
5. **Integration with SQL** – Seamlessly embeds SQL queries.

PL/SQL Block Structure

A PL/SQL block consists of three parts:

1. DECLARE (Optional)
- Defines variables, cursors, and exceptions.

2. BEGIN (Mandatory)
- Contains executable statements.

3. EXCEPTION (Optional)
- Handles runtime errors.

4. END (Mandatory)
- Marks the end of the block.

Syntax:

```
DECLARE
   -- Variable declarations
BEGIN
   -- Executable statements
EXCEPTION
   -- Error handling
END;
```

Example :

```
DECLARE
   name VARCHAR2(50) := 'John';
   age NUMBER := 25;
BEGIN
   DBMS_OUTPUT.PUT_LINE('Name: ' || name);
   DBMS_OUTPUT.PUT_LINE('Age: ' || age);
END;
```

Output

Name: John
Age: 25

Practical No : 3

Creates a simple procedure that displays the message 'Hello World!'

```
declare
   message varchar2(20) := 'Hello, World!';
begin
   dbms_output.put_line(message);
end;
/
```

```
Procedure SAY_HELLO compiled

Hello, World!
```

Practical No : 4

Use variables and constants in a PL/SQL block

1) Variables

A variable is a named storage location in a program's memory that holds a value which can be changed during program execution. Variables are used to store data that may vary as the program runs.

2) Constants

A constant is a named storage location whose value cannot be changed after initialization. Constants are used to store fixed values that remain the same throughout the program.

```
declare
   pi     constant number := 3.14159;
   radius number := 5;
   area   number;
begin
   area := pi * radius * radius;
   dbms_output.put_line('The area of the circle is: ' || area);
end;
/
```

```
The area of the circle is: 78.53975



PL/SQL procedure successfully completed.
```

Practical No : 5

Write a PL/SQL block to add two numbers

```
DECLARE
   num1 NUMBER := 10;
   num2 NUMBER := 20;
   sum NUMBER;
BEGIN
   sum := num1 + num2;
   DBMS_OUTPUT.PUT_LINE('The sum of ' || num1 || ' and ' || num2 || ' is: ' ||
sum);
END;
/
```

```
The sum of 10 and 20 is: 30


PL/SQL procedure successfully completed.
```

Practical No : 6

Create a procedure that computes the square of value of a passed value

```
declare
   input_value number := 5;
   output_value number;
begin
   compute_square(input_value, output_value);

   dbms_output.put_line('The square of ' || input_value || ' is ' || output_value);
end;
/
```

```
The square of 5 is 25



PL/SQL procedure successfully completed.
```

Practical No : 7

Create a PL/SQL block using simple IF-ELSE condition

IF-ELSE

The if-else statement is a basic control structure in programming that allows you to make decisions in your code. It checks a condition and executes different blocks of code based on whether the condition is true or false.

How It Works

1. if checks a condition.
   - If the condition is true, the code inside if runs.
   - If false, the code is skipped (or else runs if provided).
2. else (optional) runs only if the if condition is false.

```
declare
  num number := 10;
begin
  if num mod 2 = 0 then
    dbms_output.put_line(num || ' is even.');
  else
    dbms_output.put_line(num || ' is odd.');
  end if;
end;
/
```

```
10 is even.


PL/SQL procedure successfully completed.
```

Practical No : 8

Implementation of loops in PL/SQL (FOR, WHILE, DO WHILE LOOP)

Loops allow you to repeat a block of code multiple times until a condition is met. The three main types are:

1. for loop → Best when you know how many times to repeat.
2. while loop → Repeats while a condition is true (may run zero times).
3. do-while loop → Runs at least once, then checks the condition.

```
declare
  i number;
begin
   -- FOR LOOP
  dbms_output.put_line('FOR LOOP:');
  for i in 1..5 loop
     dbms_output.put_line('Iteration: ' || i);
  end loop;

   -- WHILE LOOP
  dbms_output.put_line('WHILE LOOP:');
  i := 1;
  while i <= 5 loop
     dbms_output.put_line('Iteration: ' || i);
     i := i + 1;
  end loop;

   -- DO WHILE LOOP (simulated using LOOP and EXIT WHEN)
  dbms_output.put_line('DO WHILE LOOP:');
  i := 1;
  loop
     dbms_output.put_line('Iteration: ' || i);
     i := i + 1;
     exit when i > 5;
  end loop;
end;
```

```
FOR LOOP:
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
```

```
WHILE LOOP:
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
```

```
DO WHILE LOOP:
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
```

Practical No : 9

Program to calculates the factorial of a given number

The factorial of a non-negative integer n (denoted as n!) is the product of all positive integers from 1 to n.

Formula:

n!=n×(n−1)×(n−2)×⋯×2×1$n!=n\times(n-1)\times(n-2)\times\cdots\times2\times1$
Special Cases:

- 0!=10!=1
- 1!=11!=1

```
declare
   num number := 5;
   fact number := 1;
   i number;
begin

   for i in 1..num loop
      fact := fact * i;
   end loop;

   dbms_output.put_line('Factorial of ' || num || ' is: ' || fact);
end;
```

```
Factorial of 5 is: 120



PL/SQL procedure successfully completed.
```

Practical No : 10

Implementation of Operators in PL/SQL

Operators in PL/SQL:

- Arithmetic - Perform mathematical calculations (+, -, *, /, **, MOD)
- Comparison - Evaluate conditions between values (=, !=, <>, >, <, >=, <=)
- Logical - Combine boolean expressions (AND, OR, NOT)
- Concatenation - Merge strings (||)
- Assignment - Store values in variables (:=)
- Membership - Test value presence in sets (IN, NOT IN)

```
declare
  num1   number := 10;
  num2   number := 20;
  result number;
begin
   -- Arithmetic Operators
  result := num1 + num2;
  dbms_output.put_line('Addition: ' || result);
  result := num1 - num2;
  dbms_output.put_line('Subtraction: ' || result);
  result := num1 * num2;
  dbms_output.put_line('Multiplication: ' || result);
  result := num2 / num1;
  dbms_output.put_line('Division: ' || result);
  result := mod(
    num2,
    num1
  );
  dbms_output.put_line('Modulus: ' || result);

   -- Relational Operators
  if num1 = num2 then
    dbms_output.put_line('num1 is equal to num2');
  else
    dbms_output.put_line('num1 is not equal to num2');
  end if;

  if num1 < num2 then
    dbms_output.put_line('num1 is less than num2');
```

```
  end if;
  if num1 > num2 then
    dbms_output.put_line('num1 is greater than num2');
  end if;

   -- Logical Operators
  if
    ( num1 < num2 )
    and ( num1 > 0 )
  then
    dbms_output.put_line('num1 is less than num2 and positive');
  end if;

  if ( num1 > num2 )
  or ( num1 > 0 ) then
    dbms_output.put_line('num1 is either greater than num2 or positive');
  end if;

  if not ( num1 > num2 ) then
    dbms_output.put_line('num1 is not greater than num2');
  end if;
end;
```

Arithmetic Operators: Addition: 30 Subtraction: -10 Multiplication: 200 Division: 2 Modulus: 0	Relational Operators: num1 is not equal to num2 num1 is less than num2	Logical Operators: num1 is less than num2 and positive num1 is either greater than num2 or positive num1 is not greater than num2

Practical No : 11

Implementation of Switch Case Statement in PL/SQL

1. PL/SQL uses CASE instead of SWITCH
2. Two forms:
   - CASE var WHEN value THEN ... (simple)
   - CASE WHEN condition THEN ... (searched)
3. Requires ELSE for unmatched cases

```
declare
  grade char := 'B';
begin
  case grade
    when 'A' then
      dbms_output.put_line('Excellent');
    when 'B' then
      dbms_output.put_line('Good');
    when 'C' then
      dbms_output.put_line('Average');
    when 'D' then
      dbms_output.put_line('Below Average');
    when 'F' then
      dbms_output.put_line('Fail');
    else
      dbms_output.put_line('Invalid Grade');
  end case;
end;
```

```
Good


PL/SQL procedure successfully completed.
```

Practical No : 12

Implementation of Strings in PL/SQL with Various String Function

In PL/SQL, strings are primarily handled using VARCHAR2 (variable-length) and CHAR (fixed-length) data types. PL/SQL offers numerous built-in functions for string manipulation. Common functions include LENGTH (to get the string length), SUBSTR (to extract a substring), INSTR (to find a substring's position), REPLACE (to substitute text), and the concatenation operator || (or CONCAT function) to join strings. These functions facilitate powerful text processing within database logic.

```
DECLARE
   str1 VARCHAR2(50) := 'Hello, World!';
   str2 VARCHAR2(50) := 'PL/SQL Programming';
   concatenated_str VARCHAR2(100);
   upper_str VARCHAR2(50);
   lower_str VARCHAR2(50);
   substr_str VARCHAR2(50);
   length_str NUMBER;
BEGIN
   -- Concatenate strings
   concatenated_str := str1 || ' - ' || str2;
   DBMS_OUTPUT.PUT_LINE('Concatenated String: ' || concatenated_str);

   -- Convert to uppercase
   upper_str := UPPER(str1);
   DBMS_OUTPUT.PUT_LINE('Uppercase String: ' || upper_str);

   -- Convert to lowercase
   lower_str := LOWER(str2);
   DBMS_OUTPUT.PUT_LINE('Lowercase String: ' || lower_str);

   -- Extract substring
   substr_str := SUBSTR(str1, 8, 5);
   DBMS_OUTPUT.PUT_LINE('Substring: ' || substr_str);

   -- Find length of string
   length_str := LENGTH(str1);
   DBMS_OUTPUT.PUT_LINE('Length of String: ' || length_str);
END;
```

```
Concatenated String: Hello, World! - PL/SQL Programming
Uppercase String: HELLO, WORLD!
Lowercase String: pl/sql programming
Substring: World
Length of String: 13



PL/SQL procedure successfully completed.
```

Practical No : 13

Implementation of Exception handling in PL/SQL

Exception Handling in PL/SQL is a mechanism to gracefully handle runtime errors that occur during program execution. It allows you to:

1. Catch errors (exceptions) that occur in the PL/SQL block.
2. Provide meaningful feedback instead of abrupt termination.
3. Take corrective actions (like logging errors or rolling back transactions).

Common Exceptions

- NO_DATA_FOUND (query returns no rows)
- TOO_MANY_ROWS (query returns multiple rows)
- ZERO_DIVIDE (division by zero)

```
declare
  num1   number := 10;
  num2   number := 0;
  result number;
  dummy  number;
begin
  select 1
    into dummy
    from (select 1 from dual union all select 2 from dual);
  result := num1 / num2;
  dbms_output.put_line('Result: ' || result);
exception
  when no_data_found then
     dbms_output.put_line('Error: No data found for the query.');
  when too_many_rows then
     dbms_output.put_line('Error: Query returned too many rows.');
  when zero_divide then
     dbms_output.put_line('Error: Division by zero is not allowed.');
  when others then
     dbms_output.put_line('An unexpected error occurred: ' || sqlerrm);
end;
```

```
Error: Division by zero is not allowed.


PL/SQL procedure successfully completed.
Error: No data found for the query.


PL/SQL procedure successfully completed.
Error: Query returned too many rows.


PL/SQL procedure successfully completed.
```

Practical No : 14

Implementation of Functions in PL/SQL

A PL/SQL function is :

- Accepts input parameters
- Performs computations using PL/SQL logic
- Must return exactly one value of a specified datatype
- Can be called in SQL queries or other PL/SQL blocks

Key Features:

- Supports all PL/SQL constructs (variables, conditions, loops)
- Can include exception handling blocks
- Parameters can be IN (input), OUT (output), or IN OUT (both)
- Requires explicit RETURN statement(s)

```
create or replace function calculate_bonus (
   salary           number,
   bonus_percentage number
) return number is
begin
   return salary * bonus_percentage / 100;
end;
/
begin
   dbms_output.put_line('Calculated Bonus: ' || calculate_bonus(
      50000,
      10
   ));
end;
/
```

```
Calculated Bonus: 5000


PL/SQL procedure successfully completed.
```

Practical No : 15

Implementation of triggers in PL/SQL

1. **What is a Trigger?**
   A trigger is a stored PL/SQL block that automatically executes in response to specific database events (INSERT, UPDATE, DELETE) on a table or view.

2. **When Do Triggers Fire?**
   - Before/After a DML operation
   - For each row (row-level) or once per statement (statement-level)

3. **Common Uses:**
   - Enforcing business rules
   - Logging changes (audit trails)
   - Maintaining data integrity

```
create table some_table (
  id number primary key,
  name varchar2(50)
);
create or replace trigger trg_example
before insert on some_table
for each row
begin
  dbms_output.put_line('Trigger executed: Inserting into some_table with ID: ' ||
:new.id || ' and Name: ' || :new.name);
end;
/
insert into some_table (id, name) values (1, 'Example Name');
commit;
```

```
Trigger TRG_EXAMPLE compiled

Trigger executed: Inserting into some_table with ID: 1 and Name: Example Name

1 row inserted.

Commit complete.
```

Practical No : 16

Create a stored procedure to insert and update data

```
create or replace procedure manage_data (
   p_id   number,
   p_name varchar2,
   p_mode varchar2
) is
begin
   if p_mode = 'INSERT' then
      begin
         insert into some_table (id, name) values (p_id, p_name);
         dbms_output.put_line('Data inserted: ID = ' || p_id || ', Name = ' ||
p_name);
      exception
         when dup_val_on_index then
            dbms_output.put_line('Insert skipped: ID = ' || p_id || ' already exists.');
      end;
   elsif p_mode = 'UPDATE' then
      update some_table set name = p_name where id = p_id;
      dbms_output.put_line(
         case when sql%rowcount > 0 then
            'Data updated: ID = ' || p_id || ', Name = ' || p_name
         else
            'No rows updated. ID = ' || p_id || ' not found.'
         end
      );
   else
      dbms_output.put_line('Invalid mode. Use "INSERT" or "UPDATE".');
   end if;
end;
/

begin
   manage_data(1, 'New Name', 'INSERT');
   manage_data(1, 'Updated Name', 'UPDATE');
   manage_data(2, 'Another Name', 'UPDATE');
   manage_data(3, 'Invalid Mode Test', 'DELETE');
end;
/
```

```
Procedure MANAGE_DATA compiled


Trigger executed: Inserting into some_table with ID: 1 and Name: New Name
Insert skipped: ID = 1 already exists.
Data updated: ID = 1, Name = Updated Name
No rows updated. ID = 2 not found.
Invalid mode. Use "INSERT" or "UPDATE".



PL/SQL procedure successfully completed.
```

Practical No : 17

Basics of tableau

1. What is Tableau?

Tableau is a data visualization and business intelligence tool used to create interactive dashboards, reports, and charts. It helps users analyze data visually without needing advanced programming skills.

2. Key Features of Tableau

- **Drag-and-Drop Interface** – Easy to use for creating visualizations.
- **Connectivity** – Supports Excel, SQL, Cloud, and Big Data sources.
- **Real-Time Analysis** – Updates dashboards dynamically.
- **Interactive Dashboards** – Users can filter and drill down into data.
- **Sharing & Collaboration** – Publish to Tableau Server/Cloud.

3. Tableau Workflow

- **Connect to Data** (Excel, SQL, CSV, etc.)
- **Create Worksheets** (Charts, Graphs, Tables)
- **Build Dashboards** (Combine multiple views)
- **Publish & Share** (Tableau Public/Server)

4. Common Chart Types in Tableau

- Bar Charts
- Line Graphs
- Pie Charts
- Scatter Plots
- Heat Maps
- Geographic Maps

Practical Implementation in Tableau

1. Step-by-Step Example (Sales Data Analysis)

Step 1: Load Data
- Open Tableau → Click "Connect to Data" → Select Excel/CSV.

Step 2: Create a Basic Bar Chart
- Drag "Sales" to Columns (X-axis).
- Drag "Region" to Rows (Y-axis).
- Tableau auto-generates a bar chart.

Step 3: Apply Filters
- Drag "Category" to Filters → Select specific items (e.g., "Electronics").

Step 4: Create a Dashboard
- Click "New Dashboard" → Drag the bar chart into it.
- Add a "Quick Filter" (Right-click on "Region" → Add to Filter).

Step 5: Export & Share
- Save as .twb (Tableau Workbook) or publish to Tableau Public.

2. Common Tableau Functions

- Calculated Fields (Custom formulas like Profit = Sales - Cost)
- Parameters (Dynamic user inputs, e.g., change date range)
- Trend Lines (Show patterns in data)
- Hierarchies (Drill down from Year → Quarter → Month)