**1: Creation of a Database and Writing SQL Queries to Retrieve Information from the Database**

**Aim**:  
To create a database and write SQL queries to retrieve information from it.

**Algorithm:**

1. **Create a Database**:  
   Use CREATE DATABASE to create a new database.
2. **Create Tables**:  
   Define tables with columns and data types.
3. **Insert Data**:  
   Insert sample data into the tables.
4. **Write SQL Queries**:  
   Retrieve data using the SELECT statement with WHERE, ORDER BY, and other clauses.

**SQL Coding:**

-- Step 1: Create a Database

CREATE DATABASE CompanyDB;

-- Step 2: Use the Database

USE CompanyDB;

-- Step 3: Create a Table

CREATE TABLE Employee (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(50),

emp\_salary DECIMAL(10, 2),

emp\_dept VARCHAR(50)

);

-- Step 4: Insert Data into the Table

INSERT INTO Employee (emp\_id, emp\_name, emp\_salary, emp\_dept)

VALUES (1, 'John', 50000.00, 'HR'),

(2, 'Alice', 60000.00, 'IT'),

(3, 'Bob', 55000.00, 'Finance');

-- Step 5: Retrieve Information from the Table

SELECT \* FROM Employee;

**Sample Output:**

+--------+---------+------------+---------+

| emp\_id | emp\_name| emp\_salary | emp\_dept|

+--------+---------+------------+---------+

| 1 | John | 50000.00 | HR |

| 2 | Alice | 60000.00 | IT |

| 3 | Bob | 55000.00 | Finance |

+--------+---------+------------+---------+

**2: Performing Insertion, Deletion, Modifying, Altering, Updating and Viewing Records Based on Conditions**

**Aim**:  
To perform insertion, deletion, modifying, altering, updating, and viewing records based on conditions.

**Algorithm:**

1. **Insert Data**:  
   Use INSERT INTO to add data to the table.
2. **Delete Records**:  
   Use DELETE FROM with WHERE to remove records.
3. **Modify Records**:  
   Use ALTER TABLE to modify the structure of the table.
4. **Update Data**:  
   Use UPDATE to modify existing records.
5. **View Data with Conditions**:  
   Use SELECT with WHERE, AND, OR to view records based on conditions.

**SQL Coding:**

-- Step 1: Insert New Data

INSERT INTO Employee (emp\_id, emp\_name, emp\_salary, emp\_dept)

VALUES (4, 'David', 70000.00, 'IT');

-- Step 2: Delete a Record

DELETE FROM Employee WHERE emp\_id = 3;

-- Step 3: Alter Table Structure (Add New Column)

ALTER TABLE Employee ADD emp\_age INT;

-- Step 4: Update a Record

UPDATE Employee SET emp\_salary = 75000.00 WHERE emp\_id = 2;

-- Step 5: View Data with Conditions

SELECT \* FROM Employee WHERE emp\_salary > 60000;

**Sample Output:**

**After Inserting Data**:

Query OK, 1 row affected (0.02 sec)

**After Deleting Data**:

Query OK, 1 row affected (0.01 sec)

**After Altering Table**:

Query OK, 1 row affected (0.01 sec)

**After Updating Data**:

Query OK, 1 row affected (0.02 sec)

**After Viewing Data**:

+--------+---------+------------+---------+---------+

| emp\_id | emp\_name| emp\_salary | emp\_dept| emp\_age |

+--------+---------+------------+---------+---------+

| 2 | Alice | 75000.00 | IT | NULL |

| 4 | David | 70000.00 | IT | NULL |

+--------+---------+------------+---------+---------+

**3: Create Complex Queries and Subqueries**

**Aim**:  
To create complex queries and use subqueries.

**Algorithm:**

1. **Write Simple Queries**:  
   Retrieve basic information from one or more tables.
2. **Write Complex Queries**:  
   Use multiple conditions, sorting, and grouping.
3. **Use Subqueries**:  
   Write subqueries to filter or aggregate data.

**SQL Coding:**

-- Step 1: Create a Subquery to Filter Employees with High Salary

SELECT emp\_name, emp\_salary

FROM Employee

WHERE emp\_salary > (SELECT AVG(emp\_salary) FROM Employee);

-- Step 2: Create a Complex Query with Multiple Conditions

SELECT emp\_name, emp\_dept, emp\_salary

FROM Employee

WHERE emp\_salary > 50000 AND emp\_dept = 'IT'

ORDER BY emp\_salary DESC;

**Sample Output:**

**Subquery Output**:

+---------+------------+

| emp\_name| emp\_salary |

+---------+------------+

| Alice | 75000.00 |

| David | 70000.00 |

+---------+------------+

**Complex Query Output**:

+---------+---------+------------+

| emp\_name| emp\_dept| emp\_salary |

+---------+---------+------------+

| Alice | IT | 75000.00 |

| David | IT | 70000.00 |

+---------+---------+------------+

**4: Perform Different Types of Joins**

**Aim**:  
To perform different types of joins (INNER, LEFT, RIGHT, and FULL).

**Algorithm:**

1. **Inner Join**:  
   Retrieve rows that match in both tables.
2. **Left Join**:  
   Retrieve all rows from the left table and matching rows from the right.
3. **Right Join**:  
   Retrieve all rows from the right table and matching rows from the left.
4. **Full Join**:  
   Retrieve rows that match in both tables and all non-matching rows.

**SQL Coding:**

-- Step 1: Create Department Table

CREATE TABLE Department (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(50)

);

-- Step 2: Insert Data into Department Table

INSERT INTO Department (dept\_id, dept\_name)

VALUES (1, 'HR'),

(2, 'IT'),

(3, 'Finance');

-- Step 3: Perform INNER JOIN

SELECT e.emp\_name, d.dept\_name

FROM Employee e

INNER JOIN Department d ON e.emp\_dept = d.dept\_name;

-- Step 4: Perform LEFT JOIN

SELECT e.emp\_name, d.dept\_name

FROM Employee e

LEFT JOIN Department d ON e.emp\_dept = d.dept\_name;

-- Step 5: Perform RIGHT JOIN

SELECT e.emp\_name, d.dept\_name

FROM Employee e

RIGHT JOIN Department d ON e.emp\_dept = d.dept\_name;

-- Step 6: Perform FULL OUTER JOIN

SELECT e.emp\_name, d.dept\_name

FROM Employee e

FULL OUTER JOIN Department d ON e.emp\_dept = d.dept\_name;

**Sample Output:**

**INNER JOIN Output**:

+---------+---------+

| emp\_name| dept\_name|

+---------+---------+

| John | HR |

| Alice | IT |

| David | IT |

+---------+---------+

**LEFT JOIN Output**:

+---------+---------+

| emp\_name| dept\_name|

+---------+---------+

| John | HR |

| Alice | IT |

| David | IT |

+---------+---------+

**RIGHT JOIN Output**:

+---------+---------+

| emp\_name| dept\_name|

+---------+---------+

| John | HR |

| Alice | IT |

| David | IT |

| NULL | Finance |

+---------+---------+

**FULL OUTER JOIN Output**:

+---------+---------+

| emp\_name| dept\_name|

+---------+---------+

| John | HR |

| Alice | IT |

| David | IT |

| NULL | Finance |

+---------+---------+

**5: Creation of Views, Synonyms, Sequence, Indexes, Savepoint**

**Aim**:  
To create views, synonyms, sequences, indexes, and savepoints in a database.

**Algorithm:**

1. **Create View**:  
   Create a view to simplify complex queries.
2. **Create Synonym**:  
   Create synonyms to simplify table and object referencing.
3. **Create Sequence**:  
   Generate unique numbers using sequences.
4. **Create Index**:  
   Improve query performance by creating indexes on columns.
5. **Savepoint**:  
   Set a savepoint in transactions to allow partial rollbacks.

**SQL Coding:**

-- Step 1: Create View

CREATE VIEW EmployeeView

AS SELECT emp\_name, emp\_salary, emp\_dept FROM Employee;

-- Step 2: Create Synonym CREATE SYNONYM emp\_synonym FOR Employee;

-- Step 3: Create Sequence CREATE SEQUENCE emp\_seq START WITH 1 INCREMENT BY 1;

-- Step 4: Create Index CREATE INDEX idx\_emp\_salary ON Employee(emp\_salary);

-- Step 5: Create Savepoint SAVEPOINT before\_update;

-- Perform Some Update UPDATE Employee SET emp\_salary = 80000 WHERE emp\_id = 2;

-- Rollback to Savepoint ROLLBACK TO before\_update;

---

#### \*\*Sample Output\*\*:

\*\*Creating View\*\*:

Query OK, 0 rows affected (0.01 sec)

\*\*Creating Synonym\*\*:

Query OK, 0 rows affected (0.01 sec)

\*\*Creating Sequence\*\*:

Query OK, 0 rows affected (0.01 sec)

\*\*Creating Index\*\*:

Query OK, 0 rows affected (0.01 sec)

\*\*Savepoint Rollback\*\*:

Rollback complete.

**6: Creating an Employee Database to Set Various Constraints**

**Aim:**

To create an employee database with constraints (Primary Key, Foreign Key, Unique, Check, etc.).

---

**Algorithm:**

1. \*\*Create Employee Table\*\*:

Create the table with constraints.

2. \*\*Set Primary Key\*\*:

Ensure uniqueness of employee ID.

3. \*\*Set Foreign Key\*\*:

Link tables using foreign keys.

4. \*\*Set Other Constraints\*\*:

Use `CHECK`, `UNIQUE`, and other constraints for validation.

---

**SQL Coding:**

-- Create Employee Table with Constraints

CREATE TABLE Employee (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(50) NOT NULL,

emp\_salary DECIMAL(10, 2) CHECK (emp\_salary > 0),

emp\_dept VARCHAR(50),

emp\_age INT CHECK (emp\_age BETWEEN 18 AND 65)

);

**Sample Output:**

Query OK, 0 rows affected (0.01 sec)

**7: Creating Relationships Between Databases**

**Aim**:  
To create and demonstrate relationships between two or more tables in a database using primary and foreign keys.

**Algorithm:**

1. **Create Tables**:  
   Create two or more tables, defining the appropriate primary keys and foreign key constraints to establish relationships.
2. **Insert Data**:  
   Insert sample data into the tables.
3. **Create Foreign Key Relationship**:  
   Add a foreign key constraint to establish a relationship between the tables.
4. **Query Data**:  
   Perform join operations to fetch related data from both tables.

**SQL Coding:**

-- Step 1: Create Parent Table (Department)

CREATE TABLE Department (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(50)

);

-- Step 2: Create Child Table (Employee) with Foreign Key Relationship

CREATE TABLE Employee (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(50),

dept\_id INT,

FOREIGN KEY (dept\_id) REFERENCES Department(dept\_id)

);

-- Step 3: Insert Data into Department Table

INSERT INTO Department (dept\_id, dept\_name)

VALUES (1, 'HR'),

(2, 'IT'),

(3, 'Finance');

-- Step 4: Insert Data into Employee Table

INSERT INTO Employee (emp\_id, emp\_name, dept\_id)

VALUES (101, 'John', 1),

(102, 'Alice', 2),

(103, 'Bob', 3);

-- Step 5: Query the Data using JOIN

SELECT e.emp\_id, e.emp\_name, d.dept\_name

FROM Employee e

JOIN Department d ON e.dept\_id = d.dept\_id;

**Output:**

**After Creating Tables**:

Query OK, 0 rows affected (0.02 sec)

**After Inserting Data**:

Query OK, 3 rows affected (0.03 sec)

**After Querying the Data**:

+--------+---------+-------------+

| emp\_id | emp\_name| dept\_name |

+--------+---------+-------------+

| 101 | John | HR |

| 102 | Alice | IT |

| 103 | Bob | Finance |

+--------+---------+-------------+

3 rows in set (0.01 sec)

**8: Study of PL/SQL Block**

**Aim**:  
To understand the structure and execution of a simple PL/SQL block.

**Algorithm:**

1. **Declare Variables**:  
   Declare variables to hold data.
2. **Write BEGIN and END Block**:  
   The PL/SQL block consists of DECLARE, BEGIN, EXCEPTION, and END.
3. **Perform Operations**:  
   Perform operations (e.g., assignment, arithmetic, conditional) within the BEGIN section.
4. **Exception Handling**:  
   Use exception handling to manage errors in the EXCEPTION section.

**PL/SQL Coding:**

-- PL/SQL Anonymous Block

DECLARE

v\_name VARCHAR2(50);

v\_age INT;

BEGIN

-- Assign values to variables

v\_name := 'Alice';

v\_age := 25;

-- Display values

DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_name);

DBMS\_OUTPUT.PUT\_LINE('Age: ' || v\_age);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

END;

**Output:**

Name: Alice

Age: 25

**9: Write a PL/SQL Block to Accept Input from the User**

**Aim**:  
To write a PL/SQL block that accepts input from the user.

**Algorithm:**

1. **Declare Variables**:  
   Declare variables to hold input values.
2. **Accept Input**:  
   Use & to prompt the user for input during runtime.
3. **Perform Operations**:  
   Use the input values in the block to perform some operations.

**PL/SQL Coding:**

-- PL/SQL Anonymous Block accepting user input

DECLARE

v\_name VARCHAR2(50);

v\_age INT;

BEGIN

-- Accept user input

v\_name := '&name';

v\_age := &age;

-- Display values

DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_name);

DBMS\_OUTPUT.PUT\_LINE('Age: ' || v\_age);

END;

**Output (Prompting for Input):**

Enter value for name: John

Enter value for age: 30

Name: John

Age: 30

**10: Write a PL/SQL Block that Handles All Types of Exceptions**

**Aim**:  
To write a PL/SQL block that handles all types of exceptions.

**Algorithm:**

1. **Declare Variables**:  
   Declare variables for input or calculations.
2. **Use Exception Handling**:  
   Handle different exceptions, such as NO\_DATA\_FOUND, ZERO\_DIVIDE, and OTHERS.
3. **Test Exceptions**:  
   Use conditional statements to deliberately raise exceptions.

**PL/SQL Coding:**

-- PL/SQL Block to handle exceptions

DECLARE

v\_num1 INT := 10;

v\_num2 INT := 0; -- Dividing by 0 to raise exception

v\_result INT;

BEGIN

-- Deliberately dividing by zero

v\_result := v\_num1 / v\_num2;

EXCEPTION

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Division by Zero!');

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: No Data Found!');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END;

**Output:**

Error: Division by Zero!

**11: Creation of Procedures**

**Aim**:  
To create a stored procedure in PL/SQL that performs a specific task.

**Algorithm:**

1. **Create Procedure**:  
   Write the procedure code using CREATE PROCEDURE.
2. **Call the Procedure**:  
   Invoke the procedure with EXEC or CALL.
3. **Return Values**:  
   Procedures can return values either through OUT parameters or using DBMS\_OUTPUT.

**PL/SQL Coding:**

-- Create a Procedure to display student details

CREATE OR REPLACE PROCEDURE ShowStudentDetails(p\_student\_id INT) AS

v\_name VARCHAR2(50);

v\_age INT;

BEGIN

-- Fetch student details

SELECT name, age INTO v\_name, v\_age

FROM Student

WHERE student\_id = p\_student\_id;

-- Display the details

DBMS\_OUTPUT.PUT\_LINE('Student Name: ' || v\_name);

DBMS\_OUTPUT.PUT\_LINE('Student Age: ' || v\_age);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No student found with ID ' || p\_student\_id);

END ShowStudentDetails;

**Calling the Procedure:**

-- Call the Procedure

EXEC ShowStudentDetails(1);

**Output:**

Student Name: Alice

Student Age: 20

**12: Creation of Database Triggers and Functions**

**Aim**:  
To create triggers and functions in a database.

**Algorithm:**

1. **Create a Trigger**:  
   Define a trigger using CREATE TRIGGER.
2. **Create a Function**:  
   Define a function using CREATE FUNCTION that returns a value.
3. **Test the Trigger and Function**:  
   Perform actions (insert, update, delete) to test the trigger.

**SQL Coding:**

**Creating a Trigger:**

-- Create Trigger to automatically log insert actions in Employee table

CREATE OR REPLACE TRIGGER LogEmployeeInsert

AFTER INSERT ON Employee

FOR EACH ROW

BEGIN

INSERT INTO EmployeeAudit (emp\_id, action, action\_time)

VALUES (:NEW.emp\_id, 'INSERT', SYSDATE);

END;

**Creating a Function:**

-- Create a Function to calculate employee bonus

CREATE OR REPLACE FUNCTION CalculateBonus(p\_emp\_id INT) RETURN NUMBER IS

v\_salary NUMBER;

v\_bonus NUMBER;

BEGIN

-- Fetch employee salary

SELECT salary INTO v\_salary

FROM Employee

WHERE emp\_id = p\_emp\_id;

-- Calculate bonus

v\_bonus := v\_salary \* 0.1; -- 10% bonus

RETURN v\_bonus;

END CalculateBonus;

**Output (For Trigger):**

**After Inserting Data**:

Query OK, 1 row affected (0.02 sec)

**Audit Log**:

+--------+--------+---------------------+

| emp\_id | action | action\_time |

+--------+--------+---------------------+

| 101 | INSERT | 2025-01-09 10:00:00 |

+--------+--------+