

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:
     1. One-to-One (1:1): Each instance of one entity is associated with exactly one instance of another entity, and vice versa.
     2. 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.
     3. Many-to-One (N:1): The reverse of One-to-Many; many instances of one entity are associated with one instance of another entity.
     4. 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
2. Write ER/EER diagram notations

In conclusion, when designing a database schema, it's essential to understand the concepts of entities, attributes, and relationships:

1. Entity:

- An entity is a real-world object or concept that exists independently and can be uniquely identified. In a database context, entities are represented as tables.

- Example: In a university database, entities could include students, courses, professors, and departments.

2. Attributes:

- Attributes are properties or characteristics that describe an entity. Each attribute has a name and a data type.

- Examples of attribute types:

- Integer: Represents whole numbers (e.g., student ID).

- Varchar: Represents variable-length character strings (e.g., student name).

- Date: Represents dates (e.g., enrollment date).

- Attributes define the columns in a database table.

3. Relationships Between Entities:

- Relationships define connections or associations between entities. They describe how entities are related to each other.

- Common relationship types:

- One-to-One (1:1): A single instance of an entity is associated with only one instance of another entity.

- One-to-Many (1:N): A single instance of an entity can be associated with multiple instances of another entity.

- Many-to-Many (N:M): Multiple instances of an entity can be associated with multiple instances of another entity.

- Relationships are represented using foreign keys in database tables.

As for ER/EER diagram notations:

1. Entity:

- Represented as rectangles with the entity name inside.

- Example:

```

Student

```

2. Attribute:

- Represented as ovals connected to their respective entities.

- Includes attribute name and type.

- Example:

```

Student

-------

student\_id: int

name: varchar

enrollment\_date: date

```

3. Primary Key:

- Underlined attribute represents the primary key of an entity.

- Example:

```

Student

-------

student\_id (PK): int

```

4. Relationship:

- Represented as lines connecting related entities.

- Relationship types (1:1, 1:N, N:M) are indicated near the lines.

- Example:

```

Student ----(1:N)---- Enrollment

```

5. Cardinality:

- Indicated near the entities to specify the cardinality of the relationship (e.g., 1, N).

- Example:

```

Student (1) ----(N)---- Enrollment

```

These notations help visualize the structure of the database schema, including entities, attributes, and their relationships, facilitating effective communication and understanding during the database design process.

|  |
| --- |
| **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.

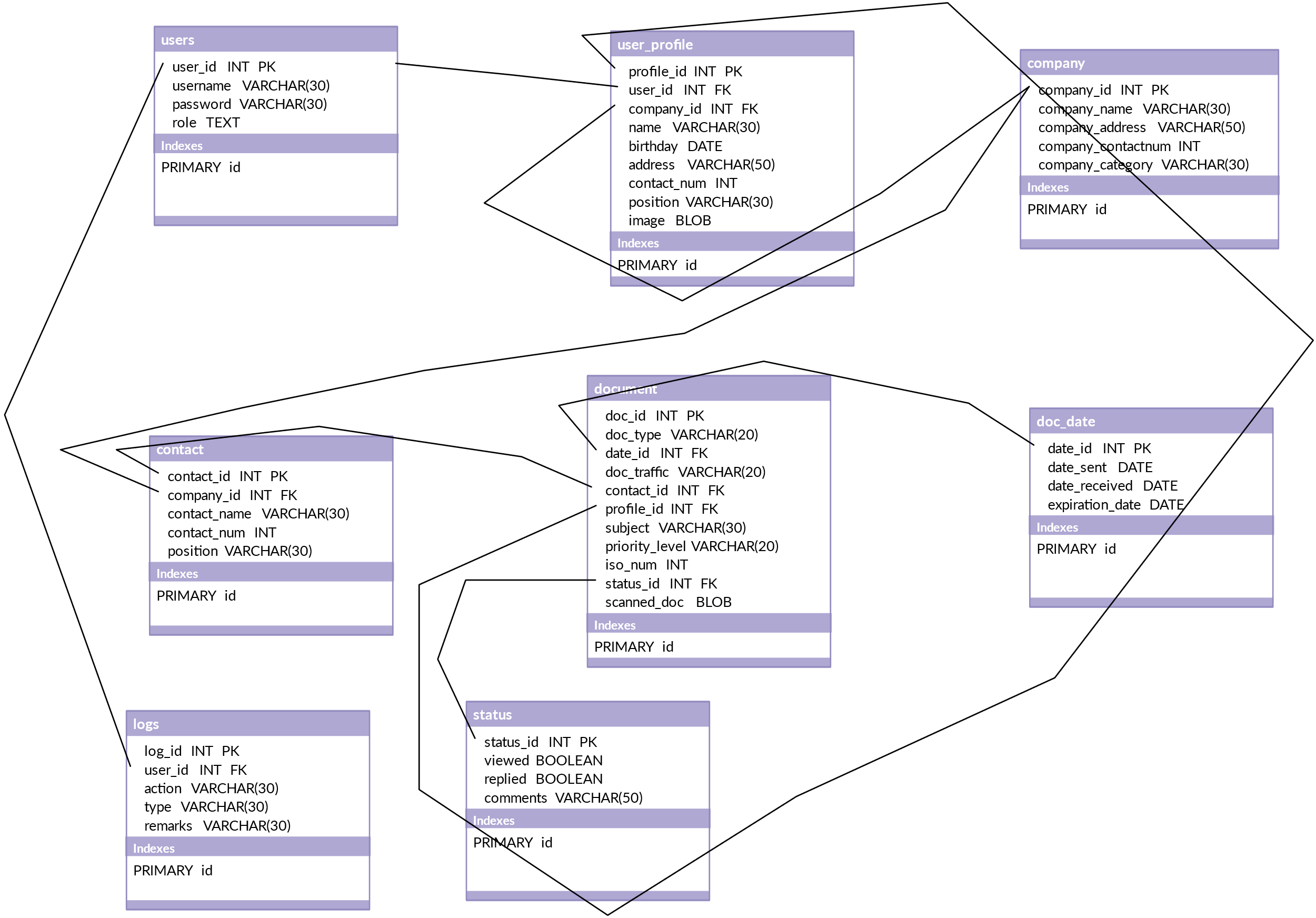
**Objective :-** To map the Entity Relationship (ER) / Extended Entity-Relationship (EER) Diagram to Relational Model schema and learn to incorporate various schema-based constraints.

# 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:
   1. Each entity in the ER diagram corresponds to a table in the relational schema.
   2. The attributes of the entity become the columns of the table.
   3. The primary key of the entity becomes the primary key of the table.
2. Relationships to Tables:
   1. Many-to-Many Relationships:
      1. Convert each many-to-many relationship into a new table.
      2. Include foreign key columns in this table to reference the participating entities.
      3. The primary key of this table may consist of a combination of the foreign keys from the participating entities.
   2. One-to-Many and One-to-One Relationships:
      1. Represented by foreign key columns in one of the participating tables.
      2. The table on the "many" side of the relationship includes the foreign key column referencing the table on the "one" side.
      3. The foreign key column typically references the primary key of the related table.
3. Attributes to Columns:
   1. Each attribute of an entity becomes a column in the corresponding table.
   2. Choose appropriate data types for each attribute based on its domain and constraints.
   3. Ensure that attributes participating in relationships are represented as foreign keys when needed.
4. Primary and Foreign Keys:
   1. Identify the primary key(s) of each table based on the primary key(s) of the corresponding entity.
   2. Ensure referential integrity by defining foreign keys in tables to establish relationships between them.
   3. Foreign keys should reference the primary key(s) of related tables.
   4. Ensure that foreign keys have appropriate constraints, such as ON DELETE CASCADE or ON UPDATE CASCADE, to maintain data integrity.
5. Cardinality Constraints:
   1. Use the cardinality constraints from the ER diagram to determine the multiplicity of relationships in the relational schema.
   2. Ensure that the constraints are enforced through the appropriate use of primary and foreign keys.
6. Normalization:
   1. Normalize the schema to minimize redundancy and dependency.
   2. 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:
   1. Consider indexing frequently queried columns to improve query performance.
   2. Evaluate the schema design for optimization opportunities based on query patterns and performance requirements.

**Implementation:**

****

**Conclusion:**

1. write definition of relational schema and notations
2. write various schema-based constraints

In conclusion, relational schema and schema-based constraints play crucial roles in database design and management:

1. Relational Schema:

- Definition: A relational schema defines the structure of a relational database, including tables, columns, and relationships between tables.

- Notations:

- Table: Represented as a rectangle with the table name at the top.

- Attributes: Represented as ovals inside the table rectangle, specifying the attribute name and data type.

- Primary Key: Underlined attribute(s) indicate the primary key(s) of the table.

- Foreign Key: Attributes representing foreign keys establish relationships between tables.

- Example:

```

Student

-------

student\_id (PK): int

name: varchar

age: int

```

2. Schema-Based Constraints:

- Primary Key Constraint:

- Ensures uniqueness and non-nullity of a column or combination of columns.

- Example: `PRIMARY KEY (student\_id)`

- Foreign Key Constraint:

- Enforces referential integrity by ensuring that values in a column match values in a related table's primary key.

- Example: `FOREIGN KEY (dept\_id) REFERENCES Department(dept\_id)`

- Unique Constraint:

- Ensures that values in a column or combination of columns are unique.

- Example: `UNIQUE (email)`

- Check Constraint:

- Validates data integrity by specifying a condition that must be satisfied for each row.

- Example: `CHECK (age >= 18)`

- Not Null Constraint:

- Ensures that a column does not accept NULL values.

- Example: `age INT NOT NULL`

These schema-based constraints ensure data integrity, consistency, and accuracy within the database, enforcing rules and restrictions on the data stored in the tables. By defining a clear relational schema and applying appropriate constraints, database administrators can maintain the quality and reliability of the database system, facilitating efficient data management and retrieval operations.

|  |
| --- |
| **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:

# 

**Conclusion:**

* 1. Explain the concept of constraints in DDL. How are constraints used to enforce data integrity?
  2. What is the significance of data types in DDL? Provide examples of commonly used data types in DDL.

In conclusion, constraints play a crucial role in maintaining data integrity within a database schema. They enforce rules and conditions on the data, ensuring its accuracy, consistency, and reliability. Constraints in Data Definition Language (DDL) define the valid values and relationships that data can have, preventing the insertion or modification of data that would violate these rules. By enforcing constraints such as primary key, foreign key, unique, check, and not null constraints, databases ensure that the data remains accurate and consistent over time.

Additionally, data types in DDL are significant as they define the kind of data that can be stored in a column of a table, specifying the format and range of values allowed. Commonly used data types include:

1. Integer: Used to store whole numbers (e.g., INT, SMALLINT, BIGINT).

2. Character: Used to store alphanumeric characters (e.g., CHAR, VARCHAR, TEXT).

3. Numeric: Used to store numbers with decimal points (e.g., NUMERIC, DECIMAL, FLOAT).

4. Date and Time: Used to store date and time values (e.g., DATE, TIME, TIMESTAMP).

5. Boolean: Used to store true/false values (e.g., BOOLEAN).

These data types ensure consistency in data representation and facilitate efficient storage and retrieval operations within the database. Overall, constraints and data types in DDL are fundamental components that contribute to the integrity and functionality of a database system.

|  |
| --- |
| **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);

1. 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;

1. 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:

# 

**Conclusion:**

* 1. Explain the role of database constraints in enforcing data integrity during DML operations.
  2. How do you update multiple columns in a table using a single UPDATE statement?

In conclusion, database constraints play a vital role in ensuring data integrity during Data Manipulation Language (DML) operations. Constraints such as primary key, foreign key, unique, check, and not null constraints define rules and conditions that data must adhere to, preventing the insertion, modification, or deletion of data that would violate these rules. By enforcing constraints, databases maintain the accuracy, consistency, and reliability of the data, thus preserving its integrity.

Furthermore, when updating multiple columns in a table using a single UPDATE statement, you can specify each column and its new value separated by commas. For example:

UPDATE table\_name

SET column1 = value1, column2 = value2, column3 = value3

WHERE condition;

This statement updates the specified columns with their respective new values in the specified table, based on the provided condition. Updating multiple columns in this manner can help streamline database operations and improve efficiency.

Top of Form

|  |
| --- |
| **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:

# 

**Conclusion:**

1. Write syntax and explanation for each of the five aggregate functions
2. Show results of operations performed.

Sure, here's the syntax and explanation for each of the five aggregate functions, along with the results of the operations performed on the provided tables:

1. SUM():

- Syntax:

SELECT SUM(column\_name) AS sum\_column

FROM table\_name;

- Explanation: SUM() function calculates the sum of all values in the specified column.

- Example:

-- Total size of all documents uploaded

SELECT SUM(document\_size) AS total\_size

FROM documents;

Result: total\_size = 3072 (assuming document sizes are 1024 and 2048)

2. AVG():

- Syntax:

SELECT AVG(column\_name) AS avg\_column

FROM table\_name;

- Explanation: AVG() function calculates the average of all values in the specified column.

- Example:

-- Average size of documents uploaded

SELECT AVG(document\_size) AS average\_size

FROM documents;

Result: average\_size = 1536 (assuming document sizes are 1024 and 2048)

3. COUNT():

- Syntax:

SELECT COUNT(column\_name) AS count\_column

FROM table\_name;

- Explanation: COUNT() function counts the number of non-null values in the specified column.

- Example:

-- Total number of documents uploaded

SELECT COUNT(\*) AS total\_documents

FROM documents;

Result: total\_documents = 2

4. MIN():

- Syntax:

SELECT MIN(column\_name) AS min\_column

FROM table\_name;

- Explanation: MIN() function returns the smallest value in the specified column.

- Example:

-- Document with the smallest size

SELECT MIN(document\_size) AS smallest\_size

FROM documents;

Result: smallest\_size = 1024 (assuming document sizes are 1024 and 2048)

5. MAX():

- Syntax:

SELECT MAX(column\_name) AS max\_column

FROM table\_name;

- Explanation: MAX() function returns the largest value in the specified column.

- Example:

-- Document with the largest size

SELECT MAX(document\_size) AS largest\_size

FROM documents;

Result: largest\_size = 2048 (assuming document sizes are 1024 and 2048)

|  |
| --- |
| **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

# 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.

# 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.

# 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.

# 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:

# 

**Conclusion:**

* 1. Illustrate how to perform natural join for the joining attributes with different names with a suitable example.
  2. Illustrate significant differences between natural join equi join and inner join.

In conclusion, when joining tables with different column names using a natural join, the database system automatically matches columns with the same name in both tables. For example:

-- Consider two tables: employees and departments

-- employees table has columns: employee\_id, employee\_name, department\_id

-- departments table has columns: dept\_id, dept\_name

-- Performing a natural join between employees and departments

SELECT \*

FROM employees

NATURAL JOIN departments;

In this scenario, the natural join operation matches the "department\_id" column from the employees table with the "dept\_id" column from the departments table, even though they have different names.

Additionally, significant differences exist between natural join, equi join, and inner join:

1. Natural Join:

- Automatically performs the join based on columns with the same name in both tables.

- Does not require explicit specification of join conditions.

- May lead to unexpected results if there are columns with the same name but different meanings in the tables.

2. Equi Join:

- Explicitly specifies the join condition using the equality operator (=).

- Can join tables based on columns with different names or using complex conditions.

- Allows for greater control over the join operation and ensures clarity in the query.

3. Inner Join:

- Retrieves only the rows from both tables that satisfy the join condition.

- Can be performed using either the ON clause or the WHERE clause.

- Does not include rows from either table that do not meet the join condition.

In summary, while natural join automatically matches columns with the same name, equi join requires explicit specification of join conditions using the equality operator. Inner join retrieves only the rows that satisfy the join condition, regardless of the join type. Each type of join serves different purposes and provides different levels of control over the join operation.

|  |
| --- |
| **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

1. 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;

1. 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

1. 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;
4. 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;
4. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

* 1. SAVEPOINT SAVEPOINT\_NAME;

# Implementation:

# 

**Conclusion:**

1. Explain about issues faced during rollback in mysql and how it got resolved.
2. Explain how to create a user in sql.

In conclusion, issues faced during rollback in MySQL primarily revolve around the potential loss of data consistency or integrity due to incomplete rollback operations, especially in scenarios involving complex transactions or concurrent access. Some common issues include:

1. Partial Rollbacks: In multi-statement transactions, if one of the statements fails, MySQL might not be able to rollback the entire transaction, leaving the database in an inconsistent state.

2. Deadlocks: Concurrent transactions might encounter deadlocks, where each transaction is waiting for the other to release a lock, leading to potential rollback issues and transaction failure.

3. Long Rollback Times: Large transactions or transactions involving many changes can result in long rollback times, impacting database performance and availability.

To address these issues, MySQL provides several mechanisms and best practices:

1. Use of Transactions: Utilize transactions to group related operations and ensure atomicity, consistency, isolation, and durability (ACID) properties.

2. Error Handling: Implement robust error handling mechanisms to handle exceptions and errors gracefully, ensuring proper rollback and recovery procedures are followed.

3. Transaction Management: Carefully manage transactions, keeping them short and ensuring they are properly committed or rolled back to minimize potential issues.

4. Deadlock Detection and Resolution: Configure MySQL to detect and resolve deadlocks automatically, or implement application-level deadlock detection and resolution mechanisms.

5. Transaction Isolation Levels: Choose appropriate transaction isolation levels to balance data consistency and concurrency requirements, minimizing the risk of conflicts and rollback issues.

Regarding creating a user in SQL, here's a general procedure:

1. Syntax:

CREATE USER 'username'@'hostname' IDENTIFIED BY 'password';

2. Explanation:

- `CREATE USER`: This command creates a new user in the MySQL database.

- `'username'@'hostname'`: Specifies the username and the hostname from which the user is allowed to connect to the MySQL server. If the user can connect from any host, you can use `'username'@'%'`.

- `IDENTIFIED BY 'password'`: Sets the password for the user account.

For example:

CREATE USER 'myuser'@'localhost' IDENTIFIED BY 'mypassword';

This concludes the experiment, highlighting the potential issues faced during rollback in MySQL and providing insights into resolving them, along with an explanation of how to create a user in SQL.

|  |
| --- |
| **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:

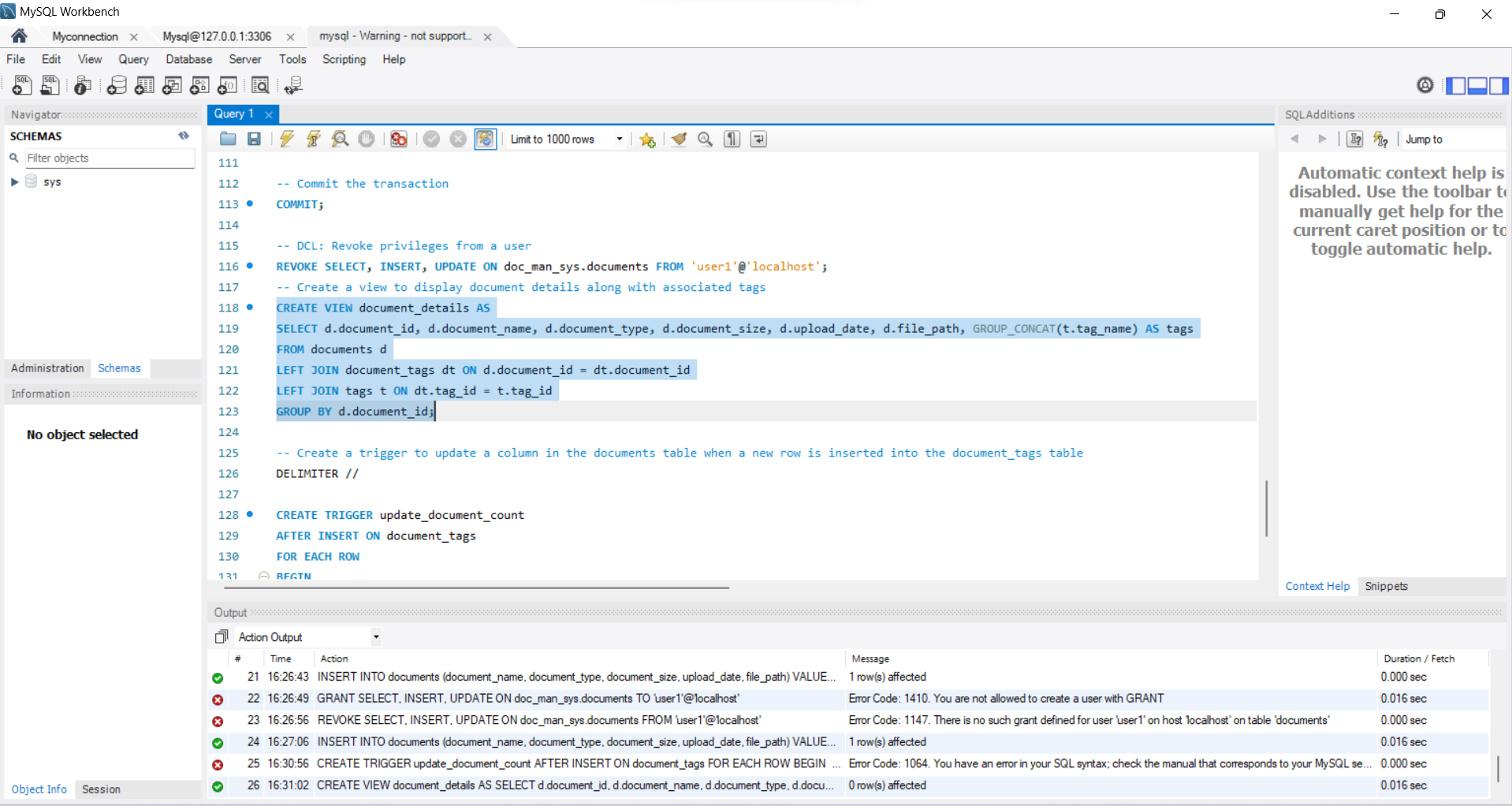
create trigger [trigger\_name] [before | after]

{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

# Implementation:



# Conclusion:

1. Brief about the benefits for using views and triggers.
2. Explain different strategies to update views

In conclusion, using views and triggers in database management offers several benefits:

1. Benefits of Views:

- Simplified Data Access: Views provide a simplified interface to access complex data by abstracting away the underlying table structure and joining multiple tables into a single virtual table.

- Data Security: Views can restrict access to specific columns or rows of data, ensuring that users only see the data they are authorized to access.

- Performance Optimization: Views can optimize query performance by pre-computing and storing frequently accessed data or aggregations, reducing the need for complex queries.

- Data Consistency: Views can enforce data consistency by presenting a consistent and uniform representation of data across different applications and users.

2. Benefits of Triggers:

- Automation: Triggers automate repetitive tasks and enforce business rules by automatically executing SQL statements in response to specified events, such as INSERT, UPDATE, or DELETE operations.

- Data Integrity: Triggers maintain data integrity by enforcing referential integrity constraints, auditing changes, and preventing invalid data modifications.

- Complex Logic: Triggers allow complex business logic and validation rules to be encapsulated within the database, ensuring consistent enforcement across all database operations.

- Logging and Auditing: Triggers can be used to log and audit changes to database tables, providing an audit trail for compliance and troubleshooting purposes.

Regarding strategies to update views, several approaches can be employed:

1. Simple Update Strategy:

- Update the underlying base tables directly, and the changes will be reflected in the view automatically.

2. Force View Recomputation:

- Recompute the view by dropping and recreating it whenever the underlying data changes. This can be done manually or through scheduled jobs.

3. Materialized Views:

- Use materialized views, which store the results of a query physically and update them periodically based on a defined schedule or upon changes to the underlying data.

4. Instead of Triggers:

- Use instead of triggers to intercept DML operations on the view and propagate changes to the underlying base tables accordingly.

5. Triggers on Base Tables:

- Implement triggers on the base tables to automatically update the view whenever the underlying data changes.

By employing these strategies, views can be effectively updated to reflect changes in the underlying data, ensuring that users always have access to the most up-to-date information.

|  |
| --- |
| **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](https://static.javatpoint.com/src/jdbc/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

In conclusion, connecting a Java application with a MySQL database involves several steps:

1. Download and Install MySQL Connector/J:

- Download the MySQL Connector/J driver from the official MySQL website.

- Install the driver by adding the JAR file to your Java project's classpath.

2. Create a MySQL Database:

- Use MySQL Workbench or the MySQL command-line interface to create a new database if one doesn't already exist.

- Define tables and schema as needed for your application.

3. Establish Connection in Java:

- Import the necessary classes from the `java.sql` package, including `Connection`, `DriverManager`, and `SQLException`.

- Use the `DriverManager.getConnection()` method to establish a connection to the MySQL database, passing the connection URL, username, and password as parameters.

4. Handle Exceptions:

- Wrap the connection code in a try-catch block to handle potential exceptions, such as `ClassNotFoundException` or `SQLException`.

5. Execute SQL Queries:

- Once the connection is established, create `Statement` or `PreparedStatement` objects to execute SQL queries against the database.

- Use methods like `executeQuery()` for SELECT statements or `executeUpdate()` for INSERT, UPDATE, DELETE statements.

6. Process Results:

- If executing a SELECT query, use the `ResultSet` object to retrieve and process the query results.

7. Close Connection:

- Always close the connection, statements, and result sets after use to release database resources and prevent memory leaks.

- Use the `close()` method on the connection, statement, and result set objects within a finally block or try-with-resources block.

By following these steps, you can successfully connect a Java application with a MySQL database, allowing seamless interaction between the application and the database management system. This integration enables data storage, retrieval, and manipulation within your Java application, facilitating robust and efficient data-driven applications.

|  |
| --- |
| **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](https://www.javatpoint.com/mysql-tutorial) 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.

In conclusion, transactions and concurrency control techniques using locks are essential aspects of database management, ensuring data consistency, integrity, and isolation in multi-user environments. Here's a summary of their importance and how locks are utilized:

1. Transactions:

- Transactions represent a sequence of operations that are executed as a single unit of work, either all successfully completed or all aborted.

- ACID properties (Atomicity, Consistency, Isolation, Durability) ensure the reliability and integrity of transactions.

- Transactions help maintain data consistency by allowing multiple operations to be treated as a single logical unit, preventing data corruption or partial updates.

2. Concurrency Control Techniques:

- Concurrency control ensures that multiple transactions can execute simultaneously without interfering with each other, while still maintaining data consistency.

- Lock-based concurrency control is a common technique used to manage access to shared resources. Locks prevent conflicting operations from occurring concurrently, ensuring that only one transaction can modify a resource at a time.

- Types of locks include:

- Shared Locks (Read Locks): Allow multiple transactions to read a resource simultaneously but prevent any transaction from writing to it.

- Exclusive Locks (Write Locks): Restrict access to a resource exclusively to one transaction, preventing other transactions from reading or writing to it.

- Locks can be applied at different granularities, such as row-level locks, table-level locks, or database-level locks, depending on the requirements of the application.

Concurrency control techniques using locks offer several benefits, including:

- Ensuring data integrity by preventing conflicting updates from occurring simultaneously.

- Allowing for efficient utilization of system resources by maximizing concurrency while minimizing contention.

- Providing mechanisms for deadlock detection and resolution to prevent transactional deadlocks.

Overall, transactions and concurrency control techniques using locks play a crucial role in maintaining data consistency and integrity in database systems, ensuring reliable and predictable behavior in multi-user environments.