

TABLE OF CONTENTS

#	Title	Lab Date	Page #	Lab Submission Date	Remarks
1)	Set Up SQL Environment and Execute Basic SQL commands via Command Line.	2083/01/30	1-3	2083/01/30	Michael Hays
2)	Databas Schema Design for University Course Registration System	2083/01/30	4-7	2083/02/06	Michael Hays
3)	SQL Join and Pattern Matching	2083/02/06	8-11	2083/02/13	Michael Hays
4)	SQL NESTED QUERIES	2083/02/13	12-15	2083/02/20	Michael Hays
5)	GROUP BY CLAUSE AND HAVING CLAUSE	2083/02/20	16-19	2083/02/27	Michael Hays
6)	Demonstrate the process of database normalization from UNF to BCNF using a suitable example. Create the required table using SQL, identify functional dependence, show xai final normal form	2083/02/27	20-24	2083/03/03	Michael Hays
7)	Research the MongoDB NoSQL database and demonstrate its basic syntax for common database operations. Create a sample database and collection, then perform (CRUD) operations.	2083/03/03	25-26	2083/03/09	Michael Hays
8)	GRANT and REVOKE Privileges in SQL.	2083/03/09	27-29	2083/03/18	Michael Hays

TITLE:- SET UP SQL ENVIRONMENT AND EXECUTE BASIC SQL COMMANDS VIA COMMAND LINE.

OBJECTIVE:-

- To install and configure an SQL database environment
- To understand how to access the database using the command line interface.
- To learn basic SQL commands.
- To create databases and tables using SQL.
- To insert, retrieve, update, and delete records from a table.
- To gain practical knowledge of database management operation

THEORY:-

- SQL (Structured Query language) is a standard language used to communicate with relational database management system. It is used to store, manipulate, retrieve and manage data in database.

Types of SQL Commands

1) DDL (Data Definition Language)

- Used to define database structures.

Examples:

- CREATE
- ALTER
- DROP

2) DML (Data Manipulation Language)

- Used to manipulate data.

Examples:

- INSERT
- UPDATE
- DELETE

3) DQL (Data Query Language)

- Used to retrieve data.

Example:

- SELECT

4) DCL (Data Control language)

- Used for access control

Examples:

- GRANT
- REVOKE

REQUIREMENTS:-

- Before starting the lab, ensure the following are available:
 - Computer system
 - SQL database software
 - Command prompt / terminal
 - Internet connection (optional)

PROCEDURE:-

Step 1: Install SQL Software

- Install any SQL database software such as:
 - MySQL official Website

Step 2:- Open Command Line Interface

Open:

- Command prompt (Windows)
- Terminal (Linux/Mac)

Step 3:- Start SQL Server

For MySQL, Use:

`mysql -u root -p`

Enter the password to log in.

Step 4:- Create a Database

`CREATE Database college;`

Step 5:- Select the Database

`USE college;`

Step 6:- Create a Table

`CREATE TABLE student (
id INT PRIMARY KEY,
name VARCHAR(50),`

```
faculty VARCHAR(30)
);
```

Step 7: Insert Records

```
INSERT INTO student VALUES
(1, 'Ram', 'BCA'),
(2, 'Sita', 'BIT'),
(3, 'Hari', 'BIM');
```

Step 8:- Display Table Data

```
SELECT * FROM student;
```

Step 9:- Update a Record

```
UPDATE student
SET faculty='BSc CSIT'
WHERE id=2;
```

Step 10: DELETE a Record

```
DELETE FROM student
where id=3;
```

Step 11: Display final Table

```
SELECT * FROM student;
```

OUTPUT

id	name	faculty
1	Ram	BCA
2	Sita	BSc CSIT

RESULT:-

- The SQL environment was successfully configured, and basic SQL commands were executed through the command line interface.

CONCLUSION:-

- In this lab we successfully installed and configured the SQL environment and learned how to use SQL commands through the command line interface.

Michael Joseph 2083/01/30.

TITLE:- Consider a University Course Registration System that maintains data about Students, Courses and Enrollments

1) Design the relational schema for the following tables:

- STUDENT (Student-ID, Name, Department, Semester)
- COURSE (Course-ID, Course-Name, Credits)
- ENROLLMENT (Student-ID, Course-ID, Grade)

2) Draw a schema diagram showing relationships and keys.

OBJECTIVE:-

- To understand the concept of relational database schema.
- To design table for a University Course Registration System.
- To identify primary keys and foreign keys.
- To understand relationship between entities.
- To create a schema diagram representing the database structure.

THEORY:-

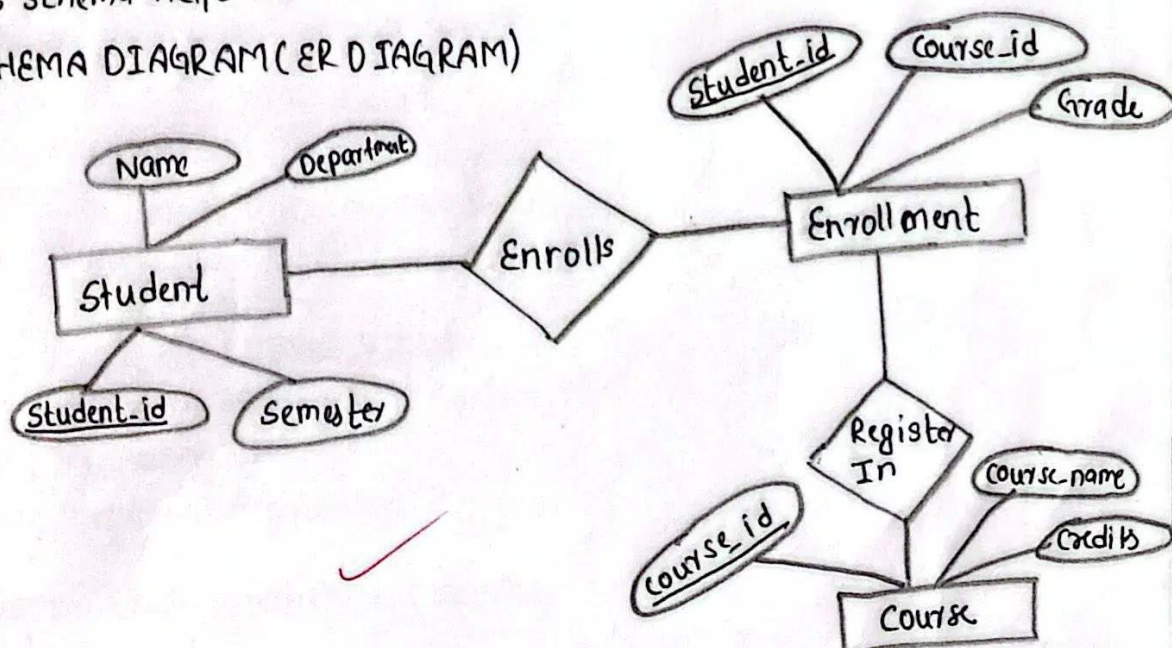
- A Database schema is the logical structure of a database that defines tables, attributes, keys and relationship between tables.

In a University Course Registration System, the database stores information about:

- Students
- Courses
- Enrollments

This schema helps maintain data integrity and avoids redundancy.

SCHEMA DIAGRAM (ER DIAGRAM)



SQL COMMAND

-- Create STUDENT table

```
CREATE TABLE STUDENT(  
    Student-ID INT PRIMARY KEY,  
    Name VARCHAR(100) NOT NULL,  
    Department VARCHAR(50),  
    Semester INT  
);
```

-- Create COURSE table

```
CREATE TABLE COURSE(  
    Course-ID INT PRIMARY KEY,  
    Course-Name VARCHAR(100) NOT NULL,  
    Credits INT  
);
```

-- Create Enrollment table

```
CREATE TABLE ENROLLMENT(  
    Student-ID INT,  
    Course-ID INT,  
    Grade VARCHAR(2),  
    PRIMARY KEY (Student-ID, Course-ID),  
    FOREIGN KEY (Student-ID) REFERENCES STUDENT (Student-ID),  
    FOREIGN KEY (Course-ID) REFERENCES COURSE (Course-ID)  
);
```

EXAMPLE

INSERT INTO STUDENT VALUES

```
(101, 'Ram Sharma', 'Computer Science', 3),  
(102, 'Sita Thapa', 'Information Technology', 2),  
(103, 'Hari Gurung', 'Software Engineering', 4);
```

INSERT INTO COURSE VALUES

```
(201, 'Database Management System', 3),  
(202, 'Operating System', 4),  
(203, 'Software Engineering', 3);
```

INSERT INTO ENROLLMENT VALUES

```
(101, 201, 'A'),  
(101, 202, 'B+'),  
(102, 201, 'A-'),  
(103, 203, 'B');
```

DISPLAY TABLES

VIEW STUDENT Table

SELECT * FROM STUDENT;

OUTPUT Example

Student-ID	Name	Department	Semester
101	Ram Sharma	Computer Science	3
102	Sita Thapa	Information Technology	2
103	Hari Gurung	Software Engineering	4

VIEW Course Table

SELECT * FROM COURSE

Course-ID	Course-Name	Credits
201	Database Management System	3
202	Operating system	4
203	Software Engineering	3

VIEW ENROLLMENT Table

SELECT * FROM ENROLLMENT

Student-ID	Course-ID	Grade
101	201	A
101	202	B+
102	201	A-
103	203	B

CONCLUSION:-

- The STUDENT, COURSE, and ENROLLMENT tables were successfully created using SQL commands. Primary keys uniquely identify records, while foreign keys establish relationships between tables. The ENROLLMENT table acts as a bridge entity to manage the many-to-many relationship between students and courses effectively.

Prithvi
2083102100.

TITLE:- SQL JOIN AND PATTERN MATCHING

OBJECTIVE:-

- Understand the concept of SQL JOIN operations.
- Learn different types of joins in SQL.
- Retrieve related data from multiple tables.
- Understand pattern matching using the LIKE OPERATOR.
- Use wildcard characters % and - for searching data.

THEORY:-

SQL JOIN

- A JOIN clause in SQL is used to combine rows from two or more tables based on a related column between them. Joins help retrieve meaningful information stored across multiple tables.

Types of JOIN

1) INNER JOIN

- Returns only the matching records from both tables.

2) LEFT JOIN

- Returns all records from the left table and matching records from the right table. If no match exists, NULL values are displayed.

3) RIGHT JOIN

- Returns all records from the right table and matching records from the left table.

Pattern Matching

- Pattern matching in SQL is performed using the LIKE operator. It helps search records based on specific patterns.

Wild cards Used

Wild card

%

-

Meaning

Represents zero or more characters

Represents a single character.

1. Create Tables

CREATE STUDENT Table

```
CREATE TABLE STUDENT (  
    Student_ID INT PRIMARY KEY,  
    Name VARCHAR (50)  
);
```

CREATE COURSE Table

```
CREATE TABLE COURSE (  
    Course_ID INT PRIMARY KEY,  
    Student_ID INT,  
    Course_Name VARCHAR (50)  
);
```

2. Insert Records

Insert Records into STUDENT Table

INSERT INTO STUDENT VALUES

```
(1, 'Ram', 'CSIT'),  
(2, 'Sita', 'BCA'),  
(3, 'Havi', 'BIM'),  
(4, 'Gita', 'CSIT'),  
(5, 'Ramesh', 'BCA');
```

Insert Records into COURSE Table

INSERT INTO COURSE VALUES

```
(101, 1, 'Database'),  
(102, 2, 'Networking'),  
(103, 3, 'Operating system'),  
(104, 1, 'Web Technology'),  
(105, 4, 'Data Structure');
```

3. JOIN QUERIES

A. INNER JOIN

```
SELECT STUDENT.Student_ID, STUDENT.Name, COURSE.Course_Name  
FROM STUDENT  
INNER JOIN COURSE  
ON STUDENT.Student_ID = COURSE.Student_ID;
```

OUTPUT

Student_ID	Name	Course_Name
1	Ram	Database
2	Sita	Networking
3	Hari	Operating System
1	Ram	web Technology
4	Gita	Data Structure

B. LEFT JOIN

```
SELECT STUDENT.Student_ID, STUDENT.Name, COURSE.Course_Name  
FROM STUDENT  
LEFT JOIN COURSE  
ON STUDENT.Student_ID = COURSE.Student_ID;
```

C. RIGHT JOIN

```
SELECT STUDENT.Student_ID, STUDENT.Name, COURSE.Course_Name  
FROM STUDENT  
RIGHT JOIN COURSE  
ON STUDENT.Student_ID = COURSE.Student_ID;
```

4. Pattern Matching Queries

A. Display names starting with letter 'R'

```
SELECT * FROM STUDENT  
WHERE Name LIKE 'R%';
```

OUTPUT

Student_ID	Name	Department
1	Ram	CSIT
5	Ramesh	BCA

B. Display Names Ending with letter 'a'.

```
SELECT * FROM STUDENT  
WHERE Name LIKE '%a';
```

OUTPUT

Student_ID	Name	Department
2	Sita	BCA
4	Gita	CSIT

C. Find Names Containing Letter 'i'

```
SELECT * FROM STUDENT  
WHERE Name LIKE '%i%';
```

OUTPUT

Student_ID	Name	Department
2	Sita	BCA
3	Hari	BIM
4	Gita	CSIT

D. Find Names Using _Wildcard

```
SELECT * FROM STUDENT  
WHERE Name LIKE '____';
```

OUTPUT:

Student_ID	Name	Department
2	Sita	BCA
3	Hari	BIM
4	Gita	CSIT

CONCLUSION:-

- This lab demonstrates the use of SQL Join operations and pattern matching. JOINS helps achieve related data from multiple tables, while the LIKE operator and wildcards (% , _) help search records based on specific patterns.

Nikhil

TITLE:- SQL NESTED QUERIES

OBJECTIVE:-

- To understand the concept of nested queries in SQL.
- To learn how to use one query inside another query.
- To retrieve data based on conditions obtained from another query.
- To perform complex database operations efficiently.

THEORY:-

- A Nested Query is a query written inside another SQL query. The inner query executes first and its result is ~~added~~ used by the outer query.

Syntax:-

```
SELECT column_name  
FROM table_name  
WHERE column_name operator  
(  
    SELECT column_name  
    FROM table_name  
    WHERE condition  
);
```

Example Table: STUDENT

S-ID	Name	Course	Marks
1	Ram	BCA	85
2	Rita	BIT	78
3	Sita	BCA	92
4	Hari	BBS	65
5	Gita	BIT	88

Create Table

```
CREATE TABLE STUDENT(  
    SID INT PRIMARY KEY,  
    Name VARCHAR(20),  
    Course VARCHAR(10),  
    Marks INT  
);
```

Insert Values

INSERT INTO STUDENT VALUES

(1, 'Ram', 'BCA', 85),
(2, 'Rita', 'BIT', 78),
(3, 'Sita', 'BCA', 92),
(4, 'Hari', 'BBS', 65),
(5, 'Gita', 'BIT', 88)

Example 1: Find students whose marks are greater than the average mark

```
SELECT Name, Marks  
FROM STUDENT  
WHERE Marks >
```

```
(  
  SELECT AVG(Marks)  
  FROM STUDENT  
);
```

OUTPUT

Name	Marks
Ram	85
Sita	92
Gita	88

Example 2: Find students belonging to the same course as Ram.

```
SELECT Name, Course  
FROM STUDENT  
WHERE Course =
```

```
(  
  SELECT Course  
  FROM STUDENT  
  WHERE Name = 'Ram'  
);
```

OUTPUT

Name	Course
Ram	BCA
Sita	BCA

Example 3:- Find students whose marks are equal to the highest marks.

```
SELECT Name, Marks
FROM STUDENT
WHERE Marks =
(
  SELECT MAX (Marks)
  FROM STUDENT
);
```

OUTPUT

Name	Marks
Sita	92

Example 4:- Find student enrolled in BIT course using the IN operator.

```
SELECT Name, Course
FROM STUDENT
WHERE Course IN
(
  SELECT Course
  FROM STUDENT
  WHERE Course = 'BIT'
);
```

OUTPUT

Name	Course
Rita	BIT
Gita	BIT

Example 5:- Find students whose marks are greater than all students in BBS.

```
SELECT Name, Marks
FROM STUDENT
WHERE Marks > ALL
(
  SELECT Marks
  FROM STUDENT
  WHERE Course = 'BBS'
);
```

OUTPUT

Name	Marks
Ram	85
Rita	78
Sita	92
Gita	88

CONCLUSION:-

- SQL nested queries are powerful tools that allow one query to be embedded inside another query. They help in solving complex data retrieval problems efficiently and make database operations more flexible and organized.

Michael

TITLE:- GROUP BY Clause and HAVING Clause in SQL

OBJECTIVE:-

- To understand the use of the GROUP BY clause in SQL.
- To learn how to group records based on common values.
- To understand the use of aggregate function such as COUNT(), SUM(), AVG(), MAX(), and MIN().
- To learn how to filter grouped data using the HAVING clause.

THEORY:-

GROUP BY Clause

- The GROUP BY clause is used to arrange rows having the same value in specified columns into groups. It is commonly used with aggregate function to perform calculations on each group.

Syntax

```
SELECT column-name, aggregate-function(column-name)
FROM table-name
GROUP BY column-name;
```

HAVING Clause

- The HAVING clause is used to filter groups created by the GROUP BY clause. It works similarly to the WHERE clause, but WHERE filters rows before grouping whereas HAVING filters groups after grouping.

Syntax

```
SELECT column-name, aggregate-function(column-name)
FROM table-name
GROUP BY column-name
HAVING condition;
```

Example Table: STUDENT

SID	Name	Course	Marks
1	Ram	BCA	85
2	Rita	BIT	78
3	Sita	BCA	92
4	Harri	BBS	65
5	Gita	BIT	88
6	Ramesh	BCA	75

Create Table

```
CREATE TABLE STUDENT (  
  SID INT PRIMARY KEY,  
  Name VARCHAR (20),  
  Course VARCHAR (10),  
  Marks INT  
);
```

Insert Values

```
INSERT INTO STUDENT VALUES  
(1, 'Ram', 'BCA', 85),  
(2, 'Rita', 'BIT', 78),  
(3, 'Sita', 'BCA', 92),  
(4, 'Hari', 'BBS', 65),  
(5, 'Gita', 'BIT', 88),  
(6, 'Ramesh', 'BCA', 75);
```

Example 1: Count the number of students in each course.

```
SELECT Course, COUNT(*) AS Total_Students  
FROM STUDENT  
GROUP BY Course;
```

OUTPUT	Total_Students
BCA	3
BIT	2
BBS	1

Example 2:- find the average marks of students in each course.

```
SELECT Course, AVG(Marks) AS Average_Marks  
FROM STUDENT  
GROUP BY Course;
```

OUTPUT	Average_Marks
Course	
BCA	84
BIT	83
BBS	65

Example 3:- Find the highest marks in each course.

```
SELECT course, MAX (Marks) AS Highest_Marks  
FROM STUDENT  
GROUP BY Course;
```

OUTPUT

Course	Highest_Marks
BCA	92
BIT	88
BBS	65

Example 4:- Display courses having more than one student using HAVING clause.

```
SELECT course, COUNT(*) AS Total_Students  
FROM STUDENT  
GROUP BY Course  
HAVING COUNT(*) > 1;
```

OUTPUT

Course	Total_Students
BCA	3
BIT	2

Example 5:- Display courses whose average marks are greater than 80.

```
SELECT course, AVG (Marks) AS Average_Marks  
FROM STUDENT  
GROUP BY Course  
HAVING AVG (Marks) > 80;
```

OUTPUT

Course	Average_Marks
BCA	84
BIT	83



CONCLUSION:-

- The GROUP BY clause is used to group rows with similar values and perform aggregate operations on them. The HAVING clause is used to filter these groups based on specified conditions.

Pranav

TITLE:- Demonstrate the process of database normalization from UNF to BCNF using a suitable example. create the required tables using SQL, identify functional dependencies, and show the resulting relations at each normal form.

OBJECTIVES:-

- To understand the process of database normalization
- To convert an UNF relation into 1NF, 2NF, 3NF and BCNF
- To identify functional dependencies.
- To create normalized tables using SQL.

THEORY:-

- Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It divides large tables into smaller related tables and establishes relationships among them.

Normal forms

1. UNF (Unnormalized form)

- Contains repeating groups and multivalued attributes.

2) 1NF (First Normal form)

- Eliminates repeating groups, all attributes contain atomic values.

3) 2NF (Second Normal form)

- Removes partial dependencies

4) 3NF (Third Normal form)

- Removes transitive dependencies

5) BCNF (Boyce - Codd Normal form)

- Every determinant must be a candidate key

Example:- Student Course Registration System

UNF (Unnormalized form)

Student ID	Student Name	Course ID	Course Name	Instructor ID	Instructor Name
S101	Ram	C101	DBMS	101	Hari
S101	Ram	C102	OS	102	Sita
S102	Shyam	C101	DBMS	101	Hari
S103	Gita	C103	SAD	103	Ramesh

Functional Dependencies

1. StudentID $\rightarrow$ StudentName
2. CourseID $\rightarrow$ CourseName, InstructorID
3. InstructorID $\rightarrow$ InstructorName
4. (StudentID, CourseID) $\rightarrow$ StudentName, CourseName, InstructorID, InstructorName

Candidate Key:

(StudentID, CourseID)

First Normal form (1NF)

- All repeating groups are removed and values are atomic.

Relation

STUDENT_COURSE(StudentID, StudentName, CourseID, CourseName, InstructorID, InstructorName)

SQL Table

```
CREATE TABLE Student_Course(  
  StudentID VARCHAR(10),  
  StudentName VARCHAR(30),  
  CourseID VARCHAR(10),  
  CourseName VARCHAR(30),  
  InstructorID VARCHAR(10),  
  InstructorName VARCHAR(30),  
  PRIMARY KEY (StudentID, CourseID)  
);
```

Second Normal form (2NF)

- Partial dependencies are removed.

Functional Dependencies

- StudentID $\rightarrow$ StudentName
- CourseID $\rightarrow$ CourseName, InstructorID
- InstructorID $\rightarrow$ InstructorName

Relations

STUDENT

StudentID

S101

S102

S103

StudentName

Ram

Shyam

Gita

COURSE

Course ID	Course Name	Instructor ID	Instructor Name
C101	DBMS	I01	Hari
C102	OS	I02	Sita
C103	SAD	I03	Ramesh

ENROLLMENT

Student ID	Course ID
S101	C101
S101	C102
S102	C101
S103	C103

SQL Commands

```
CREATE TABLE STUDENT(  
    StudentID VARCHAR(10) PRIMARY KEY,  
    StudentName VARCHAR(30)  
);
```

```
CREATE TABLE COURSE(  
    CourseID VARCHAR(10) PRIMARY KEY,  
    CourseName VARCHAR(30),  
    InstructorID VARCHAR(10),  
    InstructorName VARCHAR(30),  
);
```

```
CREATE TABLE ENROLLMENT(  
    StudentID VARCHAR(10),  
    CourseID VARCHAR(10),  
    PRIMARY KEY (StudentID, CourseID),  
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),  
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)  
);
```

Third Normal Form (3NF)

- Transitive dependencies are removed.

Since:

CourseID $\rightarrow$ Instructor ID

Instructor ID $\rightarrow$ InstructorName

Instructor Name depends transitively on Course ID.

Relations

STUDENT (StudentID, StudentName)

COURSE (CourseID, CourseName, InstructorID)

INSTRUCTOR (InstructorID, InstructorName)

ENROLLMENT (StudentID, CourseID)

SQL Commands

```
CREATE TABLE Student(
```

```
    StudentID VARCHAR(10) PRIMARY KEY,
```

```
    StudentName VARCHAR(30)
```

```
);
```

```
CREATE TABLE Instructor(
```

```
    InstructorID VARCHAR(10) PRIMARY KEY,
```

```
    InstructorName VARCHAR(30),
```

```
);
```

```
CREATE TABLE Course(
```

```
    CourseID VARCHAR(10) PRIMARY KEY,
```

```
    CourseName VARCHAR(30)
```

```
    InstructorID VARCHAR(10)
```

```
    FOREIGN KEY (InstructorID)
```

```
        REFERENCES Instructor (InstructorID)
```

```
);
```

```
CREATE TABLE Enrollment(
```

```
    StudentID VARCHAR(10),
```

```
    CourseID VARCHAR(10)
```

```
    PRIMARY KEY (StudentID, CourseID),
```

```
    FOREIGN KEY (StudentID)
```

```
        REFERENCES Student (StudentID),
```

```
    FOREIGN KEY (CourseID)
```

```
        REFERENCES Course (CourseID)
```

```
);
```

Boyce-Codd Normal form (BCNF)

- for each functional dependency;

- Student ID $\rightarrow$ Student Name
- Instructor ID $\rightarrow$ Instructor Name
- Course ID $\rightarrow$ Course Name, Instructor ID
- (Student ID, Course ID) $\rightarrow$ Entire Enrollment Record

Since every determinant is a candidate key, the relations are already in BCNF

CONCLUSION:-

- Normalization is an important database design technique that minimizes redundancy and avoids update, insertion and deletion anomalies. In this experiment, the unnormalized relation was transformed step-by-step into 1NF, 2NF, 3NF and BCNF, resulting in an efficient and consistent database structure.

Subash Kumar

TITLE:- Research the MongoDB NoSQL database and demonstrate its basic syntax for common database operations. Create a sample database and collection, then perform CRUD (Create, Read, Update, Delete) operations using MongoDB commands.

Objective:-

- To understand the concept of NoSQL databases.
- To learn the basic feature of MongoDB.
- To perform Create, Delete, Read, Update operation using MongoDB commands.

Theory:-

MongoDB is a document-oriented NoSQL database. Unlike relational databases such as MySQL, which stores data in tables made of rows and columns, MongoDB stores ^{data} as flexible, JSON-like documents. This structure is well suited to applications where the data doesn't always fit a fixed schema.

Example:-

Create a collection and Insert Documents

```
db.students.insertOne({  
  name: "Samikshya Shrestha",  
  faculty: "BICTE",  
  semester: 5,  
  fee: 25000  
})
```

```
db.students.insertMany([  
  {name: "Biplav Shrestha", faculty: "BBA", semester: 3, fee: 22000},  
  {name: "Jandeepa Shrestha", faculty: "BED", semester: 2, fee: 20000}  
])
```

Read Documents

```
db.students.find()
```

```
db.students.find({faculty: "BICTE"})
```

```
db.students.findOne({name: "Biplav Shrestha"})
```

Update Documents

```
db.students.updateOne(  
  $name: "samikshya Shrestha",  
  $set: { $inc: 27000 }  
)
```

```
// Increase semester for all BICTE students  
db.students.updateMany(  
  $faculty: "BICTE",  
  $inc: { semester: 1 }  
)
```

Delete Documents

```
// Delete one student record  
db.students.deleteOne($name: "Sandeepa Shrestha")  
  
// Delete every student in the BED faculty  
db.students.deleteMany($faculty: "BED")
```

Conclusion:-

This lab demonstrated the basic use of MongoDB by creating a database and collection and performing CRUD operations. It helped in understanding how MongoDB stores and manages data efficiently using a document-based NoSQL approach.

Signature

TITLE:- GRANT and REVOKE Privileges in SQL.

OBJECTIVE:-

- To understand SQL privileges.
- To learn how to grant privileges to users.
- To learn how to revoke privileges from users.
- To practice access control in a database.

THEORY

- SQL provides security by allowing database administrators to control user access to database objects. The GRANT command is used to give permissions to users, while the REVOKE command is used to remove those permissions.

GRANT Syntax

```
GRANT privilege-name  
ON table-name  
TO user-name;
```

REVOKE Syntax

```
REVOKE privilege-name  
ON table-name  
FROM user-name;
```

Common SQL Privileges

- SELECT - Read data
- INSERT - Add new records
- UPDATE - Modify existing records.
- DELETE - Remove records.
- ALL PRIVILEGES - Grant all permissions.

Example

Step 1:- Create a Database

```
CREATE DATABASE collegeDB;
```

Step 2:- Use the Database

USE collegedb;

Step 3:- Create a Table

```
CREATE TABLE student(  
    StudentID INT PRIMARY KEY,  
    Name VARCHAR(50),  
    Course VARCHAR(30)  
);
```

Step 4:- Insert Sample Data

```
INSERT INTO student  
VALUES  
(1, 'RAM', 'BITE'),  
(2, 'Sita', 'BBA');
```

Step 5: Grant SELECT Permission

```
GRANT SELECT  
ON student  
TO user1;
```

Step 6: Grant INSERT Permission

```
GRANT INSERT  
ON student  
TO user1;
```

Step 7: Revoke INSERT Permission

```
REVOKE INSERT  
ON student  
FROM user1;
```

Step 8: Revoke SELECT Permission

```
REVOKE SELECT  
ON student  
FROM user1;
```

OUTPUT:-

- User1 can view records after the SELECT privilege is granted.
- User1 can insert records after the INSERT privilege is granted.
- After REVOKE INSERT, User1 cannot insert new records.
- After REVOKE SELECT, User1 cannot view the table data.

CONCLUSION:-

- This lab demonstrated how to use the GRANT and REVOKE commands in SQL to control user access to database objects. These commands help improve database security by allowing administrators to manage user permission effectively.

Nikhil