Vidyavardhini's
College of Engineering & Technology
Vasai Road (W)

Department of Artificial Intelligence and Data Science

Lab Manual

Semester	IV	Class	S.E
Course Code	CSL402	Academic Year	2021-22
Course Name	Database Management System		

Vidyavardhini's College of Engineering & Technology

Vision

To be a premier institution of technical education; always aiming at becoming a valuable resource for industry and society.

Mission

- To provide a technologically inspiring environment for learning.
- To promote creativity, innovation and professional activities.
- To inculcate ethical and moral values.
- To cater personal, professional and societal needs through quality education.

Department Vision:

To foster proficient artificial intelligence and data science professionals, making remarkable contributions to industry and society.

Department Mission:

- To encourage innovation and creativity with rational thinking for solving the challenges in emerging areas.
- To inculcate standard industrial practices and security norms while dealing with Data.
- To develop sustainable Artificial Intelligence systems for the benefit of various sectors.

Program Specific Outcomes (PSOs):

PSO1: Analyze the current trends in the field of Artificial Intelligence & Data Science and convey their findings by presenting / publishing at a national / international forum.

PSO2: Design and develop Artificial Intelligence & Data Science based solutions and applications for problems in the different domains catering to industry and society.

Program Outcomes (POs):

Engineering Graduates will be able to:

- **PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **PO2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- **PO3. 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.
- **PO4. 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.

- **PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **PO6. 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.
- **PO7. 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.
- **PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO9. Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **PO10. 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.
- **PO11. 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.
- **PO12. 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.

Course Objectives

1	To Develop Entity Relationship data model.
2	To develop relational Model
3	To formulate SQL queries.
4	To learn procedural interfaces to SQL queries
5	To learn the concepts of transactions and transaction processing
6	To understand how to handle concurrent transactions and able to access data through front end (using JDBC ODBC connectivity)

Course Outcomes

At the end of the course student will be able to:		Action verb	Bloom Level
CSL402.1	Design ER and EER diagrams for real life problems with software tools.	Design	Create (Level 6)
CSL402.2	Construct database tables with different DDL and DML statements and apply integrity constraints	Apply	Apply (Level 3)
CSL402.3	Apply SQL queries ,triggers for given Schema	Apply	Apply (Level 3)
CSL402.4	Apply procedure and functions for given schema	Apply	Apply (Level 3)
CSL402.5	Design ER and EER diagrams for the real life problem with software tool.	Use	Apply (Level 3)
CSL402.6	Construct database tables with different DDL and DML statements and apply integrity constraints	Construct	Apply(Level 3)

List of Experiments

Sr. No	Name of Experiments	Mode of conduction
1	Identify the case study and detailed statement of the problem. Design an Entity Relationship (ER) / Extended Entity Relationship (EER) Model.	2
2	Mapping ER/EER to Relational schema model.	2
3	Create a database using Data Definition Language (DDL) and apply integrity constraints for the specified System	2
4	Apply DML Commands for the specified system	2
5	Perform Simple queries, string manipulation operations and aggregate functions.	2
6	Implement various Join operations.	2
7	Perform DCL and TCL commands	2
8	Implementation of Views and Triggers.	2
9	Demonstrate Database connectivity	2
10	Implementation and demonstration of Transaction and Concurrency control techniques using locks	2

Mapping of Experiments with Course Outcomes

Course Modules	Course Outcomes					
	CSL402.1	CSL402.2	CSL402.3	CSL402.4	CSL402.5	CSL402.6
Identify the case study and detailed statement of the problem. Design an Entity Relationship (ER) / Extended Entity Relationship (EER) Model.	3					
Mapping ER/EER to Relational schema model.	3					
Create a database using Data Definition Language (DDL) and apply integrity constraints for the specified System		3				
Apply DML Commands for the specified system		3				
Perform Simple queries, string manipulation operations and aggregate functions.			3			
Implement various Join operations.				3		

Perform DCL and TCL commands				3		
Implementation of Views and Triggers.					3	
Demonstrate Database connectivity						3
Implementation and demonstration of Transaction and Concurrency control techniques using locks						3

Enter correlation level 1, 2 or 3 as defined below
1: Slight (Low) 2: Moderate (Medium) 3: Substatial (High)
If there is no correlation put "—".

Experiment No.1
Design an EntityRelationship (ER) / Extended Entity-Relationship (EER) Model.
Date of Performance:
Date of Submission:

Aim :- Identify the case study and detailed statement of the problem. Design an EntityRelationship (ER) / Extended Entity-Relationship (EER) Model.

Objective :- To identify and explore a real world problem, and to design an Entity Relationship (ER) / Extended Entity-Relationship (EER) Model.

Theory:

1. **Entity:**

- An entity is a real-world object or concept that exists independently and has distinguishable attributes.
- In a database context, an entity represents a table, and each row in that table represents a unique instance of that entity.
- For example, in a university database, entities could include Student, Course, Professor, Department, etc.
- Each entity has a set of attributes that describe its properties.

2. **Attributes:**

- Attributes are the properties or characteristics that describe an entity.
- They represent the data we want to store about each instance of an entity.
- For example, attributes of a Student entity might include StudentID, Name, Age, GPA, etc.
- Attributes can be categorized as simple (atomic) attributes, which cannot be divided further, or composite attributes, which are made up of smaller sub-parts.

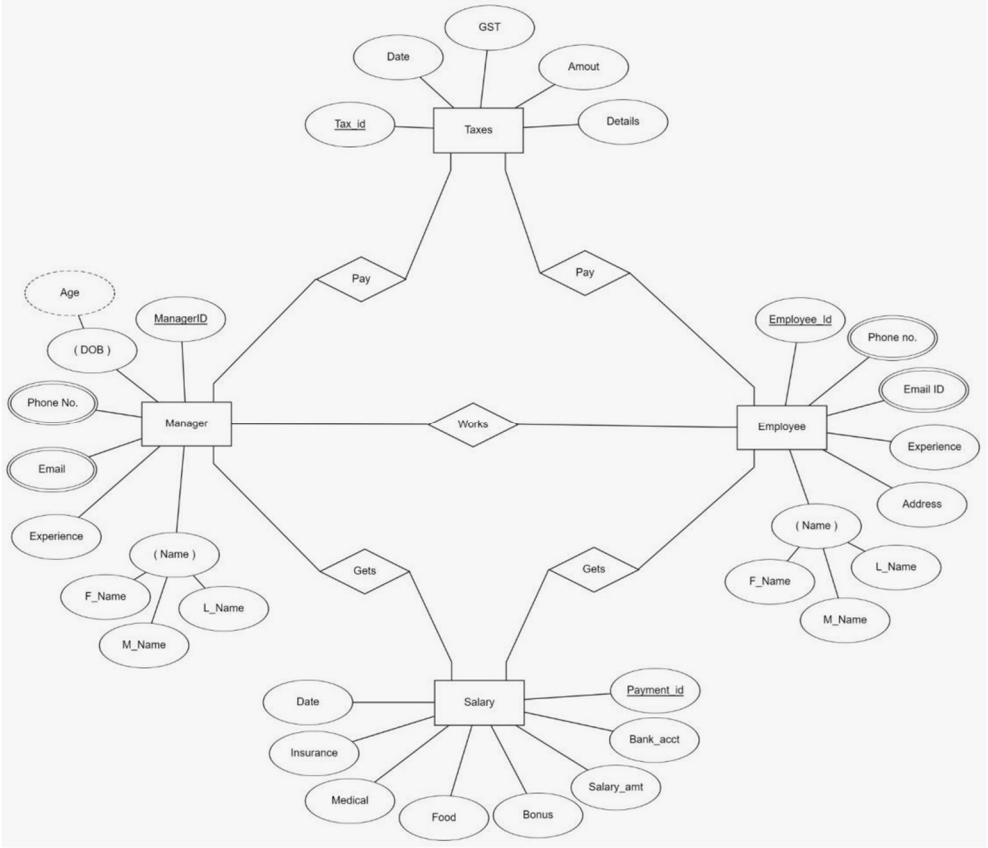
3. **Relationships:**

- Relationships describe how entities are related to each other or how they interact.
- They represent the associations between entities.
- Relationships are depicted as lines connecting related entities in the ER diagram.
- Each relationship has a degree, indicating the number of entities involved. It could be unary (involving one entity), binary (involving two entities), or ternary (involving three entities).
- Relationships also have cardinality, which defines the number of instances of one entity that can be associated with the number of instances of another entity through the relationship.

4. Cardinality:

- Cardinality specifies the number of instances of one entity that are related to the number of instances of another entity through a relationship.
- It defines the maximum and minimum number of occurrences of one entity that can be associated with the occurrences of another entity.
- Common cardinality constraints include:
  - I. One-to-One (1:1): Each instance of one entity is associated with exactly one instance of another entity, and vice versa.
  - II. One-to-Many (1:N): Each instance of one entity is associated with zero or more instances of another entity, but each instance of the second entity is associated with exactly one instance of the first entity.
  - III. Many-to-One (N:1): The reverse of One-to-Many; many instances of one entity are associated with one instance of another entity.
  - IV. Many-to-Many (N:N): Many instances of one entity can be associated with many instances of another entity.

Implementation:



Conclusion:

1. Define Entity, Attributes(also types) and Relationship between entities

Ans

1. Entity:
   - An entity is a distinct object or concept in a domain, about which data is stored.
   - It can be a person, place, thing, event, or concept that is distinguishable from other entities.
   - Entities are typically nouns and serve as the basis for data modeling in databases.
2. Attributes:
   - Attributes are the properties or characteristics of an entity.
   - They describe the entity and provide details about it.
   - Types of attributes include:
     - Simple attributes: Represent atomic values, like a person's age or a product's price.
     - Composite attributes: Composed of multiple simpler attributes, like a person's address (street, city, zip code).
     - Derived attributes: Derived from other attributes, rather than being directly stored, like a person's age calculated from their birthdate.
     - Multi-valued attributes: Can hold multiple values for a single entity, like a person's phone numbers.
     - Key attributes: Uniquely identify an entity within a set, like a person's social security number
3. Relationships:
   - Relationships describe how entities interact or associate with each other.
   - They represent connections between entities.
   - Types of relationships include:
     - One-to-One: Each entity in one set is associated with exactly one entity in another set.
     - One-to-Many: Each entity in one set can be associated with multiple entities in another set.
     - Many-to-Many: Entities in one set can be associated with multiple entities in another set, and vice versa.
     - Recursive: An entity is related to itself.
     - Unary: An entity is related to itself in a unary relationship.
     - Binary: Two entities are involved in the relationship.
     - Ternary: Three entities are involved in the relationship.

2. Write ER/EER diagram notations

Ans

a) Entities: Represented by rectangles, each entity denotes a distinct object, such as a person or a product.

b) Attributes: Shown inside the entity rectangles, attributes describe properties of entities, like name or price.

c) Relationships: Illustrated by lines connecting entities, relationships indicate connections between entities, such as a person buying a product.

d) Cardinality: Depicted near the relationship lines, cardinality defines the number of instances of one entity that can be associated with another entity.

e) Keys: Highlighted using underlines, keys uniquely identify instances of an entity, such as a primary key.

Experiment No.2
Mapping ER/EER to Relational schema model.
Date of Performance:
Date of Submission:

Aim :- Prepare the schema for Relational Model with the ER/ERR diagram, drawn for the identified case study in experiment no.1.

Objective :- To map the Entity Relationship (ER) / Extended Entity-Relationship (EER) Diagram to Relational Model schema and learn to incorporate various schema-based constraints.

Theory:

Mapping an Entity-Relationship (ER) model to a relational database schema involves translating the conceptual model represented in the ER diagram into tables and relationships in a relational database management system (DBMS). Here are the general rules for mapping ER to a schema in a DBMS:

1. Entities to Tables:
    a. Each entity in the ER diagram corresponds to a table in the relational schema.
    b. The attributes of the entity become the columns of the table.
    c. The primary key of the entity becomes the primary key of the table.

2. Relationships to Tables:
    a. Many-to-Many Relationships:
        i. Convert each many-to-many relationship into a new table.
        ii. Include foreign key columns in this table to reference the participating entities.
        iii. The primary key of this table may consist of a combination of the foreign keys from the participating entities.
    b. One-to-Many and One-to-One Relationships:
        i. Represented by foreign key columns in one of the participating tables.
        ii. The table on the "many" side of the relationship includes the foreign key column referencing the table on the "one" side.
        iii. The foreign key column typically references the primary key of the related table.

3. Attributes to Columns:
    a. Each attribute of an entity becomes a column in the corresponding table.
    b. Choose appropriate data types for each attribute based on its domain and constraints.
    c. Ensure that attributes participating in relationships are represented as foreign keys when needed.

4. Primary and Foreign Keys:
    a. Identify the primary key(s) of each table based on the primary key(s) of the corresponding entity.
    b. Ensure referential integrity by defining foreign keys in tables to establish relationships between them.
    c. Foreign keys should reference the primary key(s) of related tables.

d. Ensure that foreign keys have appropriate constraints, such as ON DELETE CASCADE or ON UPDATE CASCADE, to maintain data integrity.

5. Cardinality Constraints:
   a. Use the cardinality constraints from the ER diagram to determine the multiplicity of relationships in the relational schema.
   b. Ensure that the constraints are enforced through the appropriate use of primary and foreign keys.

6. Normalization:
   a. Normalize the schema to minimize redundancy and dependency.
   b. Follow normalization rules such as First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), etc., to ensure data integrity and minimize anomalies.

7. Indexing and Optimization:
   a. Consider indexing frequently queried columns to improve query performance.
   b. Evaluate the schema design for optimization opportunities based on query patterns and performance requirements.

Implementation:

Conclusion:

1. write definition of relational schema and notations

Ans Schema Definition:

A relational schema is a blueprint or structure that represents the logical arrangement of data elements (tables) and the relationships between them in a relational database. It outlines the tables, their

attributes, and the constraints that govern their relationships. The schema provides a formal description of how the database is organized, facilitating data management, querying, and manipulation.

Relational Schema Notations:

1. Tables: Represented by rectangles, each table corresponds to an entity or relation in the database.

2. Attributes: Shown inside the table rectangles, attributes describe the properties or characteristics of the entities. Each attribute has a name and a data type.

3. Primary Key: Identified by underlining or highlighting, the primary key uniquely identifies each record within a table. It ensures data integrity and serves as a unique identifier.

4. Foreign Key: Denoted by an attribute in one table that references the primary key of another table. It establishes relationships between tables, enforcing referential integrity.

5. Relationships: Illustrated by lines connecting tables, relationships indicate how tables are related to each other. These lines typically represent one-to-one, one-to-many, or many-to-many relationships.

2. write various schema-based constraints

Ans
1. Primary Key Constraint: Enforces unique identification for rows in a table.
2. Foreign Key Constraint: Maintains relationships between tables to ensure referential integrity.
3. Unique Constraint: Ensures uniqueness of values in specified columns.
4. Check Constraint: Imposes conditions on allowable values in a column.
5. Not Null Constraint: Requires a column to have a value, prohibiting null entries.
6. Default Constraint: Provides a default value when none is specified.
7. Referential Integrity Constraint: Ensures validity of relationships between tables.

Experiment No.3
Create a database using Data Definition Language(DDL) and apply integrity constraints for the specified system
Date of Performance:
Date of Submission:

Aim:- Write a query to create tables for each relation in the relational schema of experiment no.2. Apply drop and alter commands on those tables.

Objective:- To learn commands of Data Definition Language(DDL) to create and define databases, and also learn to apply integrity constraints for the specified system.

Theory:

DDL Commands & Syntax:-

Data Definition Language (DDL) is a subset of SQL and a part of DBMS(Database Management System). DDL consist of Commands to commands like CREATE, ALTER, TRUNCATE and DROP. These commands are used to create or modify the tables in SQL.

DDL Commands:

1. Create
2. Alter
3. truncate
4. drop
5. Rename

CREATE:

This command is used to create a new table in SQL. The user must give information like table name, column names, and their data types.

Syntax –CREATE TABLE table_name

(

column_1 datatype,

column_2 datatype,

column_3 datatype,

....

);

ALTER :

This command is used to add, delete or change columns in the existing table. The user needs to know the existing table name and can add, delete, or modify tasks easily.

Syntax –

ALTER TABLE table_name

ADD column_name datatype;

TRUNCATE :

This command is used to remove all rows from the table, but the structure of the table still exists.

Syntax –

TRUNCATE TABLE table_name;

DROP :

This command is used to remove an existing table along with its structure from the Database.

Syntax –

DROP TABLE table_name;

RENAME :

It is possible to change name of table with or without data in it using simple RENAME command. We can rename any table object at any point of time.

Syntax –

RENAME TABLE <Table Name> To <New_Table_Name>;

Implementation:

1. Create :

```
CREATE DATABASE PayrollManagement;
USE PayrollManagement;

CREATE TABLE Employee (
   EmployeeID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   Salary DECIMAL(10, 2),
   TaxRate DECIMAL(5, 4),
   ManagerID INT,
```

FOREIGN KEY (ManagerID) REFERENCES Manager(ManagerID)
);

```
1  CREATE DATABASE PayrollManagement;
2  USE PayrollManagement;
4  CREATE TABLE Employee (
5      MANAGERID INT PRIMARY KEY,
6      FirstName VARCHAR(50),
7      LastName VARCHAR(50),
8      Salary DECIMAL(10, 2),
9      TaxRate DECIMAL(5, 4)
10 );
```

2. Alter:
ALTER TABLE Employee
ADD CONSTRAINT chk_TaxRate CHECK (TaxRate >= 0 AND TaxRate <= 1);

```
11
12  ALTER TABLE Employee
13  ADD CONSTRAINT chk_TaxRate CHECK (TaxRate >= 0 AND TaxRate <= 1);
14
```

3. Trucncate:
TRUNCATE TABLE Employee;

```
14
15  TRUNCATE TABLE Employee;
```

4. Drop:
DROP TABLE Manager;

```
16
17  DROP TABLE Manager;
18
```

5. Rename:
RENAME TABLE Employee TO PayrollEmployees;

```
18
19  RENAME TABLE Employee TO PayrollEmployees;
```

Conclusion:

1. Explain the concept of constraints in DDL. How are constraints used to enforce data integrity?

Ans

1. Constraints in DDL (Data Definition Language) are rules or conditions applied to the database schema to enforce data integrity.

2. They ensure that data stored in tables adheres to predefined rules, preventing inconsistencies or errors.

3. Constraints are used to specify requirements for data values, such as uniqueness, validity, or relationships between tables.

4. By enforcing constraints, DDL ensures the accuracy, consistency, and reliability of data stored in the database.

5. Common constraints include primary key, foreign key, unique, check, not null, default, and referential integrity constraints.

6. Primary key constraints uniquely identify each record in a table, while foreign key constraints maintain relationships between tables.

7. Unique constraints enforce uniqueness of values, check constraints validate data based on specified conditions, and not null constraints require values to be present.

8. Default constraints provide default values when none are specified, and referential integrity constraints ensure the consistency of relationships between tables.

2. What is the significance of data types in DDL? Provide examples of commonly used data types in DDL.

Ans

1. Data Type Significance:
   - Data types in Data Definition Language (DDL) specify the type of data that can be stored in a database column, ensuring data integrity and optimal storage efficiency.
   - They enforce constraints on the values that can be inserted into a column, preventing data inconsistencies and errors.

2. Examples of Commonly Used Data Types:
   - Integer: Stores whole numbers, such as age or quantity.
   - Varchar: Variable-length character string, suitable for storing text data like names or addresses.
   - Decimal: Holds fixed-point numbers with decimal precision, commonly used for monetary values.
   - Date: Stores date values, facilitating date-related operations and queries.
   - Boolean: Represents true/false or yes/no values, useful for logical operations and conditions.
   - Blob: Binary large object, used for storing large binary data like images or documents.

Experiment No.4
Apply DML commands for the specified system
Date of Performance:
Date of Submission:

Aim :- Write insert query to insert rows for each table created of your database management system. Use update and delete commands to manipulate the inserted values in the table.

Objective :- To learn commands of Data Manipulation Language(DML) to insert, update or delete the values in the database system.

Theory:

Data Manipulation Language (DML) is a subset of SQL (Structured Query Language) used for managing data within relational database management systems (RDBMS). DML commands are used to perform operations such as inserting, updating, and deleting data from database tables.

1. Inserting Data

The INSERT statement is used to add new rows of data into a table. It specifies the table to insert data into and provides values or expressions for each column in the new row. If a column list is not specified, values must be provided for all columns in the table in the order they were defined.

Syntax:-

INSERT INTO table_name (column1, column2, column3) VALUES (value1, value2, value3);

2. Updating Data

The UPDATE statement is used to modify existing data within a table. It allows you to change the values of one or more columns in one or more rows based on specified conditions. If no condition is specified, all rows in the table will be updated.

Syntax:

UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;

3. Deleting Data

The DELETE statement is used to remove one or more rows from a table based on specified conditions. If no condition is specified, all rows in the table will be deleted.

Syntax:

DELETE FROM table_name WHERE condition;

Implementation:

INSERT INTO Employee (EmployeeID , Firstname , Lastname , Salary , TaxRate , ManagerID)
VALUES (201, 'John, 'Doe', 50000.00, 0.15, 101);

```
40 •    INSERT INTO Employee (EmployeeID, FirstName, LastName, Salary, TaxRate, ManagerID)
41      VALUES
42          (201, 'John', 'Doe', 50000.00, 0.15, 101),
43          (202, 'Alice', 'Smith', 60000.00, 0.12, 102),
44          (203, 'Michael', 'Johnson', 70000.00, 0.18, 103);
```

UPDATE Employee

SET salary = 55000
WHERE employee_id = 1;

DELETE FROM Employee
WHERE employee_id = 1;

INSERT INTO Manager (ManagerID , FirstName , LastName , Salary , TaxRate)
VALUES (101, 'Emily', 'Davis', 80000.00, 0.20);

```
33 •    INSERT INTO Manager (ManagerID, FirstName, LastName, Salary, TaxRate)
34      VALUES
35          (101, 'Emily', 'Davis', 80000.00, 0.20),
36          (102, 'David', 'Wilson', 90000.00, 0.17),
37          (103, 'Emma', 'Taylor', 100000.00, 0.22);
```

UPDATE Manager
SET department_id = 102
WHERE manager_id = 101;

DELETE FROM Manager
WHERE manager_id = 101;

Conclusion:

1. Explain the role of database constraints in enforcing data integrity during DML operations.

Ans Database constraints play a crucial role in maintaining data integrity during Data Manipulation Language (DML) operations:

1. Entity Integrity: Primary key constraints ensure each row in a table is uniquely identified, preventing duplicates.

2. Referential Integrity: Foreign key constraints establish relationships between tables, enforcing consistency across related data.

3. Domain Integrity: Check constraints enforce rules on column values, such as data type or range restrictions, ensuring valid data entry.

4. Null Integrity: Not null constraints mandate that a column must have a value, preventing null entries where inappropriate.

5. User-defined Integrity: Custom constraints allow businesses to enforce specific rules tailored to their needs, further enhancing data integrity. By enforcing these constraints, the database ensures that only valid, consistent, and accurate data is stored, maintaining data integrity across all DML operations.

2. How do you update multiple columns in a table using a single UPDATE statement?

Ans

To update multiple columns in a table with one SQL statement:
- Use the UPDATE statement.
- Specify the table name.
- Set each column with its new value using SET.
- Add a WHERE clause to specify the condition for which rows to update.
- This allows updating specific columns in specific rows efficiently.

Experiment No.5
Perform simple queries, string manipulation operations and aggregate functions.
Date of Performance:
Date of Submission:

Aim :- Write simple query to manipulate string operations and perform aggregate functions like (MIN, MAX, SUM, AVERAGE, COUNT).

Objective :- To apply aggregate functions and string manipulation functions to perform simple queries in the database system

Theory:

Simple Queries in SQL:

In SQL, a simple query is a request for data from a database table or tables. It allows users to retrieve specific information by specifying the columns they want to retrieve and any conditions for filtering rows based on certain criteria. Simple queries are the backbone of interacting with databases, enabling users to extract the data they need for analysis, reporting, or further processing.

String Manipulation Operations:

String manipulation operations in SQL involve modifying or transforming string values stored in database columns. These operations are crucial for tasks such as formatting data, combining strings, converting case, or extracting substrings. By using string functions and operators, users can manipulate text data to suit their requirements, whether it's for display purposes or for further analysis.

Aggregate Functions:

Aggregate functions in SQL are used to perform calculations on sets of values and return a single result. These functions allow users to summarize data across multiple rows, providing insights into the overall characteristics of the dataset. Common aggregate functions include calculating counts, sums, averages, minimums, and maximums of numerical values. They are essential tools for data analysis, enabling users to derive meaningful insights from large datasets.

Benefits of Understanding These Concepts:

- Data Retrieval: Simple queries allow users to fetch specific data from databases, facilitating data retrieval for various purposes.
- Data Transformation: String manipulation operations enable users to format and transform text data according to their needs, improving data consistency and readability.
- Data Analysis: Aggregate functions help users summarize and analyze large datasets, providing valuable insights into trends, patterns, and statistical measures.
- Data Reporting: By combining simple queries, string manipulation operations, and aggregate functions, users can generate reports and visualizations that communicate key findings effectively.

Implementation:

```
use company1;

Insert into manager(name1,Experience,Phone_No,Salary,ManagerID)

Values('hea',7,3456767,9800,609);
```

1)Upper

```
Select upper('sir') from manager;
```

Result Grid | Filter Ro

upper('sir')
SIR
SIR
SIR
SIR
SIR

2)Lower

```
Select lower('SIR') from manager;
```

Result Grid | Filte

lower('SIR')
sir
sir
sir
sir
sir

3)Trim

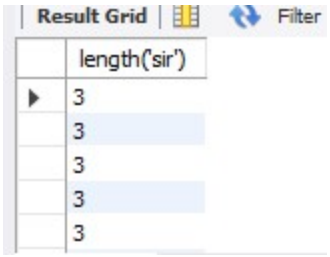
```
Select trim(' sir ') from manager;
```

Result Grid | F

trim(' sir ')
sir
sir
sir
sir

4)length

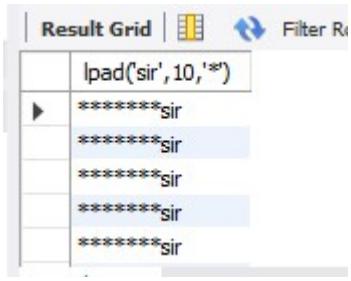
```
Select length('sir') from manager;
```



	length('sir')
▶	3
	3
	3
	3
	3

5)Lpad

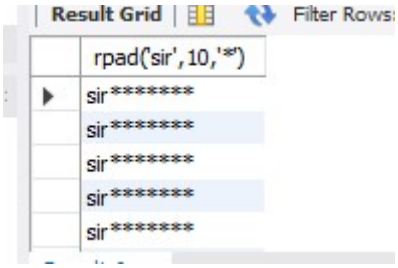
Select lpad('sir',10,'*') from manager;



	lpad('sir',10,'*')
▶	*******sir
	*******sir
	*******sir
	*******sir
	*******sir

6)Rpad

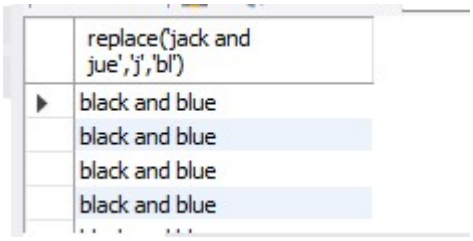
Select rpad('sir',10,'*') from manager;



	rpad('sir',10,'*')
▶	sir*******
	sir*******
	sir*******
	sir*******
	sir*******

7)Replace

Select replace('jack and jue','j','bl') from manager;



	replace('jack and jue','j','bl')
▶	black and blue
	black and blue
	black and blue
	black and blue

Aggregate Function

1)min

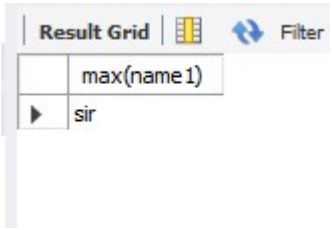
Select min(name1) from manager;

Result Grid

	min(name1)
▶	hea

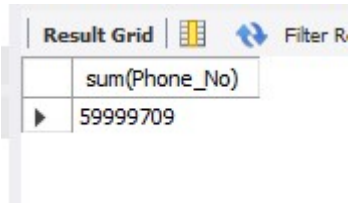
2)max

Select max(name1) from manager;



3)Sum

Select sum(Phone_No) from manager



Conclusion:

1. Write syntax and explanation for each of the five aggregate functions

Ans

1. COUNT():

- Syntax: COUNT(expression)

- Explanation: Calculates the number of rows that match a specified condition or expression.

2. SUM():

- Syntax: SUM(expression)

- Explanation: Calculates the sum of values in a numeric column or result set, ignoring NULL values.

3. AVG():

- Syntax: AVG(expression)

- Explanation: Calculates the average value of a numeric column or result set, ignoring NULL values.

4. MIN():

- Syntax: MIN(expression)

- Explanation: Returns the minimum value from a set of values, ignoring NULL values.

5. MAX():

- Syntax: MAX(expression)

- Explanation: Returns the maximum value from a set of values, ignoring NULL values.

These functions are used for data analysis and summarization in SQL queries.

2. Show results of operations performed.

Ans
1. COUNT(): Shows the number of rows that match a specified condition or expression.
2. SUM(): Displays the total sum of values in a numeric column or result set.
3. AVG(): Presents the average value of a numeric column or result set.
4. MIN(): Shows the smallest value from a set of values.
5. MAX(): Displays the largest value from a set of values.

These functions provide valuable insights into data analysis and summarization.

Experiment No.6
Implement various join operations
Date of Performance:
Date of Submission:

Aim :- Write simple query to implement join operations(equi join, natural join, inner join, outer joins).

Objective :- To apply different types of join to retrieve queries from the database management system.

Theory:

SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

A. INNER JOIN

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

INNER JOIN table2

ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

B. LEFT JOIN

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

C. RIGHT JOIN

RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. For the rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

RIGHT JOIN table2

ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

D. FULL JOIN

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain NULL values.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

FULL JOIN table2

ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Implementation:

SELECT Employee.EmployeeID, Employee.FirstName AS EmployeeFirstName, Employee.LastName AS EmployeeLastName,

Manager.ManagerID, Manager.FirstName AS ManagerFirstName, Manager.LastName AS ManagerLastName

FROM Employee

INNER JOIN Manager ON Employee.ManagerID = Manager.ManagerID;

```
53 •  SELECT Employee.EmployeeID, Employee.FirstName AS EmployeeFirstName, Employee.LastName AS EmployeeLastName,
54    Manager.ManagerID, Manager.FirstName AS ManagerFirstName, Manager.LastName AS ManagerLastName
55    FROM Employee
56    INNER JOIN Manager ON Employee.ManagerID = Manager.ManagerID;
57
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

EmployeeID	EmployeeFirstName	EmployeeLastName	ManagerID	ManagerFirstName	ManagerLastName
201	John	Doe	101	Emily	Davis
204	Sophia	Brown	101	Emily	Davis
202	Alice	Smith	102	David	Wilson
205	James	Miller	102	David	Wilson
203	Michael	Johnson	103	Emma	Taylor

SELECT Employee.EmployeeID, Employee.FirstName AS EmployeeFirstName, Employee.LastName AS EmployeeLastName,

Manager.ManagerID, Manager.FirstName AS ManagerFirstName, Manager.LastName AS ManagerLastName

FROM Employee

LEFT JOIN Manager ON Employee.ManagerID = Manager.ManagerID;

```
58 •  SELECT Employee.EmployeeID, Employee.FirstName AS EmployeeFirstName, Employee.LastName AS EmployeeLastName,
59    Manager.ManagerID, Manager.FirstName AS ManagerFirstName, Manager.LastName AS ManagerLastName
60    FROM Employee
61    LEFT JOIN Manager ON Employee.ManagerID = Manager.ManagerID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

EmployeeID	EmployeeFirstName	EmployeeLastName	ManagerID	ManagerFirstName	ManagerLastName
201	John	Doe	101	Emily	Davis
202	Alice	Smith	102	David	Wilson
203	Michael	Johnson	103	Emma	Taylor
204	Sophia	Brown	101	Emily	Davis
205	James	Miller	102	David	Wilson

SELECT Employee.EmployeeID, Employee.FirstName AS EmployeeFirstName, Employee.LastName AS EmployeeLastName,

Manager.ManagerID, Manager.FirstName AS ManagerFirstName, Manager.LastName AS ManagerLastName

FROM Employee

RIGHT JOIN Manager ON Employee.ManagerID = Manager.ManagerID;

```
63 •  SELECT Employee.EmployeeID, Employee.FirstName AS EmployeeFirstName, Employee.LastName AS EmployeeLastName,
64    Manager.ManagerID, Manager.FirstName AS ManagerFirstName, Manager.LastName AS ManagerLastName
65    FROM Employee
66    RIGHT JOIN Manager ON Employee.ManagerID = Manager.ManagerID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

EmployeeID	EmployeeFirstName	EmployeeLastName	ManagerID	ManagerFirstName	ManagerLastName
201	John	Doe	101	Emily	Davis
204	Sophia	Brown	101	Emily	Davis
202	Alice	Smith	102	David	Wilson
205	James	Miller	102	David	Wilson
203	Michael	Johnson	103	Emma	Taylor

SELECT Employee.EmployeeID, Employee.FirstName AS EmployeeFirstName, Employee.LastName AS EmployeeLastName,

Manager.ManagerID, Manager.FirstName AS ManagerFirstName, Manager.LastName AS ManagerLastName

FROM Employee

FULL OUTER JOIN Manager ON Employee.ManagerID = Manager.ManagerID;

Conclusion:

1. Illustrate how to perform natural join for the joining attributes with different names with a suitable example.

Ans

In a natural join, tables are joined based on columns with the same name. If columns have different names, they can still be joined implicitly.

Example:

Two tables: "employees" and "departments".

"employees" has columns: emp_id, emp_name, dept_id.

"departments" has columns: dept_id, dept_name.

Performing a natural join:

```sql
SELECT *
FROM employees
NATURAL JOIN departments;
```

Result:

| emp_id | emp_name | dept_id | dept_name |

Even though the department ID columns have different names, the natural join automatically matches them based on their values.

2. Il lustrate significant differences between natural join equi join and inner join.

Ans

1. Natural Join:
   - Joins based on columns with the same name.
   - Automatically matches columns with identical names.
   - Can lead to unexpected results if column names are inconsistent.

2. Equi Join
   - Specific type of inner join using the equality operator (=).
   - Requires specifying columns to join on explicitly.
   - Returns rows where specified columns match.

3. Inner Join:
   - General join returning rows when there's a match based on specified conditions.
   - Can use any join condition, including equality or other logical conditions.
   - Requires explicitly specifying the join condition.

Key Differences:
- Natural join is based on column names, while equi join and inner join require explicit conditions.
- Equi join is a specific type of inner join using the equality operator.
- Inner join is a broader concept encompassing all joins where rows match based on specified conditions.

Experiment No.7
Perform DCL and TCL commands
Date of Performance:
Date of Submission:

Aim :- Write a query to implement Data Control Language(DCL) and Transaction Control Language(TCL) commands

Objective :- To learn DCL commands like Grant and Revoke privileges to the user and TCL commands to commit the transactions and recover it using rollback and save points.

Theory:

Data Control Language:

DCL commands are used to grant and take back authority from any database user.

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

1. GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

b. Revoke: It is used to take back permissions from the user.

Example

1. REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

1. COMMIT;

Example:

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

1. ROLLBACK;

Example:

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

2. SAVEPOINT SAVEPOINT_NAME;

Implementation:

```
GRANT SELECT, UPDATE ON Employee TO Alice, Bob;


REVOKE SELECT, UPDATE ON Employee FROM Charlie, David;


DELETE FROM Employee

WHERE AGE = 25;

COMMIT;


DELETE FROM Employee

WHERE AGE = 35;
```

ROLLBACK;

SAVEPOINT delete_employee;

```
74 •   GRANT SELECT, UPDATE ON Employee TO Alice, Bob;
75
76 •   REVOKE SELECT, UPDATE ON Employee FROM Charlie, David;
77
78 •   DELETE FROM Employee
79     WHERE AGE = 25;
80 •   COMMIT;
81
82 •   DELETE FROM Employee
83     WHERE AGE = 35;
84
85 •   ROLLBACK;
86
87 •   SAVEPOINT delete_employee;
```

Conclusion:

1. Explain about issues faced during rollback in mysql and how it got resolved.

Ans In MySQL, rollback issues arise from transactional inconsistencies, such as deadlocks, lock timeouts, large transactions, or disk space exhaustion.

To resolve these issues, MySQL implements:

- Different transaction isolation levels to control visibility of changes.

- Proper transaction management practices to prevent long-running transactions.

- Monitoring and tuning of transaction activity and database performance.

- Continuous improvements in storage engine technology to enhance concurrency control and rollback performance.

2. Explain how to create a user in sql.

Ans To create a user in SQL:

1. Use the CREATE USER statement.
2. Specify the username and password.
3. Optionally, define privileges or roles for the user.

For example:
```sql
```

```
CREATE USER 'username'@'hostname' IDENTIFIED BY 'password';
```
This creates a user with the specified username and password.

You can also grant privileges to the user using the GRANT statement:
```sql
GRANT SELECT, INSERT ON database.* TO 'username'@'hostname';
```
This grants SELECT and INSERT privileges on a specific database to the user.

Finally, don't forget to flush privileges to apply the changes:
```sql
FLUSH PRIVILEGES;
```
This ensures that the changes take effect immediately.

Experiment No.8
Implementation of Views and Triggers
Date of Performance:
Date of Submission:

Aim :- Write a SQL query to implement views and triggers

Objective :- To learn about virtual tables in the database and also PLSQL constructs

Theory:

SQL Views:

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.
You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.
A view is created with the CREATE VIEW statement.

CREATE VIEW Syntax

CREATE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;

SQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

SQL CREATE OR REPLACE VIEW Syntax

CREATE OR REPLACE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;

SQL Dropping a View

A view is deleted with the DROP VIEW statement.

SQL DROP VIEW Syntax

DROP VIEW view_name;

Trigger: A trigger is a stored procedure in the database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]

Explanation of syntax:

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger_body]: This provides the operation to be performed as trigger is fired

Conclusion:

1. Brief about the benefits for using views and triggers.

Ans Using views and triggers in databases provides numerous advantages:

Views:

1. Data Abstraction: Simplify data access by presenting a subset of information from one or more tables.

2. Data Security: Control access to sensitive data, offering a layer of security by limiting what users can see.

3. Performance Optimization: Improve query performance by storing frequently used complex operations.

4. Consistency: Ensure data consistency by centralizing logic and business rules.

5. Encapsulation: Hide complexity by encapsulating complex SQL queries, offering a simpler interface for users.

Triggers:

1. Enforce Data Integrity: Automatically enforce constraints on data to maintain integrity.

2. Audit Trail: Record changes made to the database for auditing purposes.

3. Complex Business Logic: Implement complex business logic not achievable with standard SQL operations.

4. Cascade Updates: Automatically propagate changes to related tables to maintain referential integrity.

5. Data Replication: Facilitate data replication between databases for consistency across distributed systems.

2. Explain different strategies to update views

Ans

Different strategies to update views include:

1. Simple Update: Directly updating the underlying tables that the view is based on. This strategy is straightforward but may not always be feasible if the view involves complex joins or aggregates.

2. Update Through Triggers: Employing INSTEAD OF triggers to intercept updates to the view and translate them into corresponding updates to the underlying tables. This allows for more flexibility and control over how updates are handled.

3. Updateable Views with Joins: Creating updateable views that involve simple joins and have a one-to-one correspondence with a single underlying table. Updates to such views can be propagated directly to the underlying table.

4. Partitioned Views: Using partitioned views to break down updates into smaller, more manageable pieces by dividing the data into separate tables or views based on specific criteria. Updates can then be applied to individual partitions as needed.

5. Materialized Views: Maintaining materialized views that store precomputed results of complex queries, allowing for efficient updates by refreshing the view periodically or incrementally.

6. Dynamic SQL: Dynamically generating SQL statements to update the underlying tables based on the current state of the view. This approach requires careful handling to ensure correctness and security. Each strategy has its advantages and limitations, and the choice depends on factors such as the complexity of the view, performance requirements, and the level of control needed over updates.

Experiment No.9
Demonstrate Database connectivity
Date of Performance:
Date of Submission:

Aim :- Write a java program to connect Java application with the MySQL database

Objective :- To learn database connectivity

Theory:

Database used : MySql

1. Driver class: The driver class for the mysql database is com.mysql.jdbc.Driver.
2. Connection URL: The connection URL for the mysql database is jdbc:mysql://localhost:3306/loan management where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, can also use IP address, 3306 is the port number and loan management is the database name.
3. Username: The default username for the mysql database is Hiren.
4. Password: It is the password given by the user at the time of installing the mysql database. Password used is " ".

To connect a Java application with the MySQL database, follow the following steps.

- First create a database and then create a table in the mysql database.
- To connect java application with the mysql database, mysqlconnector.jar file is required to be loaded.
- download the jar file mysql-connector.jar
- add the jar file to the same folder as the java program.
- Compile and run the java program to retrieve data from the database.

Conclusion: Data has been retrieved successfully from a table by establishing database connectivity of java program with mysql database.

1. Explain steps to connect a java application with the MySQL database

Ans
1. Download MySQL JDBC Driver: Download the MySQL JDBC driver (mysql-connector.jar) from the MySQL website or Maven repository.
2. Create a Database and Table: Use MySQL client tools or command-line interface to create a database and table within MySQL. For example:

```sql
CREATE DATABASE mydatabase;
USE mydatabase;
CREATE TABLE mytable (
    id INT PRIMARY KEY,
    name VARCHAR(50)
);
```

3. Load MySQL JDBC Driver: Load the MySQL JDBC driver class using `Class.forName()`. This step is necessary to register the driver with the DriverManager.

```java
Class.forName("com.mysql.cj.jdbc.Driver");
```

4. Establish Connection: Use `DriverManager.getConnection()` to establish a connection to the MySQL database by providing the JDBC URL, username, and password.

```java
String url = "jdbc:mysql://localhost:3306/mydatabase";
String username = "root";
String password = "password";
Connection connection = DriverManager.getConnection(url, username, password);
```

5. Create Statement: Create a Statement object to execute SQL queries against the database.

```java
Statement statement = connection.createStatement();
```

6. Execute Query: Use the `executeQuery()` method to execute a SELECT query and retrieve data from the database.

```java
ResultSet resultSet = statement.executeQuery("SELECT * FROM mytable");
```

7. Process Results: Iterate through the ResultSet to retrieve data from each row and process it accordingly.

```java
while (resultSet.next()) {
    int id = resultSet.getInt("id");
    String name = resultSet.getString("name");
    System.out.println("ID: " + id + ", Name: " + name);
}
```

8.Close Resources: Close the ResultSet, Statement, and Connection objects in a `finally` block or using try-with-resources to release database resources.

```java
resultSet.close();
statement.close();
connection.close();
```

These steps demonstrate how to establish a connection, execute a query, and retrieve data from a MySQL database using JDBC in a Java application.

Experiment No.10
Implementation and demonstration of Transaction and Concurrency control techniques using locks
Date of Performance:
Date of Submission:

Aim :- Write a query to lock and unlock a table for transaction and concurrency control.

Objective :- To learn locking of tables for transaction processing and concurrency control.

Theory:

A lock is a mechanism associated with a table used to restrict the unauthorized access of the data in a table. MySQL allows a client session to acquire a table lock explicitly to cooperate with other sessions to access the table's data. MySQL also allows table locking to prevent unauthorized modification into the same table during a specific period.

Table Locking in MySQL is mainly used to solve concurrency problems. It will be used while running a transaction, i.e., first read a value from a table (database) and then write it into the table (database).

MySQL provides two types of locks onto the table, which are:

READ LOCK: This lock allows a user to only read the data from a table.

WRITE LOCK: This lock allows a user to do both reading and writing into a table.

The following is the syntax that allows us to acquire a table lock explicitly:

LOCK TABLES table_name [READ | WRITE];

The following is the syntax that allows us to release a lock for a table in MySQL:

UNLOCK TABLES;

Conclusion: Locking and unlocking of tables is achieved and verified using insert command in the same table of a database system.

1. Explain Transaction and Concurrency control techniques using locks.

   Ans Transactions are units of work in a database, ensuring Atomicity (all-or-nothing), Consistency (data integrity), Isolation (transactions appear serial), and Durability (committed changes are permanent). They group operations like fund transfers or inventory updates.

   Concurrency control techniques manage simultaneous transaction execution:

   - Locking: Transactions acquire shared (read) or exclusive (write) locks on data objects.

   - Two-Phase Locking (2PL): Transactions acquire locks in two phases: growing (acquiring locks) and shrinking (releasing locks).

   - Deadlock Detection and Resolution: Mechanisms to identify and resolve deadlocks where transactions wait for each other.

   - Timestamp-based Concurrency Control: Assigns timestamps to transactions, determining execution order.

   - Optimistic Concurrency Control: Transactions execute without locks, resolving conflicts before committing changes.

   These techniques ensure safe, efficient transaction execution in multi-user databases, maintaining data integrity and scalability.