DBMS pract 1: SQL queries using Insert, Select, Update, delete with operators, functions, and set operator etc.

```
>> CREATE TABLE Books (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255),
    author VARCHAR(255),
    published_year INT,
    price DECIMAL(10, 2)
);
```

*1. Insert Statements

```
-- Inserting a single record
INSERT INTO Books (title, author, published_year, price)
VALUES ('The Great Gatsby', 'F. Scott Fitzgerald', 1925, 10.99);

-- Inserting multiple records
INSERT INTO Books (title, author, published_year, price)
VALUES
('To Kill a Mockingbird', 'Harper Lee', 1960, 7.99),
('1984', 'George Orwell', 1949, 8.99),
('Pride and Prejudice', 'Jane Austen', 1813, 5.99);
```

*2.Select Statements

```
-- Select all records
SELECT * FROM Books;

-- Select specific columns
SELECT title, author FROM Books;

-- Select records with a condition
SELECT * FROM Books WHERE published_year > 1950;
```

```
-- Using aggregate functions
SELECT COUNT(*) AS TotalBooks, AVG(price) AS AveragePrice FROM Books;

-- Using ORDER BY to sort results
SELECT * FROM Books ORDER BY published_year DESC;

-- Using DISTINCT to get unique authors
SELECT DISTINCT author FROM Books;

-- Using a function (e.g., YEAR)
SELECT title, YEAR(CURRENT_DATE) - published_year AS Age FROM Books;
```

*3. Update Statements

```
-- Update a single record
UPDATE Books
SET price = price * 1.1
WHERE title = 'The Great Gatsby';

-- Update multiple records
UPDATE Books
SET price = price * 0.9
WHERE published_year < 1950;
```

*4. Delete Statements

```
-- Delete a specific record
DELETE FROM Books WHERE title = '1984';

-- Delete records with a condition
DELETE FROM Books WHERE published_year < 1900;
```

***5. Set Operators**

```
CREATE TABLE Authors (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255),
    nationality VARCHAR(100)
);
-- Example of UNION
SELECT author AS name FROM Books
UNION
SELECT name FROM Authors;
-- Example of INTERSECT
SELECT author FROM Books
INTERSECT
SELECT name FROM Authors;
```

*6. Joins

```
-- Assuming we have a foreign key relationship, an example with INNER JOIN
SELECT b.title, a.name
FROM Books b
JOIN Authors a ON b.author = a.name;
```

DBMS pract 2: SQL Queries - all types of Join, Sub-Query and View

```
CREATE TABLE Authors (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255),
    nationality VARCHAR(100)
);

CREATE TABLE Books (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255),
    author_id INT,
    published_year INT,
    price DECIMAL(10, 2),
    FOREIGN KEY (author_id) REFERENCES Authors(id)
);
```

***1. Types of Joins**

Inner Join

```
SELECT b.title, a.name
FROM Books b
INNER JOIN Authors a ON b.author_id = a.id;
```

Left Join

```
SELECT b.title, a.name
FROM Books b
LEFT JOIN Authors a ON b.author_id = a.id;
```

Right Join

```
SELECT b.title, a.name
FROM Books b
RIGHT JOIN Authors a ON b.author_id = a.id;
```

Full Outer Join

```
SELECT b.title, a.name
FROM Books b
FULL OUTER JOIN Authors a ON b.author_id = a.id;
```

*2. Subqueries

Subquery in SELECT

```
SELECT name
FROM Authors
WHERE id IN (SELECT author_id FROM Books WHERE published_year > 2000);
```

*3. Views

```
CREATE VIEW BookAuthors AS
SELECT b.title, a.name AS author_name, b.published_year
FROM Books b
JOIN Authors a ON b.author_id = a.id;

SELECT * FROM BookAuthors;

UPDATE BookAuthors
SET published_year = 2022
WHERE title = 'The Great Gatsby';
```

DBMS pract 3: MongoDB Queries using CRUD operations.

```
{
  "_id": ObjectId("..."),
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "published_year": 1925,
  "price": 10.99
}
```

1. Create Operations

Insert One Document

```
db.books.insertOne({
  title: "The Great Gatsby",
  author: "F. Scott Fitzgerald",
  published_year: 1925,
  price: 10.99
});
```

2. Read Operations

Find All Documents

```
db.books.find();
```

Find Specific Documents

```
// Find a book by title
db.books.find({ title: "1984" });

// Find books published after 1950
db.books.find({ published_year: { $gt: 1950 } });
```

3. Update Operations

```
db.books.updateOne(
   { title: "The Great Gatsby" }, // Filter
   { $set: { price: 11.99 } } // Update operation
);
```

4. Delete Operations

```
db.books.deleteOne({ title: "1984" });
```

DBMS pract 4: A PL/SQL block of code (Use of Control structure and Exception handling)

```
DECLARE
   -- Declare variables
   employee_id NUMBER := 101;  -- Change this ID to test with different inputs
   employee_salary NUMBER;
   increment_amount NUMBER := 500;

   -- Custom exception
   salary_too_high EXCEPTION;

BEGIN
   -- Fetch employee salary from employees table
   SELECT salary INTO employee_salary
```

```
    FROM employees
    WHERE employee_id = employee_id;

    -- Control structure: IF-ELSE to check if salary is above a threshold
    IF employee_salary > 5000 THEN
      RAISE salary_too_high; -- Raise exception if salary is too high
    ELSE
      -- Control structure: LOOP to increment salary
      FOR i IN 1..5 LOOP
        employee_salary := employee_salary + increment_amount;
      END LOOP;
    END IF;

    -- Display the final salary after increments
    DBMS_OUTPUT.PUT_LINE('Final Salary after increment: ' || employee_salary);

EXCEPTION
    -- Handle exceptions
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Error: No employee found with ID ' || employee_id);
    WHEN salary_too_high THEN
      DBMS_OUTPUT.PUT_LINE('Error: Salary too high for increment.');
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
/
```

DBMS pract 5: PL/SQL code block using Cursors (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor) Problem

```
DECLARE
```

```
    -- Implicit Cursor Variable
    total_employees NUMBER;

    -- Explicit Cursor to fetch employees in a specific department
    CURSOR emp_dept_cursor IS
        SELECT employee_id, first_name, last_name, department_id
        FROM employees
        WHERE department_id = 10;  -- change department_id as needed

    -- Parameterized Cursor to fetch employees by department ID
    CURSOR emp_by_dept_cursor (p_dept_id NUMBER) IS
        SELECT employee_id, first_name, last_name
        FROM employees
        WHERE department_id = p_dept_id;

    -- Variables to store employee details
    emp_id NUMBER;
    emp_first_name VARCHAR2(50);
    emp_last_name VARCHAR2(50);

BEGIN
    -- Implicit Cursor: Get total number of employees
    SELECT COUNT(*) INTO total_employees
    FROM employees;

    DBMS_OUTPUT.PUT_LINE('Total Employees: ' || total_employees);

    -- Explicit Cursor: Fetch employees in a specific department
    OPEN emp_dept_cursor;
    LOOP
        FETCH emp_dept_cursor INTO emp_id, emp_first_name, emp_last_name, department_id;
        EXIT WHEN emp_dept_cursor%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_id || ' - Name: ' || emp_first_name || ' ' ||
emp_last_name || ' - Dept ID: ' || department_id);

    END LOOP;

    CLOSE emp_dept_cursor;


    -- Cursor FOR Loop: Display employees with salary above a certain threshold

    FOR emp_rec IN (SELECT employee_id, first_name, last_name, salary FROM employees WHERE
salary > 5000) LOOP

        DBMS_OUTPUT.PUT_LINE('High Salary Employee - ID: ' || emp_rec.employee_id || ', Name: ' ||
emp_rec.first_name || ' ' || emp_rec.last_name || ', Salary: ' || emp_rec.salary);

    END LOOP;


    -- Parameterized Cursor: Fetch employees by department ID

    DBMS_OUTPUT.PUT_LINE('Employees in Department 20:');

    FOR emp_rec IN emp_by_dept_cursor(20) LOOP

        DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_rec.employee_id || ' - Name: ' ||
emp_rec.first_name || ' ' || emp_rec.last_name);

    END LOOP;


EXCEPTION

    WHEN NO_DATA_FOUND THEN

        DBMS_OUTPUT.PUT_LINE('No data found for the specified criteria.');

    WHEN OTHERS THEN

        DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);

END;

/
```