

DBMS(BCS403) Lab Programs

((BCS403)

|  |
| --- |
| **Experiment 1** |
| Create a table called Employee & execute the following.  Employee (EMPNO, ENAME, JOB, MANAGER\_NO, SAL, COMMISSION)  1. Create a user and grant all permissions to the user.  2. Insert the any three records in the employee table contains  attributes, EMPNO, ENAME JOB, MANAGER\_NO, SAL,  COMMISSION and use rollback.  Check the result.  3. Add primary key constraint and not null constraint to the employee table  4. Insert null values to the employee table and verify the result. |

Solution Experiment 1:

Create a table called Employee & execute the following.

Employee (EMPNO, ENAME, JOB, MANAGER\_NO, SAL, COMMISSION)

CREATE TABLE Employee (

EMPNO INT,

ENAME VARCHAR(100),

JOB VARCHAR(50),

MANAGER\_NO INT,

SAL DECIMAL(10, 2),

COMMISSION DECIMAL(10, 2),

);

Question 1

1. Create a user and grant all permissions to the user.

Solution

**For MySQL:**

1. **Create a new user:**

MYSQL>CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';

Replace **'newuser'** with the desired username and **'password'** with a strong password for the user.

**2. Grant all permissions to the new user:**

MYSQL>GRANT ALL PRIVILEGES ON \*.\* TO 'newuser'@'localhost' WITH GRANT OPTION;

This command grants all privileges on all databases and tables (**\*.\***) to the new user. The

**WITH GRANT OPTION** allows the new user to grant permissions to other users.

**3.Flush the privileges to ensure that the changes take effect:**

MYSQL>FLUSH PRIVILEGES;

### For Oracle:

1. **Create a new user:**

SQL>CREATE USER newuser IDENTIFIED BY password;

Replace **newuser** with the desired username and **password** with a strong password for the user.

**2.Grant all permissions to the new user:**

SQL>GRANT ALL PRIVILEGES TO newuser;

Question 2:

2. Insert the any three records in the employee table contains attributes, EMPNO, ENAME JOB, MANAGER\_NO, SAL,COMMISSION and use rollback.

Solution

To insert records into the **Employee** table and demonstrate the use of **ROLLBACK**, follow these steps in SQL. I'll show you how to insert three records and then roll back the transaction, which undoes the inserts.

**Insert Records and Rollback in SQL**

1. **Start a transaction**: Begin a transaction to group the insert operations.
2. **Insert records**: Add three records into the **Employee** table.
3. **Rollback**: Roll back the transaction to undo the inserts.

Here’s how you can do this in SQL:

-- Start a transaction

SQL>START TRANSACTION;

-- Insert three records into the Employee table

SQL>INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER\_NO, SAL, COMMISSION) VALUES

(101, 'John Doe', 'Manager', 101, 75000, 5000),

(102, 'Jane Smith', 'Developer', 102, 60000, 3000),

(103, 'Sam Brown', 'Analyst', 103, 55000, 2000);

-- Rollback the transaction

SQL>ROLLBACK;

Question 3:

3. Add primary key constraint and not null constraint to the employee table

Solution:

SQL>CREATE TABLE Employee (

EMPNO INT PRIMARY KEY,

ENAME VARCHAR(50) NOT NULL,

JOB VARCHAR(50),

MANAGER\_NO INT,

SAL DECIMAL(10, 2),

COMMISSION DECIMAL(10, 2),

FOREIGN KEY (MANAGER\_NO) REFERENCES Employee(EMPNO)

);

Question 4

4. Insert null values to the employee table and verify the result.

Solution:

To insert **NULL** values into the **Employee** table for certain columns, you can simply omit those columns from the **INSERT** statement, or explicitly insert **NULL** where needed. Here's how you can do it:

### Inserting NULL Values into the Employee Table

-- Inserting NULL values for the MANAGER\_NO and COMMISSION columns

SQL>INSERT INTO Employee (EMPNO, ENAME, JOB, SAL) VALUES

(104, 'Alice Johnson', 'Assistant', 45000),

(105, 'Bob Roberts', 'Intern', 30000, NULL);

-- Displaying the contents of the Employee table

SQL>SELECT \* FROM Employee;

**Explanation for above:**

1. **Inserting Records**:
   * For **EMPNO** 104, **ENAME** 'Alice Johnson', **JOB** 'Assistant', and **SAL** 45000 are provided. Since **MANAGER\_NO** and **COMMISSION** are not provided, they will be inserted as **NULL**.
   * For **EMPNO** 105, **ENAME** 'Bob Roberts', **JOB** 'Intern', and **SAL** 30000 are provided. Additionally, **MANAGER\_NO** is not provided and will default to **NULL**. **COMMISSION** is explicitly set to **NULL**.

|  |
| --- |
| **Experiment 2** |
| Create a table called Employee that contain attributes EMPNO,ENAME,JOB, MGR,SAL & execute the following.  1. Add a column commission with domain to the Employee table.  2. Insert any five records into the table.  3. Update the column details of job  4. Rename the column of Employ table using alter command.  5. Delete the employee whose Empno is 105. |

First step create an Employee table,

SQL>CREATE TABLE Employee (

EMPNO INT PRIMARY KEY,

ENAME VARCHAR(255),

JOB VARCHAR(255),

MGR INT,

SAL DECIMAL(10, 2)

);

Question 1:

1. Add a column commission with domain to the Employee table.

Solution:

To add a new column named **COMMISSION** to the existing **Employee** table with a specified domain, you can use the **ALTER TABLE** statement in SQL. Here's how you can do it:

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

SQL>Describe Employee;

Question 2:

2. Insert any five records into the table.

Solution:

SQL>INSERT INTO Employee (EMPNO, ENAME, JOB, MGR, SAL, COMMISSION) VALUES (101, 'John Doe', 'Manager', NULL, 50000, 2000), (102, 'Jane Smith', 'Developer', 1, 40000, 1500),

(103, 'Alice Johnson', 'Analyst', 1, 45000, NULL),

(104, 'Bob Roberts', 'Assistant', 2, 35000, 1000),

(105, 'Emily Brown', 'Intern', 2, 30000, 500);

Question 3:

3.Update the column details of job

Solution:

SQL>UPDATE Employee

SET JOB = 'Senior Developer'

WHERE EMPNO = 2;

SQL>UPDATE Employee

SET JOB = 'Executive Assistant'

WHERE EMPNO = 4;

SQL>UPDATE Employee

SET JOB = 'Junior Developer'

WHERE EMPNO = 5;

SQL>Select \* from Employee;

Question 4:

4. Rename the column of Employ table using alter command.

Solution:

SQL>ALTER TABLE Employee RENAME COLUMN MGR TO MANAGER\_NO

SQL>ALTER TABLE Employee RENAME COLUMN ENAME TO EMP\_NAME;

SQL>Describe Employee;

Question 5:

5.Delete the employee whose Empno is 105.

SQL>DELETE FROM Employee WHERE EMPNO = 105;

SQL> Select \* from Employee;

|  |
| --- |
| **Experiment 3** |
| Queries using aggregate functions(COUNT,AVG,MIN,MAX,SUM),Group by, Orderby. **Employee(E\_id, E\_name, Age, Salary)**  1. Create Employee table containing all Records E\_id, E\_name, Age, Salary.  2. Count number of employee names from employee table  3. Find the Maximum age from employee table.  4. Find the Minimum age from employee table.  5. Find salaries of employee in Ascending Order.  6. Find grouped salaries of employees. |

First step create an Employee table as **Employee(E\_id, E\_name, Age, Salary)**

Question 1:

1. Create Employee table containing all Records E\_id, E\_name, Age, Salary.

Solution:

SQL>CREATE TABLE Employee (

E\_id INT PRIMARY KEY,

E\_name VARCHAR(255),

Age INT,

Salary DECIMAL(10, 2)

);

SQL>INSERT INTO Employee (E\_id, E\_name, Age, Salary) VALUES

(1, 'John Doe', 30, 50000.00),

(2, 'Jane Smith', 28, 60000.00),

(3, 'Alice Johnson', 35, 70000.00),

(4, 'Bob Roberts', 32, 55000.00),

(5, 'Emily Brown', 27, 48000.00);

Question 2:

2. Count number of employee names from employee table

Solution:

To count the number of employee names from the **Employee** table, you can use the **COUNT()** function in SQL. Here's how you can do it:

SQL>SELECT COUNT(E\_name) AS TotalEmployees FROM Employee;

This SQL query will count the number of non-null values in the **E\_name** column of the **Employee** table and alias the result as **TotalEmployees**.

If you want to count all rows regardless of whether **E\_name** is null or not, you can use:

SQL>SELECT COUNT(\*) AS TotalEmployees FROM Employee;

Question 3:

3. Find the Maximum age from employee table.

Solution:

SQL>SELECT MAX(Age) AS MaximumAge FROM Employee;

Question 4:

4. Find the Minimum age from employee table.

Solution:

SQL> SELECT MIN(Age) AS MinimumAge FROM Employee;

Question 5:

5. Find salaries of employee in Ascending Order.

Solution:

**Ascending Order**

SQL> SELECT E\_id, E\_name, Salary FROM Employee ORDER BY Salary

ASC;

**Descending Order**

SQL>SELECT E\_id, E\_name, Salary FROM Employee ORDER BY Salary

DESC;

Question 6:

6. Find grouped salaries of employees.

Solution:

SQL>SELECT Salary, COUNT(\*) AS NumEmployees FROM Employee GROUP BY Salary;

This SQL query will count the number of employees (**NumEmployees**) for each unique salary (**Salary**) value in the **Employee** table.

|  |
| --- |
| **Experiment 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 CUSTOMERS Table- **CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)**

SQL>CREATE TABLE CUSTOMERS (

ID INT PRIMARY KEY,

NAME VARCHAR(255),

AGE INT,

ADDRESS VARCHAR(255),

SALARY DECIMAL(10, 2)

);

SQL>DESC CUSTOMERS;

Question:

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.

Solution:

DELIMITER //

CREATE TRIGGER salary\_difference\_trigger

AFTER INSERT OR UPDATE OR DELETE ON CUSTOMERS

FOR EACH ROW

BEGIN

DECLARE old\_salary DECIMAL(10, 2);

DECLARE new\_salary DECIMAL(10, 2)

IF (OLD.SALARY IS NOT NULL AND NEW.SALARY IS NOT NULL) THEN

SET old\_salary = OLD.SALARY;

SET new\_salary = NEW.SALARY;

IF (old\_salary <> new\_salary) THEN

SELECT CONCAT('Salary difference: $', ABS(new\_salary - old\_salary))

AS 'Salary Difference'

FROM DUAL;

END IF;

END IF;

END;

//

DELIMITER ;

Explanation of above program:

* **DELIMITER //**: Changes the delimiter to **//** so that the entire trigger definition can be included in a single SQL statement.
* **CREATE TRIGGER salary\_difference\_trigger**: Starts the definition of the trigger named **salary\_difference\_trigger**.
* **AFTER INSERT OR UPDATE OR DELETE ON CUSTOMERS**: Specifies that the trigger should fire after INSERT, UPDATE, or DELETE operations on the **CUSTOMERS** table.
* **FOR EACH ROW**: Indicates that the trigger should be executed once for each row affected by the operation.
* **BEGIN ... END**: Encloses the body of the trigger.
* **DECLARE old\_salary DECIMAL(10, 2);** and **DECLARE new\_salary DECIMAL(10, 2);**: Declares variables to store the old and new salary values.
* The **IF** statement checks if both the old and new salary values are not NULL.
* If the old and new salaries are different, it calculates the absolute difference between them and displays the result using a **SELECT** statement.
* **FROM DUAL;**: Used in MySQL to execute expressions without selecting data from any table.
* **DELIMITER ;**: Resets the delimiter back to semicolon.

This trigger will display the salary difference whenever a row is inserted, updated, or deleted in the **CUSTOMERS** table. Adjust the data types and lengths as needed based on your requirements.

|  |
| --- |
| **Experiment 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 Employee Table- **Employee(E\_id, E\_name, Age, Salary)**

SQL>CREATE TABLE Employee (

E\_id INT PRIMARY KEY,

E\_name VARCHAR(255),

Age INT,

Salary DECIMAL(10, 2)

);

Question:

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.

Solution:

-- Declare variables

SQL>DECLARE emp\_id INT;

DECLARE emp\_name VARCHAR(255);

DECLARE emp\_age INT;

DECLARE emp\_salary DECIMAL(10, 2);

-- Declare cursor for Employee table

DECLARE emp\_cursor CURSOR FOR

SELECT E\_id, E\_name, Age, Salary

FROM Employee;

-- Open the cursor

OPEN emp\_cursor;

-- Loop to fetch values from the cursor

FETCH emp\_cursor INTO emp\_id, emp\_name, emp\_age, emp\_salary;

WHILE @@FETCH\_STATUS = 0 DO

-- Output the values

SELECT CONCAT('Employee ID: ', emp\_id) AS Employee\_Info;

SELECT CONCAT('Employee Name: ', emp\_name) AS Employee\_Info;

SELECT CONCAT('Employee Age: ', emp\_age) AS Employee\_Info;

SELECT CONCAT('Employee Salary: ', emp\_salary) AS Employee\_Info;

-- Fetch next row

FETCH emp\_cursor INTO emp\_id, emp\_name, emp\_age, emp\_salary;

END WHILE;

-- Close the cursor

CLOSE emp\_cursor;

Explanation for the above:

* **DECLARE**: This keyword is used to declare variables and cursors.
* **emp\_cursor**: This cursor is declared to fetch records from the **Employee** table.
* **FETCH emp\_cursor INTO ...**: This statement fetches the next row of data from the cursor into the specified variables.
* **@@FETCH\_STATUS**: This system variable returns the status of the last fetch operation. It will be 0 if the fetch is successful.
* **WHILE @@FETCH\_STATUS = 0 DO ... END WHILE**: This loop iterates over each row fetched by the cursor.
* **CLOSE emp\_cursor**: This statement closes the cursor after fetching all the required data.

|  |
| --- |
| **Experiment 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. |

here's a PL/SQL block of code using a parameterized cursor to merge data from the **N\_RollCall** table into the **O\_RollCall** table:

SQL>DECLARE

v\_n\_id N\_RollCall.ID%TYPE;

v\_n\_name N\_RollCall.Name%TYPE;

v\_n\_status N\_RollCall.Status%TYPE;

v\_exists BOOLEAN;

CURSOR n\_cursor IS

SELECT ID, Name, Status

FROM N\_RollCall;

BEGIN

-- Open the cursor

OPEN n\_cursor;

-- Loop through each row in the cursor

LOOP

-- Fetch the next row

FETCH n\_cursor INTO v\_n\_id, v\_n\_name, v\_n\_status;

EXIT WHEN n\_cursor%NOTFOUND;

-- Check if the data exists in O\_RollCall

SELECT COUNT(\*)

INTO v\_exists

FROM O\_RollCall

WHERE ID = v\_n\_id;

-- If the data does not exist, insert it into O\_RollCall

IF v\_exists = 0 THEN

INSERT INTO O\_RollCall (ID, Name, Status)

VALUES (v\_n\_id, v\_n\_name, v\_n\_status);

END IF;

END LOOP;

-- Close the cursor

CLOSE n\_cursor;

END;

/

Explanation for the above:

* We declare variables to store the data retrieved from the **N\_RollCall** table (**v\_n\_id**, **v\_n\_name**, **v\_n\_status**), as well as a boolean variable (**v\_exists**) to check if the data already exists in **O\_RollCall**.
* We define a cursor (**n\_cursor**) to select data from the **N\_RollCall** table.
* We open the cursor to start fetching data.
* We loop through each row in the cursor.
* Inside the loop, we fetch the data from the cursor into the variables.
* We check if the data with the same ID already exists in the **O\_RollCall** table. If it does not exist (**v\_exists = 0**), we insert the data into the **O\_RollCall** table.
* After processing all rows, we close the cursor.

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

### Step 1: Install MongoDB

#### Windows:

1. Download the MongoDB installer from the [MongoDB Download Center](https://www.mongodb.com/try/download/community).
2. Run the installer and follow the installation wizard.
3. MongoDB will be installed in the **C:\Program Files\MongoDB\Server\<version>** directory by default.

### Step 2: Start MongoDB Server

After installing MongoDB, you need to start the MongoDB server. Here's how you can do it:

#### Windows:

1. Open Command Prompt as an administrator.
2. Navigate to the MongoDB installation directory's **bin** folder. For example:

cd C:\Program Files\MongoDB\Server\<version>\bin

3.Run the **mongod** command to start the MongoDB server.

**Step 3: Connect to MongoDB**

1. Open a new Terminal or Command Prompt window.
2. Run the following command to connect to MongoDB:

**Mongo**

### Step 4: Perform Basic CRUD Operations

#### **Create (Insert):**

**To insert documents into a collection, you can use the insertOne() or insertMany() method.**

// Insert a single document

db.collection.insertOne({ name: "John", age: 30 });

// Insert multiple documents

db.collection.insertMany([

{ name: "Jane", age: 25 },

{ name: "Alice", age: 35 }

]);

#### **Read (Find):**

**To retrieve documents from a collection, you can use the find() method.**

// Find all documents in the collection

db.collection.find();

// Find documents that match a specific condition

db.collection.find({ name: "John" });

#### **Update:**

**To update documents in a collection, you can use the updateOne() or updateMany() method.**

// Update a single document

db.collection.updateOne({ name: "John" }, { $set: { age: 31 } });

// Update multiple documents

db.collection.updateMany({ age: { $lt: 30 } }, { $set: { status: "Young" } });

***Delete:***

***To delete documents from a collection, you can use the deleteOne() or deleteMany() method.***

// Delete a single document

db.collection.deleteOne({ name: "John" });

// Delete multiple documents

db.collection.deleteMany({ age: { $gte: 30 } });

**Step 5: Exit MongoDB Shell**

**To exit the MongoDB shell, simply type exit and press Enter.**

**That's it! You've installed MongoDB and performed basic CRUD operations. You can now explore more advanced features and functionalities of MongoDB as needed.**

**Some MongoDB Commands to Know**

### Check the Current Database

To check the current database you are using, you can use the **db** command:

**db**

This command will display the name of the current database.

### View Collections

To view the collections in the current database, you can use the **show collections** command:

**show collections**

This command will list all collections in the current database.

### Find Documents

To view the documents in a collection, you can use the **find()** method:

**db.collectionName.find()**

Replace **collectionName** with the name of the collection you want to query. For example, if you inserted documents into a collection named **employees**, you can use:

**db.employees.find()**

### Query Specific Documents

To query specific documents based on certain criteria, you can use the **find()** method with a query condition:

**db.collectionName.find({ criteria })**

Replace **collectionName** with the name of the collection and **criteria** with the query condition. For example, to find documents where the **name** field is "John":

**db.employees.find({ name: "John" })**