DAYANANDA SAGAR ACADEMY OF
TECHNOLOGY &MANAGEMENT

Accredited by NAAC with A+ Grade 6
Programs Accredited by NBA
(CSE, ISE, ECE, EEE, MECH,CIVIL)

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

2024-2025

DBMS LAB MANNUAL
(BCS403)

Compiled by

Prof. Jyothis K P

Prof. Lakshmi M R

Dr. Nagaraj M Lutimath

Dr. Nandini C
HOD, CSE, DSATM

Dr. M Ravishankar
Principal, DSATM

INSTITUTION VISION AND MISSION

Vision of the Institution

To strive at creating the institution a center of highest caliber of learning, so as to create an overall intellectual atmosphere with each deriving strength from the other to be the best of engineers, scientists with management &design skills.

Mission of the Institution:

- To serve its region, state, the nation and globally by preparing students to make
- meaningful contributions in an increasing complex global society challenge.
- To encourage, reflection on and evaluation of emerging needs and priorities with state of art infrastructure at institution.
- To support research and services establishing enhancements in technical, health, economic, human and cultural development.
- To establish inter disciplinary center of excellence, supporting/ promoting student's implementation.
- To increase the number of Doctorate holders to promote research culture on campus.
- To establish IIPC, IPR, EDC, innovation cells with functional MOU's supporting student's quality growth.

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY &MANAGEMENT

Approved by AICTE

Accredited by NAAC with A+ Grade 6 Programs Accredited by NBA(CSE, ISE, ECE, EEE, MECH,CIVIL)

QUALITY POLICY

Dayananda Sagar Academy of Technology and Management aims at achieving academic excellence through continuous improvement in all spheres of Technical and Management education. In pursuit of excellence cutting-edge and contemporary skills are imparted to the utmost satisfaction of the students and the concerned stakeholders

OBJECTIVES & GOALS

- Creating an academic environment to nurture and develop competent entrepreneurs, leaders and professionals who are socially sensitive and environmentally conscious.
- Integration of Outcome Based Education and cognitive teaching and learning strategies to enhance learning effectiveness.
- Developing necessary infrastructure to cater to the changing needs of Business and Society.
- Optimum utilization of the infrastructure and resources to achieve excellence in all areas of relevance.
- Adopting learning beyond curriculum through outbound activities and creative assignments.
- Imparting contemporary and emerging techno-managerial skills to keep pace with the changing global trends.
- Facilitating greater Industry-Institute Interaction for skill development and employability enhancement.
- Establishing systems and processes to facilitate research, innovation and entrepreneurship for holistic development of students.
- Implementation of Quality Assurance System in all Institutional processes

Vision and Mission of the Department

Department Vision

Epitomize CSE graduate to carve a niche globally in the field of computer science to excel in the world of information technology and automation by imparting knowledge to sustain skills for the changing trends in the society and industry.

Department Mission

M1: To educate students to become excellent engineers in a confident and creative environment through world-class pedagogy.

M2: Enhancing the knowledge in the changing technology trends by giving hands-on experience through continuous education and by making them to organize & participate in various events.

M3: Impart skills in the field of IT and its related areas with a focus on developing the required competencies and virtues to meet the industry expectations.

M4: Ensure quality research and innovations to fulfill industry, government & social needs.

M5: Impart entrepreneurship and consultancy skills to students to develop self-sustaining life skills in multi-disciplinary areas.

Programme Educational Objectives

PEO 1: Engage in professional practice to promote the development of innovative systems and optimized solutions for Computer Science and Engineering.

PEO 2: Adapt to different roles and responsibilities in interdisciplinary working environment by respecting professionalism and ethical practices within organization and society at national and international level.

PEO 3: Graduates will engage in life-long learning and professional development to acclimate the rapidly changing work environment and develop entrepreneurship skills.

Program Outcomes (POs)

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Programme Specific Outcomes

PSO 1: Foundation of Mathematical Concepts: Ability to use mathematical methodologies to crack problem using suitable mathematical analysis, data structure and suitable algorithm.

PSO 2: Foundation of Computer System: Ability to interpret the fundamental concepts and methodology of computer systems. Students can understand the functionality of hardware and software aspects of computer systems.

PSO 3: Foundations of Software Development: Ability to grasp the software development lifecycle and methodologies of software systems. Possess competent skills and knowledge of software design process. Familiarity and practical proficiency with a broad area of programming concepts and provide new ideas and innovations towards research.

PSO 4: Foundations of Multi-Disciplinary Work: Ability to acquire leadership skills to perform professional activities with social responsibilities, through excellent flexibility to function in multi-disciplinary work environment with self-learning skills.

BCS403: DBMS LABORATORY

Course objectives: This course will enable students to

Sl. No	Course Objectives
1	Understand the fundamental concepts of Database Management Systems
2	Apply the concepts of Database for the given Scenario
3	Analyse given scenario and use appropriate Database Technique
4	Design database or application for a given scenario
5	Implement a database application for a given real world problem.

Database: A Database is a collection of interrelated data and a Database Management Systemis a a software system that enables users to define, create and maintain the database and which provides controlled access to the database

SQL: It is structured query language, basically used to pass the query to retrieve andmanipulate the information from database. Depending upon the nature of query, SQL is divided into different components:

- **DDL**(Data Definition Language)
- **DML**(Data Manipulation Language)
- **DCL**(Data Control Language)

□ Table□View □Index

Introduction to SQL:

□SQL stands for Structured Query Language
□SQL lets you access and manipulate databases
□SQL is an ANSI (American National Standards Institute) standard

Commands of SQL are grouped into four languages.
1>DDL

DDL is abbreviation of Data Definition Language. It is used to create and modify the structure of database objects in database.

Examples: CREATE, ALTER, DROP, RENAME, TRUNCATE statements

2>DML

DML is abbreviation of Data Manipulation Language. It is used to retrieve, store, modify, delete, insert and update data in database.

Examples: SELECT, UPDATE, INSERT, DELETE statements

3>DCL

DCL is abbreviation of Data Control Language. It is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

Examples: GRANT, REVOKE statements

4>TCL

TCL is abbreviation of Transactional Control Language. It is used to manage different transactions occurring within a database.

Examples: COMMIT, ROLLBACK statements

Data Definition Language (DDL)

1. Data definition Language (DDL) is used to create, rename, alter, modify, drop, replace, and delete tables, Indexes, Views, and comment on database objects; and establish a default database.
2. The DDL part of SQL permits database tables to be created or deleted. It also defines indexes (keys), specify links between tables, and impose constraints between tables. The most important DDL statements in SQL are:

□CREATE TABLE- Creates a new table
□ALTER TABLE- Modifies a table
□DROP TABLE- Deletes a table
□
□TRUNCATE -Use to truncate (delete all rows) a table.
□CREATE INDEX- Creates an index (search key)
□DROP INDEX- Deletes an index

1. The CREATE TABLE Statement

The CREATE TABLE statement is used to create a table in a database.
Syntax

CREATE TABLE table name

(attr1_name attr1_datatype(size) attr1_constraint,
attr2_name attr2_datatype(size) attr2_constraint,….);

SQL Constraints

Constraints are used to limit the type of data that can go into a table.

Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

□
We will focus on the following constraints:
□

□NOT NULL UNIQUE
□PRIMARY KEY
□FOREIGN KEY
□CHECK DEFAULT

Add constraint after table creation using alter table option

Syntax - Alter table add constraint constraint_name constraint_type(Attr_name) Example -

Alter table stud add constraint
prk1 primary key(rollno);
Drop constraint:

Syntax- Drop Constraint Constraint_name;
Example - Drop constraint prk1;

2. The Drop TABLE Statement
Removes the table from the
database Syntax

DROP TABLE table_name;

3. The ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
Syntax

To add a column in a table, use the following syntax:

ALTER TABLE table_name

ADD column_name datatype;

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE table_name DROP COLUMN column_name;

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

4. The RENAME TABLE Statement
Rename the old table to new table;
Syntax

Rename old_tabname to new_tabname;

5. The TRUNCATE TABLE Statement

The ALTER TABLE Statement is used to truncate (delete all rows) a table.
Syntax

To truncate a table, use following syntax: TRUNCATE TABLE table_name;

6. CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.A view contains rows and columns, just like a

real table. The fields in a view are fields from one or more real tables in the database.
Syntax

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name

WHERE condition;
```

7. SQL Dropping a View

You can delete a view with the DROP VIEW command.
Syntax

```
DROP VIEW view_name;
```

8 . Create Index Statement

1. Index in SQL is created on existing tables to retrieve the rows quickly. When there are thousands of records in a

table, retrieving information will take a long time.

2. Therefore indexes are created on columns which are accessed frequently, so that the information can be

retrieved quickly.

3. Indexes can be created on a single column or a group of columns. When a index is created, it first sorts the

data and then it assigns a ROWID for each row.

Syntax

```
CREATE INDEX index_name

ON table_name (column_name1, column_name2...);
```

index_name is the name of the INDEX.

table_name is the name of the table to which the indexed column belongs.
column_name1, column_name2. is the list of columns which make up the INDEX.

9. Drop Index Statement
Syntax:DROP INDEX index_name;
10. Create Synonym statement
1.
2. Use the CREATE SYNONYM statement to create a synonym, which is an alternative name

for a table, view,

sequence, procedure, stored function, package, materialized view.

3. Synonyms provide both data independence and location transparency. Synonyms permit applications to

function without modification regardless of which user owns the table or view and regardless of which

database holds the table or view.

4. You can refer to synonyms in the following DML statements: SELECT, INSERT, UPDATE, DELETE

Syntax - Create synonym synonym-name for object-name;
Example-Create synonym synonym_name for table_**name**
DML command

Data Manipulation Language (DML) statements are used for managing data in database. DMLcommands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

1) INSERT command

Insert command is used to insert data into a table. Following is its general syntax,
INSERT into table-name values(data1,data2,..)

Lets see an example,

Consider a table Student with following fields.
S_id S_Name age

INSERT into Student values(101,'Adam',15);

The above command will insert a record into Student table.
S_id S_Name age

101 Adam 15

2) UPDATE command

Update command is used to update a row of a table. Following is its general syntax,
UPDATE table-name set column-name = value where condition;

Lets see an example,

update Student set age=18 where s_id=102;
Example to Update multiple columns

UPDATE Student set s_name='Abhi',age=17 where s_id=103;

3) Delete command

Delete command is used to delete data from a table. Delete command can also be used with condition to delete a particular row. Following is its general syntax,

DELETE from table-name;

Example to Delete all Records from a Table
DELETE from Student;

The above command will delete all the records from Student table.
Example to Delete a particular Record from a Table

Consider Student table

DELETE from Student where s_id=103;

SQL Functions

SQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc. SQL functions are divided into two catagories,

- **Aggregate Functions**
- **Scalar Functions**

Aggregate Functions

These functions return a single value after calculating from a group of values.Following are some frequently used Aggregate functions.

1) AVG ()
Average returns average value after calculating from values in a numeric column.
Its general Syntax is,

SELECT AVG (column_name) from table_name
e.g.SELECT avg(salary) from Emp;

2) COUNT ()
Count returns the number of rows present in the table either based on some condition or without condition. Its general Syntax is,

SELECT COUNT (column_name) from table-name;
Example using COUNT ()

Consider following Emp table
eid name age salary

401 Anu 22 9000

402 Shane 29 8000

SQL query to count employees, satisfying specified condition is,
SELECT COUNT (name) from Emp where salary = 8000;

3) FIRST ()
First function returns first value of a selected column
Syntax for FIRST function is,

SELECT FIRST (column_name) from table-name
SQL query

SELECT FIRST (salary) from Emp;

4) LAST ()
LAST return the return last value from selected column
Syntax of LAST function is,

SELECT LAST(column_name) from table-name
SQL query will be,

SELECT LAST(salary) from emp;

5) MAX()
MAX function returns maximum value from selected column of the table.
Syntax of MAX function is,

SELECT MAX(column_name) from table-name
SQL query to find Maximum salary is,

SELECT MAX(salary) from emp;

6) MIN()

MIN function returns minimum value from a selected column of the table.

Syntax for MIN function is,

SELECT MIN(column_name) from table-name
SQL query to find minimum salary is,
SELECT MIN(salary) from emp;

7) SUM()
SUM function returns total sum of a selected columns numeric values.
Syntax for SUM is,

SELECT SUM(column_name) from table-name
SQL query to find sum of salaries will be,
SELECT SUM(salary) from emp;

Scalar Functions

Scalar functions return a single value from an input value. Following are soe frequently used Scalar Functions.

1) UCASE()
UCASE function is used to convert value of string column to Uppercase character.
Syntax of UCASE,

SELECT UCASE(column_name) from table-name
Example of UCASE()

SQL query for using UCASE is,
SELECT UCASE(name) from emp;

2) LCASE()
LCASE function is used to convert value of string column to Lowecase character.
Syntax for LCASE is:

SELECT LCASE(column_name) from table-name

3) MID()
MID function is used to extract substrings from column values of string type in a table.
Syntax for MID function is:

SELECT MID(column_name, start, length) from table-name

4) ROUND()
ROUND function is used to round a numeric field to number of nearest integer. It is used on Decimal point values. Syntax of Round function is,

SELECT ROUND(column_name, decimals) from table-name

Operators:

AND and OR operators are used with Where clause to make more precise conditions for fetching data from database by combining more than one condition together.

1) AND operator

AND operator is used to set multiple conditions with Where clause.
Example of AND

SELECT * from Emp WHERE salary < 10000 AND age > 25

2) OR operator

OR operator is also used to combine multiple conditions with Where clause. The only difference between AND and OR is their behavior. When we use AND to combine two or more than two conditions, records satisfying all the condition will be in the result. But in case of OR, atleast one condition from the conditions specified must be satisfied by any record to be in the result.

Example of OR

SELECT * from Emp WHERE salary > 10000 OR age > 25
Set Operation in SQL

SQL supports few Set operations to be performed on table data. These are used to get meaningful

results from data, under different special conditions.

3) Union

UNION is used to combine the results of two or more Select statements. However, it will eliminate duplicate rows from its result set. In case of union, number of columns and datatype must be same in both the tables.

Example of UNION
select * from First
UNION

select * from second

4) Union All

This operation is similar to Union. But it also shows the duplicate rows.Union All query will be like,

select * from First
UNION ALL

select * from second

5) Intersect

Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of Intersect the number of columns and datatype must be same. MySQL does not support INTERSECT operator.

Intersect query will be,
select * from First
INTERSECT

select * from second

6) Minus

Minus operation combines result of two Select statements and return only those result which belongs to first set of result. MySQL does not support INTERSECT operator.

Minus query will be,
select * from First
MINUS

select * from second

LIST OF PROGRAMS

Sl. No.	Experiments/Programs	COs
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.	CO5
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.	CO5
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 employeetable 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.	CO5
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)	CO5

5	Create cursor for Employee table & extract the values from the table. Declare the variables ,Open the cursor & extrct the values from the cursor. Close the cursor. Employee(E_id, E_name, Age, Salary)	CO5
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.	CO5
7	Develop a simple web application using PHP to manage a product inventory. Connect to a MySQL database to store product information (name, price, quantity). Allow users to view the inventory, add new products, and update existing product details.	CO5
8	Analyze different concurrency control mechanisms (locking, timestamps) for managing concurrent access to a data base. Simulate scenarios with multiple users modifying the same data and demonstrate how locking or timestamps prevent data inconsistencies.	CO5
9	Create a database for a library management system. Include tables for books, authors, and members. Use appropriate data types and constraints (primary key, foreign key, etc.). Write queries to: List all books by a specific author. Find books borrowed by a particular member but not yet returned. Calculate the total number of books in each category.	CO5
10	Analyze a sample database schema provided by your instructor. Identify any normalization anomalies (redundancy, inconsistency). Apply normalization techniques (1NF, 2NF, 3NF, BCNF) to bring the schema to a higher normal form, minimizing data redundancy.	CO5
Open ended Programs		
1	Create a database for a university course registration system. Include tables for courses, students, and enrollments. Use views to simplify complex queries. Write queries using views to: List all courses offered by a specific department with the number of enrolled students. Find students enrolled in more than two courses this semester. Create a trigger to automatically update a "total credits" field for a student whenever they enroll in a new course.	
2	Simulate a bank transaction system using SQL. Implement transactions with ACID properties (Atomicity, Consistency, Isolation, Durability) using appropriate SQL statements (BEGIN TRANSACTION, COMMIT, ROLLBACK). Develop functionalities for deposit, withdrawal, and fund transfer between accounts, ensuring data integrity	

	across operations.	
3	Implement user authentication and access control mechanisms for a database using SQL. Create different user roles with varying privileges (read, write, delete) for specific tables. Develop a program to demonstrate secure login and authorization based on user roles.	

Experiment1

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.

Create a user and grant all permissions to the user

```
CREATE USER myuser IDENTIFIED BY mypassword;
GRANT ALL PRIVILEGES ON Employee TO myuser;
```

1. Create the Employee

```
table CREATE TABLE Employee (
   EMPNO INT,
   ENAME VARCHAR(50), JOB
   VARCHAR(50),
   MANAGER_NO INT, SAL
   DECIMAL(10, 2),
   COMMISSION DECIMAL(10, 2)
);
```

2. Insert three records into the employee table and rollback

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES
   (1, 'John Doe', 'Manager', NULL, 5000.00, 1000.00),
   (2, 'Jane Smith', 'Developer', 1, 4000.00, 500.00),
   (3, 'Bob Johnson', 'Analyst', 1, 3500.00, NULL);
ROLLBACK;
```

Check if the records were inserted

```
SELECT * FROM Employee;
```

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

```
ALTER TABLE Employee
ADD CONSTRAINT PK_Employee PRIMARY KEY (EMPNO);
```

```
ALTER TABLE Employee
MODIFY (EMPNO INT NOT NULL, ENAME
        VARCHAR(50) NOT NULL, JOB
        VARCHAR(50) NOT NULL, SAL
        DECIMAL(10, 2) NOT NULL);
```

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

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION) VALUES
(4, NULL, 'Intern', NULL, NULL, NULL);
```

OUTPUT:

EMPNO	ENAME	JOB	MANAGER_NO	SAL	COMMISSION
1	John Doe	Manager	NULL	5000	1000
2	Jane Smith	Developer	1	4000	500
3	Bob Johnson	Analyst	1	3500	NULL
4	NULL	Intern	NULL	NULL	NULL

Experiment 2

2. Create a table called Employee that contain attributes EMPNO, ENAME, JOB, MGR, SAL and 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.

1. Create the Employee table with attributes EMPNO, ENAME, JOB, MGR, SAL

```
CREATE TABLE Employee (

    EMPNO INT,

    ENAME VARCHAR(50), JOB
    VARCHAR(50), MGR INT,

    SAL DECIMAL(10, 2)

);
```

2. Add a column 'commission' to the Employee

```
table ALTER TABLE Employee

ADD commission DECIMAL (10, 2);
```

3. Insert five records into the table

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MGR, SAL, commission)
VALUES
    (101, 'John Doe', 'Manager', NULL, 5000.00, 1000.00),

    (102, 'Jane Smith', 'Developer', 101, 4000.00, 500.00),

    (103, 'Bob Johnson', 'Analyst', 101, 3500.00, NULL),

    (104, 'Alice Jones', 'Designer', 101, 3800.00, 800.00),

    (105, 'Michael Brown', 'Engineer', 101, 4200.00, 700.00);
```

4. Update the column details of 'JOB'

```
ALTER TABLE Employee

MODIFY (JOB VARCHAR(50) NOT NULL);
```

5. Rename the column 'MGR' to 'MANAGER' in the Employee table

```
ALTER TABLE Employee
```

RENAME COLUMN MGR TO MANAGER;

6. Delete the employee whose Empno is 105 DELETE

FROM Employee WHERE EMPNO = 105;

Select*from Employee;

OUTPUT:

EMPNO	ENAME	JOB	MANAGER	SAL	commission
101	John Doe	Manager	NULL	5000	1000
102	Jane Smith	Developer	101	4000	500
103	Bob Johnson	Analyst	101	3500	NULL
104	Alice Jones	Designer	101	3800	800

Experiment 3

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.

1. Create the Employee table

```
CREATE TABLE Employee (E_id INT,E_name VARCHAR(50),Age INT,Salary
DECIMAL(10, 2));
```

2. Insert records into the Employee table

```
INSERT INTO Employee (E_id, E_name, Age, Salary)VALUES
(1, 'John Doe', 30, 50000.00),
  (2, 'Jane Smith', 25, 45000.00),
     (3, 'Bob Johnson', 35, 60000.00),
   (4, 'Alice Jones', 28, 52000.00),
        (5, 'Michael Brown', 32, 55000.00);
Select * from Employee;
```

3.Count number of employee names from employeetable

```
SELECT COUNT(E_name) AS num_employees FROM Employee;
```

4. Find the Maximum age from employee table

```
SELECT MAX(Age) AS max_age FROM Employee; 35
```

5. Find the Minimum age from employee table

```
SELECT MIN(Age) AS min_age FROM Employee; 25
```

6. Find salaries of employee in Ascending Order

SELECT E_name, Salary FROM Employee ORDER BY Salary ASC;

Jane Smith	45000.00
John Doe	50000.00
Alice Jones	52000.00
Michael Brown	55000.00
Bob Johnson	60000.00

7. Find grouped salaries of employees

SELECT Salary, COUNT(*) AS num_employees FROM Employee GROUP BY Salary;

50000.00	1
45000.00	1
60000.00	1
52000.00	1
55000.00	1

Experiment 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)

Create the customers table

```
            CREATE TABLE customers (
ID INT PRIMARY KEY, NAME
VARCHAR(50), AGE INT,
               ADDRESS VARCHAR(100), SALARY
               DECIMAL(10, 2)

            );

            -- Create a sequence for trigger
            CREATE SEQUENCE salary_diff_seq;

            -- Create the trigger

            CREATE OR REPLACE TRIGGER salary_diff_trigger
            AFTER INSERT OR UPDATE OR DELETE ON customers
            FOR EACH ROW

            DECLARE
               old_salary DECIMAL(10, 2);

               new_salary DECIMAL(10, 2);

               salary_diff DECIMAL(10, 2);
            BEGIN

               -- Get the old and new salary values

IF INSERTING OR UPDATING THEN
                  old_salary := NVL(:OLD.SALARY, 0);
                  new_salary := NVL(:NEW.SALARY, 0);

END IF;

               IF DELETING THEN

                  old_salary := NVL(:OLD.SALARY, 0);
                  new_salary := 0;

               END IF;

               -- Calculate the salary difference
               salary_diff := new_salary - old_salary;

               -- Display the salary difference
               DBMS_OUTPUT.PUT_LINE('Salary difference for ID ' || :OLD.ID || ': ' || salary_diff);
```

```
    -- Increment sequence
    SELECT salary_diff_seq.NEXTVAL INTO NULL FROM DUAL;
END;

/
```

Experiment 5

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

```
DECLARE
   -- Declare variables to hold values from the cursor
   v_E_id     Employee.E_id%TYPE;
   v_E_name  Employee.E_name%TYPE;
   v_Age      Employee.Age%TYPE;
   v_Salary  Employee.Salary%TYPE;

   -- Declare cursor for the Employee table
   CURSOR emp_cursor IS
      SELECT E_id, E_name, Age, Salary
      FROM Employee;
BEGIN
   -- Open the cursor
   OPEN emp_cursor;

   -- Fetch and process each row from the cursor
   LOOP
      FETCH emp_cursor INTO v_E_id, v_E_name, v_Age, v_Salary;
      EXIT WHEN emp_cursor%NOTFOUND;

      -- Process the values, for example, you can print them
      DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_E_id || ', Name: ' || v_E_name || ', Age:
' || v_Age || ', Salary: ' || v_Salary);
   END LOOP;

   -- Close the cursor
   CLOSE emp_cursor;
END;
/
```

Experiment 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.

```
create database sql6;
use sql6;

create table o_rollcall(roll_no int,name varchar(20),address varchar(20));
create table n_rollcall(roll_no int,name varchar(20),addressvarchar(20));
insert into o_rollcall values('1','Hitesh','Nandura');

insert into o_rollcall values('2','Piyush','MP');
insert into o_rollcall values('3','Ashley','Nsk');
insert into o_rollcall values('4','Kalpesh','Dhule');
insert into o_rollcall values('5','Abhi','Satara');

delimiter //

create procedure p3(in r1 int)
 begin

 declare r2 int;

 declare exit_loop boolean;

 declare c1 cursor for select roll_no from o_rollcall
where roll_no>r1;

 declare continue handler for not found set
exit_loop=true;

 open c1;
e_loop:loop
fetch c1 into r2;

if not exists(select * from n_rollcall where
roll_no=r2)

then

insert into n_rollcall select * from o_rollcall where
roll_no=r2;

end if;

 if exit_loop
then

close c1;
leave e_loop;
end if;

end loop e_loop;
end

//

call p3(3);
```

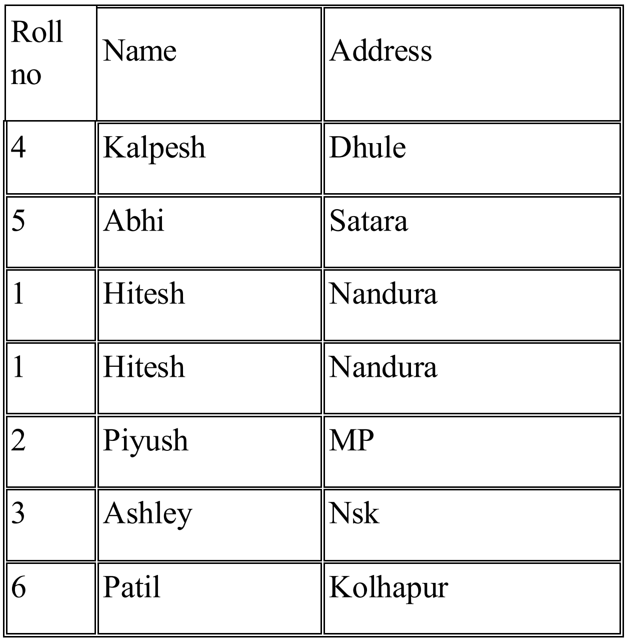
```
select * from n_rollcall;
call p3(0);

 select * from n_rollcall;

 insert into o_rollcall values('6','Patil','Kolhapur');
 call p3(4);

select * from n_rollcall;
```

OUTPUT:



Roll no	Name	Address
4	Kalpesh	Dhule
5	Abhi	Satara
1	Hitesh	Nandura
1	Hitesh	Nandura
2	Piyush	MP
3	Ashley	Nsk
6	Patil	Kolhapur

Experiment 7

7. Develop a simple web application using PHP to manage a product inventory. Connect to a MySQL database to store product information (name, price, quantity). Allow users to view the inventory, add new products, and update existing product details.

Procedure:

Step 1: Create the Database & Table

Create a database named **product_db** and a table named **products** to store product details.

SQL Commands:

sql

```
CREATE DATABASE product_db;
USE product_db;

CREATE TABLE products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    quantity INT NOT NULL
);
```

Step 2: Develop the PHP Application

1. Connecting to MySQL Database (db_connect.php)

php

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$database = "product_db";

// Create connection
```

```
$conn = new mysqli($servername, $username, $password, $database);


// Check connection

if ($conn->connect_error) {

    die("Connection failed: " . $conn->connect_error);

}

?>
```

2. Displaying Product Inventory (index.php)

php

```
<?php

include 'db_connect.php';

$result = $conn->query("SELECT * FROM products");

?>


<!DOCTYPE html>

<html>

<head>

    <title>Product Inventory</title>

</head>

<body>

    <h2>Product Inventory</h2>

    <table border="1">

        <tr>

            <th>ID</th>

            <th>Name</th>

            <th>Price</th>

            <th>Quantity</th>

        </tr>

        <?php while($row = $result->fetch_assoc()) { ?>

        <tr>

            <td><?php echo $row['id']; ?></td>

            <td><?php echo $row['name']; ?></td>

            <td><?php echo $row['price']; ?></td>
            <td><?php echo $row['quantity']; ?></td>
        </tr>
        <?php } ?>
    </table>
</body>
</html>
```

3. Adding a New Product (add_product.php)

php

```
<?php
include 'db_connect.php';

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST['name'];
    $price = $_POST['price'];
    $quantity = $_POST['quantity'];

    $sql = "INSERT INTO products (name, price, quantity) VALUES ('$name', '$price', '$quantity')";

    if ($conn->query($sql) === TRUE) {
        echo "New product added successfully!";
    } else {
        echo "Error: " . $conn->error;
    }
}
?>

<form method="post">
    Name: <input type="text" name="name" required><br>
    Price: <input type="number" name="price" required><br>
    Quantity: <input type="number" name="quantity" required><br>
    <input type="submit" value="Add Product">
</form>
```

4. Updating Product Details (update_product.php)

php

```
<?php
include 'db_connect.php';

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $id = $_POST['id'];
    $price = $_POST['price'];
    $quantity = $_POST['quantity'];

    $sql = "UPDATE products SET price='$price', quantity='$quantity' WHERE id='$id'";

    if ($conn->query($sql) === TRUE) {
        echo "Product updated successfully!";
    } else {
        echo "Error: " . $conn->error;
    }
}
?>

<form method="post">
    Product ID: <input type="number" name="id" required><br>
    New Price: <input type="number" name="price" required><br>
    New Quantity: <input type="number" name="quantity" required><br>
    <input type="submit" value="Update Product">
</form>
```

Execution Steps:

1. Install **XAMPP** and start **Apache & MySQL** services.
2. Create the database **product_db** and **execute the SQL commands** in **phpMyAdmin** or **MySQL Workbench**.
3. Save the PHP files in the **htdocs** folder (C:\xampp\htdocs\inventory\).
4. Open the browser and visit:
    - ○ http://localhost/inventory/index.php → View Inventory
    - ○ http://localhost/inventory/add_product.php → Add Product

○ http://localhost/inventory/update_product.php → Update Product

Output:

1. Product Inventory Table (index.php)

ID	Name	Price	Quantity
1	Pen	10.00	100
2	Book	50.00	30

2. Add Product Form (add_product.php)
Input: Pen, 10, 100
Output: "New product added successfully!"

3. Update Product Form (update_product.php)
Input: ID: 1, Price: 15, Quantity: 120
Output: "Product updated successfully!"

Experiment 8

Step 1: Create a Sample Database & Table

Create a database concurrency_db and a table accounts to simulate concurrent access.

SQL Commands:

```
CREATE DATABASE concurrency_db;

USE concurrency_db;


CREATE TABLE accounts (

    account_id INT PRIMARY KEY,

    account_holder VARCHAR(50),

    balance DECIMAL(10,2)

);
```

Step 2: Insert Sample Data

```
INSERT INTO accounts VALUES (1, 'Alice', 5000.00);

INSERT INTO accounts VALUES (2, 'Bob', 7000.00);
```

Scenario 1: Simulating Concurrency Issues (Without Locking)

Two users trying to withdraw money from the same account at the same time

Transaction 1 (User 1):

```
START TRANSACTION;

SELECT balance FROM accounts WHERE account_id = 1; -- Returns 5000

UPDATE accounts SET balance = balance - 1000 WHERE account_id = 1;

COMMIT;
```

Transaction 2 (User 2):

```
START TRANSACTION;

SELECT balance FROM accounts WHERE account_id = 1; -- Also sees 5000

UPDATE accounts SET balance = balance - 2000 WHERE account_id = 1;

COMMIT;
```

Problem:

- Both users see 5000 balance before updating.
- The second transaction does not consider the first withdrawal, causing data inconsistency.

Step 3: Apply Table-Level Locking

Ensuring only one user modifies data at a time

sql

```
LOCK TABLE accounts WRITE;
UPDATE accounts SET balance = balance - 1000 WHERE account_id = 1;
UNLOCK TABLES;
```

Execution Steps:

1. User 1 executes the transaction → Table is locked.
2. User 2 tries to execute → Must wait until the first transaction is complete.
3. Ensures data consistency by preventing simultaneous updates.

Scenario 3: Using Row-Level Locking for Better Concurrency

Step 4: Implementing Row Locks (Using SELECT ... FOR UPDATE)

Lock only the row being updated instead of the whole table

User 1 Transaction (Withdraw 1000 from Alice's account):

```
START TRANSACTION;
SELECT balance FROM accounts WHERE account_id = 1 FOR UPDATE;
UPDATE accounts SET balance = balance - 1000 WHERE account_id = 1;
COMMIT;
```

- The FOR UPDATE statement locks only the selected row.
- Other transactions must wait before accessing the row.

Scenario 4: Implementing Timestamp-Based Concurrency Control

Step 5: Adding a Timestamp Column

Detect if another transaction modified the data before commit

Modify Table:

```
ALTER TABLE accounts ADD COLUMN last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP;
```

Transaction 1 (User 1):

```
SELECT balance, last_updated FROM accounts WHERE account_id = 1;
UPDATE accounts SET balance = balance - 1000 WHERE account_id = 1
AND last_updated = '2025-02-17 10:30:00'; -- Use actual timestamp from SELECT query
```

Why is this useful?

Prevents lost updates by ensuring that no other transaction changed the data before committing.

Execution Steps:

1. Create the database and accounts table.
2. Insert sample records into the table.
3. Simulate concurrent access issues (without locking).
4. Apply table locks and row locks to prevent conflicts.
5. Use timestamps to detect and prevent overwrites.
6. Test all queries using multiple users or transactions.

Expected Output:

- Without locking: Incorrect balance updates due to race conditions.
- With locking: Only one transaction modifies the record at a time.
- With timestamps: Prevents overwrites by detecting modifications.

Transaction	Initial Balance	Operation	New Balance	Concurrency Control Used?
User 1	5000	Withdraw 1000	4000	✕ No Lock – Data Loss
User 2	5000	Withdraw 2000	3000	✕ No Lock – Data Loss
User 1	5000	Withdraw 1000	4000	☑ Table Lock Used
User 2	4000	Withdraw 2000	2000	☑ Ensured Consistency
User 1	5000	Withdraw 1000	4000	☑ Row Lock Used

User 2	4000	Withdraw 2000	2000	✅ Timestamp Check Passed

Experiment No: 9

Create a database for a Library Management System. Include tables for books, authors, and members. Use appropriate data types and constraints (primary key, foreign key, etc.)

Procedure:

Step 1: Create the Database & Tables

We need three tables: Books, Authors, and Members

SQL Commands:

```
sql


CREATE DATABASE LibraryDB;

USE LibraryDB;


-- Table for Authors

CREATE TABLE Authors (

   author_id INT AUTO_INCREMENT PRIMARY KEY,

   name VARCHAR(100) NOT NULL

);


-- Table for Books

CREATE TABLE Books (

   book_id INT AUTO_INCREMENT PRIMARY KEY,

   title VARCHAR(255) NOT NULL,

   author_id INT,

   category VARCHAR(100),

   available BOOLEAN DEFAULT TRUE,

   FOREIGN KEY (author_id) REFERENCES Authors(author_id)

);


-- Table for Members

CREATE TABLE Members (

   member_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    membership_date DATE
);

-- Table for Borrowed Books
CREATE TABLE BorrowedBooks (
    borrow_id INT AUTO_INCREMENT PRIMARY KEY,
    member_id INT,
    book_id INT,
    borrow_date DATE,
    return_date DATE,
    FOREIGN KEY (member_id) REFERENCES Members(member_id),
    FOREIGN KEY (book_id) REFERENCES Books(book_id)
);
```

Step 2: Insert Sample Data

SQL Commands:

```
-- Insert Authors
INSERT INTO Authors (name) VALUES ('J.K. Rowling'), ('George Orwell'), ('Agatha Christie');

-- Insert Books
INSERT INTO Books (title, author_id, category) VALUES
('Harry Potter', 1, 'Fantasy'),
('1984', 2, 'Dystopian'),
('Murder on the Orient Express', 3, 'Mystery'),
('Animal Farm', 2, 'Political Satire'),
('The Casual Vacancy', 1, 'Drama');

-- Insert Members
INSERT INTO Members (name, membership_date) VALUES
('Alice', '2024-01-10'),
('Bob', '2023-12-15');

-- Insert Borrowed Books
```

```
INSERT INTO BorrowedBooks (member_id, book_id, borrow_date, return_date) VALUES

(1, 1, '2024-02-01', NULL), -- Book not returned

(2, 2, '2024-01-15', '2024-02-10'); -- Book returned
```

Step 3: Execute Queries

1. List all books by a specific author

Find all books written by 'George Orwell'

```
SELECT Books.title

FROM Books

JOIN Authors ON Books.author_id = Authors.author_id

WHERE Authors.name = 'George Orwell';
```

Execution Steps:

1. **Create the database & tables** using MySQL Workbench.
2. **Insert sample data** into all tables.
3. **Run queries** to retrieve books by author, books borrowed but not returned, and total books per category.
4. **Verify outputs** and discuss real-world applications of the system.

Experiment No: 10

Analyze a sample database schema provided by your instructor. Identify any normalization anomalies (redundancy, inconsistency). Apply normalization techniques (**1NF, 2NF, 3NF, BCNF**) to bring the schema to a higher normal form, minimizing data redundancy.

Software & Tools Required:

MySQL Workbench (For SQL Query Execution)

DBVisualizer or DBeaver (For Schema Visualization & Analysis)

Procedure:

Step 1: Sample Unnormalized Database Schema (UNF - Unnormalized Form)

Consider a sample database schema that stores student enrollment details in a college:

Student_ID	Student_Name	Course_ID	Course_Name	Instructor	Instructor_Phone
101	Alice	CSE101	DBMS	Prof. John	9876543210
101	Alice	CSE102	CN	Prof. Smith	9123456789
102	Bob	CSE101	DBMS	Prof. John	9876543210
103	Charlie	CSE103	AI	Prof. Emma	9345678123

Problems in Unnormalized Form (UNF):

- **Data Redundancy:** Instructor details are repeated multiple times.
- **Update Anomalies:** If Prof. John changes the phone number, it needs to be updated in multiple rows.
- **Insertion Anomalies:** A new instructor **cannot be added** unless assigned to a course.
- **Deletion Anomalies:** Deleting a course also removes instructor details.

Step 2: Applying First Normal Form (1NF)

1NF Rule: Each column must have atomic values (no repeating groups or arrays).

Revised Schema (1NF)

Student_ID	Student_Name	Course_ID	Course_Name	Instructor_ID	Instructor_Name	Instructor_Phone
101	Alice	CSE101	DBMS	I001	Prof. John	9876543210
101	Alice	CSE102	CN	I002	Prof. Smith	9123456789
102	Bob	CSE101	DBMS	I001	Prof. John	9876543210
103	Charlie	CSE103	AI	I003	Prof. Emma	9345678123

Issues Remaining:

- **Course details depend only on Course_ID, not Student_ID** (violates 2NF).
- **Instructor details depend on Instructor_ID, not Course_ID** (violates 2NF).

Step 3: Applying Second Normal Form (2NF)

2NF Rule: Remove Partial Dependency (A non-key column should depend on the whole primary key, not part of it).

Revised Schema (2NF)

- **Students Table** *(Stores student details)*

```
CREATE TABLE Students (
    Student_ID INT PRIMARY KEY,
    Student_Name VARCHAR(100)
);
```

Student_ID	Student_Name
101	Alice
102	Bob
103	Charlie

Courses Table *(Stores course details)*

```
CREATE TABLE Courses (
    Course_ID VARCHAR(10) PRIMARY KEY,
    Course_Name VARCHAR(100),
    Instructor_ID VARCHAR(10),
    FOREIGN KEY (Instructor_ID) REFERENCES Instructors(Instructor_ID)
);
```

Course_ID	Course_Name	Instructor_ID
CSE101	DBMS	I001
CSE102	CN	I002
CSE103	AI	I003

Instructors Table *(Stores instructor details separately)*

```
CREATE TABLE Instructors (
    Instructor_ID VARCHAR(10) PRIMARY KEY,
    Instructor_Name VARCHAR(100),
    Instructor_Phone VARCHAR(15)
);
```

Instructor_ID	Instructor_Name	Instructor_Phone
I001	Prof. John	9876543210
I002	Prof. Smith	9123456789
I003	Prof. Emma	9345678123

Enrollments Table *(Links students to courses - Many-to-Many Relationship)*

```
sql

CREATE TABLE Enrollments (
    Student_ID INT,
    Course_ID VARCHAR(10),
    PRIMARY KEY (Student_ID, Course_ID),
    FOREIGN KEY (Student_ID) REFERENCES Students(Student_ID),
    FOREIGN KEY (Course_ID) REFERENCES Courses(Course_ID)
);
```

Student_ID	Course_ID
101	CSE101
101	CSE102
102	CSE101
103	CSE103

Issue Fixed: No partial dependencies!

Step 4: Applying Third Normal Form (3NF)

3NF Rule: Remove Transitive Dependencies (Non-key column should not depend on another non-key column).

Already in 3NF:

- **Each non-key column depends only on the primary key** (No instructor-phone dependency on Courses).

Final Fix: Everything is now **properly normalized!**

Step 5: Applying Boyce-Codd Normal Form (BCNF)

BCNF Rule: Every determinant must be a candidate key.

- **No violations exist in our final schema.**
- **Tables are now fully normalized!**

Execution Steps:

1. Create **unnormalized schema** and insert sample data.
2. Apply **1NF, 2NF, 3NF, BCNF** and modify tables.
3. Execute **SQL queries** to test the new schema.