# BCS403 Program 1

**1.** Create a table called Employee & execute the following.
Employee(EMPNO,ENAME,JOB, MANAGER_NO, SAL, COMMISSION)

◉ Create a user and grant all permissions to the user.
◉ Insert the any three records in the employee table contains attributes.
◉ EMPNO,ENAME JOB, MANAGER_NO, SAL, COMMISSION and use rollback. Check the result.
◉ Add primary key constraint and not null constraint to the employee table.
◉ Insert null values to the employee table and verify the result.

**Step 1: Create a user and grant all permissions:**

```
-- Connect as a privileged user (e.g., SYS or SYSTEM)
CREATE USER emp_user IDENTIFIED BY password;

-- Grant necessary privileges to emp user (adjust privileges as needed)
GRANT CONNECT, RESOURCE, DBA TO emp_user;
```

**Step 2: Create the Employee table and insert records using rollback:**

```
-- Connect as the newly created user
CONNECT emp_user/password


-- Create the Employee table
CREATE TABLE Employee (
    EMPNO NUMBER,
    ENAME VARCHAR2(50),
    JOB VARCHAR2(50),
    MANAGER_NO NUMBER,
    SAL NUMBER,
    COMMISSION NUMBER
);


-- Insert three records into the Employee table
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (1, 'John Doe', 'Manager', NULL, 5000, 1000);

INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (2, 'Jane Smith', 'Developer', 1, 4000, NULL);

INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (3, 'Michael Brown', 'Analyst', 1, 4500, 500);


-- Use rollback to undo the insertions
ROLLBACK;


-- Check that the records were rolled back (should return 0 rows)
SELECT * FROM Employee;
```

**Step 4: Add primary key constraint and not null constraint:**

```
-- Alter table to add primary key constraint on EMPNO
ALTER TABLE Employee
ADD CONSTRAINT pk_employee PRIMARY KEY (EMPNO);



-- Alter table to add NOT NULL constraints on required columns
ALTER TABLE Employee
MODIFY (ENAME VARCHAR2(50) NOT NULL,
        JOB VARCHAR2(50) NOT NULL,
        SAL NUMBER NOT NULL);
```

**Step 5: Insert null values to the employee table and verify the result:**

```
-- Attempt to insert a record with a NULL value in a NOT NULL column
(ENAME)
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (4, NULL, 'Tester', 2, 3000, 200);


-- This insert will fail due to the NOT NULL constraint on ENAME


-- Check the result by querying the table
SELECT * FROM Employee;
```

# BCS403 Program 2

**2.** Create a table called Employee that contain attributes EMPNO,ENAME,JOB, MGR,SAL & execute the following.

◉ Add a column commission with domain to the Employee table.
◉ Insert any five records into the table.
◉ Update the column details of job.
◉ Rename the column of Employ table using alter command.
◉ Delete the employee whose Empno is 105.

**Step 1: Create the Employee Table:**

```
CREATE TABLE Employee (
    EMPNO INTEGER PRIMARY KEY,
    ENAME VARCHAR(50),
    JOB VARCHAR(50),
    MGR INTEGER,
    SAL DECIMAL(10, 2)
);
```

**Step 2: Add a Column for Commission:**

```
ALTER TABLE Employee
ADD COMMISSION DECIMAL(10, 2);
```

**Step 3: Insert Five Records into the Table:**

```
INSERT INTO Employee VALUES (101, 'Braham Kumar', 'Manager', NULL,
80000.00, 5000.00);
INSERT INTO Employee VALUES (102, 'Shubham Kumar', 'Analyst', 101,
60000.00, 3000.00);
INSERT INTO Employee VALUES (103, 'Aman Kumar', 'Clerk', 102, 40000.00,
1000.00);
INSERT INTO Employee VALUES (104, 'Rajan Gupta', 'Salesman', 101, 70000.00,
4000.00);
INSERT INTO Employee VALUES (105, 'Shiv Yadav', 'Clerk', 102, 38000.00,
800.00);
```

**Step 4: Update the Column Details of Job:**

```
UPDATE Employee
SET JOB = 'Senior Manager'
WHERE EMPNO = 101;

UPDATE Employee
SET JOB = 'Senior Analyst'
WHERE EMPNO = 102;
```

```sql
UPDATE Employee
SET JOB = 'Office Clerk'
WHERE EMPNO = 103 OR EMPNO = 105;

UPDATE Employee
SET JOB = 'Sales Executive'
WHERE EMPNO = 104;
```

### Step 5: Rename the Column of Employee Table:

```sql
ALTER TABLE Employee
RENAME COLUMN ENAME TO EMP_NAME;
```

### Step 6: Delete the Employee Whose EMPNO is 105:

```sql
DELETE FROM Employee
WHERE EMPNO = 105;
```

# BCS403 Program 3

**3.** Queries using aggregate functions(COUNT, AVG, MIN, MAX, SUM),Group by, Orderby. Employee(E_id, E_name, Age, Salary)

◉ Create Employee table containing all Records E_id, E_name, Age, Salary.
◉ Count number of employee names from employee table.
◉ Find the Maximum age from employee table.
◉ Find the Minimum age from employee table.
◉ Find salaries of employee in Ascending Order.
◉ Find grouped salaries of employees.

**Step 1: Create Employee table:**

```
CREATE TABLE Employee (
    E_id INTEGER PRIMARY KEY,
    E_name VARCHAR(100),
    Age INTEGER,
    Salary DECIMAL(10, 2)
);
```

**Step 2: Insert Five Records into the Table:**

```
INSERT INTO Employee VALUES (1, 'Braham Kumar', 30, 50000);
INSERT INTO Employee VALUES (2, 'Shubham Kumar', 25, 60000);
INSERT INTO Employee VALUES (3, 'Anjali Kumari', 35, 55000);
INSERT INTO Employee VALUES (4, 'Aman Kumar', 28, 62000);
INSERT INTO Employee VALUES (5, 'Shoaib Akhtar', 40, 70000);
```

**Step 3: Count the number of employee names from the employee table:**

```
SELECT COUNT(E_NAME) AS "NUMBER OF EMPLOYEES"
FROM EMPLOYEE;
```

**Step 4: Find the Maximum age from the employee table:**

```
SELECT MAX(AGE) AS "MAXIMUM AGE"
FROM EMPLOYEE;
```

**Step 5: Find the Minimum age from the employee table:**

```
SELECT MIN(AGE) AS "MINIMUM AGE"
FROM EMPLOYEE;
```

**Step 6: Find salaries of employees in ascending order:**

```
SELECT E_NAME, SALARY
FROM EMPLOYEE
ORDER BY SALARY ASC;
```

**Step 7: Find grouped salaries of employees:**

```
SELECT Age, SUM(SALARY) AS "TOTAL SALARY"
FROM EMPLOYEE
GROUP BY AGE;
```

---

# BCS403 Program 4

**4.** Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old & new Salary. CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)

**Step 1: Create the `CUSTOMERS` Table:**

```
CREATE TABLE CUSTOMERS (
    ID NUMBER PRIMARY KEY,
    NAME VARCHAR2(100),
    AGE NUMBER,
    ADDRESS VARCHAR2(200),
    SALARY NUMBER
);
```

**Step 2: Create the Trigger:**

```
CREATE OR REPLACE TRIGGER salary_difference_trigger
AFTER INSERT OR UPDATE OR DELETE
ON CUSTOMERS
FOR EACH ROW
DECLARE
    v_old_salary CUSTOMERS.SALARY%TYPE;
    v_new_salary CUSTOMERS.SALARY%TYPE;
    v_salary_diff CUSTOMERS.SALARY%TYPE;
BEGIN
    IF INSERTING THEN
        v_old_salary := 0;
        v_new_salary := :NEW.SALARY;
    ELSIF UPDATING THEN
        v_old_salary := :OLD.SALARY;
        v_new_salary := :NEW.SALARY;
    ELSIF DELETING THEN
        v_old_salary := :OLD.SALARY;
        v_new_salary := 0;
    END IF;

    v_salary_diff := v_new_salary - v_old_salary;

    DBMS_OUTPUT.PUT_LINE('Salary difference: ' || v_salary_diff);
END;
```

**Step 3: Test the Trigger:** Now, perform `INSERT`, `UPDATE`, and `DELETE` operations on the `CUSTOMERS` table to see the trigger in action.

**Insert Operation:**

```
INSERT INTO CUSTOMERS
VALUES (1, 'Braham Kumar', 30, 'Bangalore', 8000);
```

### Update Operation:

```
UPDATE CUSTOMERS
SET SALARY = 6000
WHERE ID = 1;
```

### Delete Operation:

```
DELETE FROM CUSTOMERS
WHERE ID = 1;
```

---

# BCS403 Program 5

**5.** Create cursor for Employee table & extract the values from the table. Declare the variables ,Open the cursor & extract the values from the cursor. Close the cursor. Employee(E_id, E_name, Age, Salary)

### Step 1: Create the `Employee` Table:

```
CREATE TABLE Employee (
    E_id NUMBER PRIMARY KEY,
    E_name VARCHAR2(100),
    Age NUMBER,
    Salary NUMBER
);
```

### Step 2: Insert value in `Employee` Table:

```
INSERT INTO EMPLOYEE VALUES (1, 'Braham Kumar', 30, 50000);
INSERT INTO EMPLOYEE VALUES (2, 'Shubham', 32, 55000);
INSERT INTO EMPLOYEE VALUES (3, 'Bikash', 28, 48000);
```

### Step 3: Create a Cursor:

```
DECLARE

    v_E_id Employee.E_id%TYPE;
    v_E_name Employee.E_name%TYPE;
    v_Age Employee.Age%TYPE;
    v_Salary Employee.Salary%TYPE;

    CURSOR employee_cursor IS
        SELECT E_id, E_name, Age, Salary FROM Employee;
BEGIN

    OPEN employee_cursor;

    LOOP
        FETCH employee_cursor INTO v_E_id, v_E_name, v_Age, v_Salary;

        EXIT WHEN employee_cursor%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE('E_id: ' || v_E_id || ', E_name: ' || v_E_name
|| ', Age: ' || v_Age || ', Salary: ' || v_Salary);
    END LOOP;

    CLOSE employee_cursor;
END;
```

---

# BCS403 Program 6

**6.** Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

**Step 1: Create Table N_RollCall:**

```
CREATE TABLE N_RollCall (
    id NUMBER,
    name VARCHAR2(100),
    roll_date DATE
);
```

**Step 2: Insert data in N_RollCall Table:**

```
INSERT INTO N_RollCall VALUES (1, 'Braham Kumar', '01-22-2015');
INSERT INTO N_RollCall VALUES (2, 'Shubham Kumar', '04-22-2016');
INSERT INTO N_RollCall VALUES (3, 'Bikash Singh', '05-22-2017');
```

**Step 3: Create Table O_RollCall:**

```
CREATE TABLE O_RollCall (
    id NUMBER,
    name VARCHAR2(100),
    roll_date DATE
);
```

**Step 4: Insert data in O_RollCall Table:**

```
INSERT INTO O_RollCall VALUES (4, 'Amit Singh', '01-22-2015');
INSERT INTO O_RollCall VALUES (5, 'Mukesh Kumar', '04-22-2016');
INSERT INTO O_RollCall VALUES (6, 'Abhay Singh', '05-22-2017');
```

**Step 5: Write the PL/SQL Block:**

```
DECLARE
    v_n_rollcall_id N_RollCall.id%TYPE;
    v_n_rollcall_name N_RollCall.name%TYPE;
    v_n_rollcall_date N_RollCall.roll_date%TYPE;

    CURSOR c_merge_rollcall_data IS
        SELECT id, name, roll_date
        FROM N_RollCall nrc
        WHERE NOT EXISTS (
            SELECT 1
            FROM O_RollCall orc
            WHERE orc.id = nrc.id
            AND orc.name = nrc.name
            AND orc.roll_date = nrc.roll_date
```

```
        );
BEGIN

    OPEN c_merge_rollcall_data;

    LOOP
        FETCH c_merge_rollcall_data INTO v_n_rollcall_id,
v_n_rollcall_name, v_n_rollcall_date;
        EXIT WHEN c_merge_rollcall_data%NOTFOUND;

        INSERT INTO O_RollCall (id, name, roll_date)
        VALUES (v_n_rollcall_id, v_n_rollcall_name, v_n_rollcall_date);
    END LOOP;

    CLOSE c_merge_rollcall_data;

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Data merged successfully from N_RollCall to
O_RollCall.');
EXCEPTION
    WHEN OTHERS THEN

        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
        ROLLBACK;
END;
```

**Step 6: To see the output command:**

```
SELECT * FROM O_RollCall;
```

---

# BCS403 Program 7

**7.** Install an Open Source NoSQL Data base MongoDB & perform basic CRUD(Create, Read, Update & Delete) operations. Execute MongoDB basic Queries using CRUD operations.

## Performing Basic CRUD Operations:

1. Create Operation:

```
use basictask;
```

### 2. Insert Operation:

#### ◉ Insert single documents:

```
db.users.insertOne({
      name: "Braham Kumar",
      age: 25,
      email: "braham@gmail.com",
      status: "inactive"
});
```

#### ◉ Insert multiple documents:

```
db.users.insertMany([
    {
        name: "Braham Kumar",
        age: 25,
        email: "braham@gmail.com",
        status: "inactive"
    },
    {
        name: "Shubham Kumar",
        age: 35,
        email: "shubham@gmail.com",
        status: "active"
    },
    {
        name: "Bikash Singh",
        age: 28,
        email: "bikash@gmail.com",
        status: "active"
    },
    {
        name: "Shoaib Akhtar",
        age: 28,
        email: "shoaib@gmail.com",
        status: "active"
    }
]);
```

### 3. Read Query:

**◉ Find all documents:**

```
db.users.find();
```

## OUTPUT:

```
[
  {
    _id: ObjectId('666c78f13c52fc36f3cdcdf6'),
    name: 'Braham Kumar',
    age: 25,
    email: 'braham@gmail.com',
    status: 'inactive'
  },
  {
    _id: ObjectId('666c78f13c52fc36f3cdcdf7'),
    name: 'Shubham Kumar',
    age: 35,
    email: 'shubham@gmail.com',
    status: 'active'
  },
  {
    _id: ObjectId('666c78f13c52fc36f3cdcdf8'),
    name: 'Bikash Singh',
    age: 28,
    email: 'bikash@gmail.com',
    status: 'active'
  },
  {
    _id: ObjectId('666c78f13c52fc36f3cdcdf9'),
    name: 'Shoaib Akhtar',
    age: 28,
    email: 'shoaib@gmail.com',
    status: 'active'
  }
]
```

**◉ Find documents with specific criteria (e.g., find all users with age greater than 25):**

```
db.users.find({ age: { $gt: 25 } });
```

## OUTPUT:

```
[
  {
    _id: ObjectId('666c78f13c52fc36f3cdcdf7'),
    name: 'Shubham Kumar',
    age: 35,
    email: 'shubham@gmail.com',
    status: 'active'
  },
  {
    _id: ObjectId('666c78f13c52fc36f3cdcdf8'),
    name: 'Bikash Singh',
    age: 28,
    email: 'bikash@gmail.com',
    status: 'active'
```

```
  },
  {
    _id: ObjectId('666c78f13c52fc36f3cdcdf9'),
    name: 'Shoaib Akhtar',
    age: 28,
    email: 'shoaib@gmail.com',
    status: 'active'
  }
]
```

**4. Update Query:**

**◉ Update a single document:**

```
db.users.updateOne(
  { name: "Bikash Singh" },
  { $set: { age: 31 } }
);
db.users.find({ age: { $gt: 31 } });
```

**OUTPUT:**

```
[
  {
    _id: ObjectId('666c78f13c52fc36f3cdcdf7'),
    name: 'Shubham Kumar',
    age: 35,
    email: 'shubham@gmail.com',
    status: 'active'
  }
]
```

**◉ Update multiple documents:**

```
db.users.updateMany(
  { age: { $gt: 28 } },
  { $set: { status: "inactive" } }
);
db.users.find({ age: { $gt: 28 } });
```

**OUTPUT:**

```
[
  {
    _id: ObjectId('666c78f13c52fc36f3cdcdf7'),
    name: 'Shubham Kumar',
    age: 35,
    email: 'shubham@gmail.com',
    status: 'inactive'
  },
  {
    _id: ObjectId('666c78f13c52fc36f3cdcdf8'),
    name: 'Bikash Singh',
    age: 31,
    email: 'bikash@gmail.com',
    status: 'inactive'
```

```
  }
]
```

**5. Delete Query:**

**◉ Delete a single document:**

```
db.users.deleteOne({ name: "Bikash Singh" });
```

**◉ Delete multiple documents:**

```
db.mycollection.deleteMany({ age: { $gt: 28 } });
```

**5. Basic Queries:**

**◉ Count documents in a collection:**

```
db.users.count();
```

**◉ Sorting documents:**

```
db.users.find().sort({ age: 1 });  // Ascending order
db.users.find().sort({ age: -1 }); // Descending order
```

**◉ Limiting the number of documents returned:**

```
db.users.find().limit(5);
```

**◉ Aggregation queries (e.g., group by and aggregate functions):**

```
db.users.aggregate([
  { $group: { _id: "$age", count: { $sum: 1 } } }
]);
```