Experiment - 7

AIM: Write the queries to implement the set operations and also create views based on views.

Objective:
To understand and implement SQL set operations and view creation using SQL queries.

Theory:
Set operations in SQL are used to combine the results of two or more SELECT statements. These operations must have the same number of columns and compatible data types.

Types of Set Operations:
1. UNION – Combines results and removes duplicates.
2. UNION ALL – Combines results without removing duplicates.
3. INTERSECT – Returns only the common records.
4. MINUS (or EXCEPT) – Returns records from the first SELECT that are not in the second.

Views:
A view is a virtual table that provides a way to simplify complex queries. A view on a view means creating a new view using an existing one.

Implementation:
```
-- Step 1: Create Tables
CREATE TABLE Student (
  ID INT,
  Name VARCHAR(50)
);

CREATE TABLE Alumni (
  ID INT,
  Name VARCHAR(50)
);

-- Step 2: Insert Sample Data
INSERT INTO Student VALUES (1, 'Aman'), (2, 'Riya'), (3, 'Karan');
INSERT INTO Alumni VALUES (2, 'Riya'), (4, 'Neha');

-- Step 3: Set Operations
SELECT * FROM Student
UNION
SELECT * FROM Alumni;

SELECT * FROM Student
INTERSECT
SELECT * FROM Alumni;

SELECT * FROM Student
EXCEPT
SELECT * FROM Alumni;

-- Step 4: Create Views
CREATE VIEW Student_View AS
SELECT * FROM Student;

CREATE VIEW Top_Student_View AS
SELECT * FROM Student_View WHERE ID < 3;
```

Conclusion:
Set operations help in comparing datasets, while views simplify complex queries. Creating views on views improves data abstraction and reusability.

Experiment - 8

AIM: Demonstrate the concept of Control Structures also demonstrate the concept of Exception Handling.

Objective:
To learn PL/SQL control structures like IF-ELSE, loops, and exception handling blocks.

Theory:
Control Structures in PL/SQL:
1. Conditional Statements: IF...THEN, IF...THEN...ELSE, CASE
2. Loops: FOR loop, WHILE loop, LOOP...EXIT

Exception Handling:
PL/SQL allows trapping runtime errors using EXCEPTION blocks.
Types:
- Predefined exceptions: e.g., ZERO_DIVIDE
- User-defined exceptions

Implementation:
```
DECLARE
  a INT := 10;
  b INT := 0;
  result NUMBER;
BEGIN
  IF a > b THEN
    DBMS_OUTPUT.PUT_LINE('a is greater than b');
  ELSE
    DBMS_OUTPUT.PUT_LINE('b is greater or equal to a');
  END IF;

  FOR i IN 1..5 LOOP
```

```
    DBMS_OUTPUT.PUT_LINE('Loop iteration: ' || i);
  END LOOP;

  BEGIN
    result := a / b;
  EXCEPTION
    WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE('Division by zero occurred');
  END;
END;
```

Conclusion:
Control structures add logic to procedures, and exception handling improves reliability by catching runtime errors.

Experiment - 9

AIM: Create employee management system using MongoDB.

Objective:
To perform MongoDB operations like insert, update, find, and delete in a NoSQL document-based database.

Theory:
MongoDB is a NoSQL database that stores data in JSON-like documents.

Operations in MongoDB:
- insertOne(), find(), updateOne(), deleteOne()
- Uses db.collection syntax.

Implementation:
```
use EmployeeDB;

db.createCollection("employees");

db.employees.insertMany([
  { emp_id: 1, name: "Ravi", dept: "HR", salary: 30000 },
  { emp_id: 2, name: "Sneha", dept: "IT", salary: 50000 },
  { emp_id: 3, name: "Amit", dept: "Sales", salary: 40000 }
]);

db.employees.find();
db.employees.find({ dept: "IT" });

db.employees.updateOne({ emp_id: 2 }, { $set: { salary: 55000 } });

db.employees.deleteOne({ emp_id: 3 });
```

Conclusion:
MongoDB provides a flexible schema to manage employee data and supports CRUD operations efficiently in document-based format.

Experiment - 10

AIM: Connect employee management system using JDBC.

Objective:
To learn how to connect and manipulate data using JDBC with an employee management system.

Theory:
JDBC (Java Database Connectivity) is an API for connecting and executing queries with databases using Java.

Steps:
1. Load JDBC Driver.
2. Establish Connection.
3. Create Statement.
4. Execute Queries.
5. Close Connection.

Implementation:
```
import java.sql.*;
class ConnectDB {
    public static void main(String args[]) {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/EmployeeDB", "root",
"password");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM employees");
            while (rs.next())
                System.out.println(rs.getInt(1) + " " + rs.getString(2));
            con.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Conclusion:
JDBC is a standard way to connect Java programs to relational databases and allows data manipulation through Java code.

Experiment - 11

AIM: Executing Queries using NoSQL(document-based)

Objective:
To execute NoSQL queries using document-based operations in MongoDB.

Theory:
NoSQL databases like MongoDB store data as documents. These databases are flexible and ideal for semi-structured or unstructured data.

Common Commands:
- db.collection.find()
- db.collection.insertOne()
- db.collection.updateOne()
- db.collection.deleteOne()

Implementation:

```
use ProductDB;

db.products.insertMany([
  { product_id: 1, name: "Laptop", price: 50000 },
  { product_id: 2, name: "Mouse", price: 500 },
  { product_id: 3, name: "Keyboard", price: 800 }
]);

db.products.find();
db.products.updateOne({ product_id: 2 }, { $set: { price: 600 } });
db.products.deleteOne({ product_id: 3 });
```

HIMANSHU CHAUHAN

Conclusion:
NoSQL document-based queries allow quick data operations with flexible schema, ideal for modern applications.