



Vidyavardhini's

College of Engineering & Technology

Vasai Road (W)

Department of Artificial Intelligence and Data Science

Lab Manual

Semester	IV	Class S.E
Name	Yash Ravindra Kerkar	Roll no. 67
Course Code	CSL402	Academic Year 2023-24
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)
CSL 402.4	Apply procedure and functions for given schema	Apply	Apply (Level 3)
CSL 402.5	Design ER and EER diagrams for the real life problem with software tool.	Use	Apply (Level 3)
CSL 402.6	Construct database tables with different DDL and DML statements and apply integrity constraints.	Construct	Apply (Level 3)



Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

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					
	CSL 402.1	CSL 402.2	CSL402. 3	CSL 402.4	CSL 402.5	CSL 402.6
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		



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Perform DCL and TCL commands				3		
Implementation of Views and Triggers.					3	
Demonstrate Database connectivity						3
Implementation and demonstrate 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: Substantial (High)

If there is no correlation put “—”.



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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.



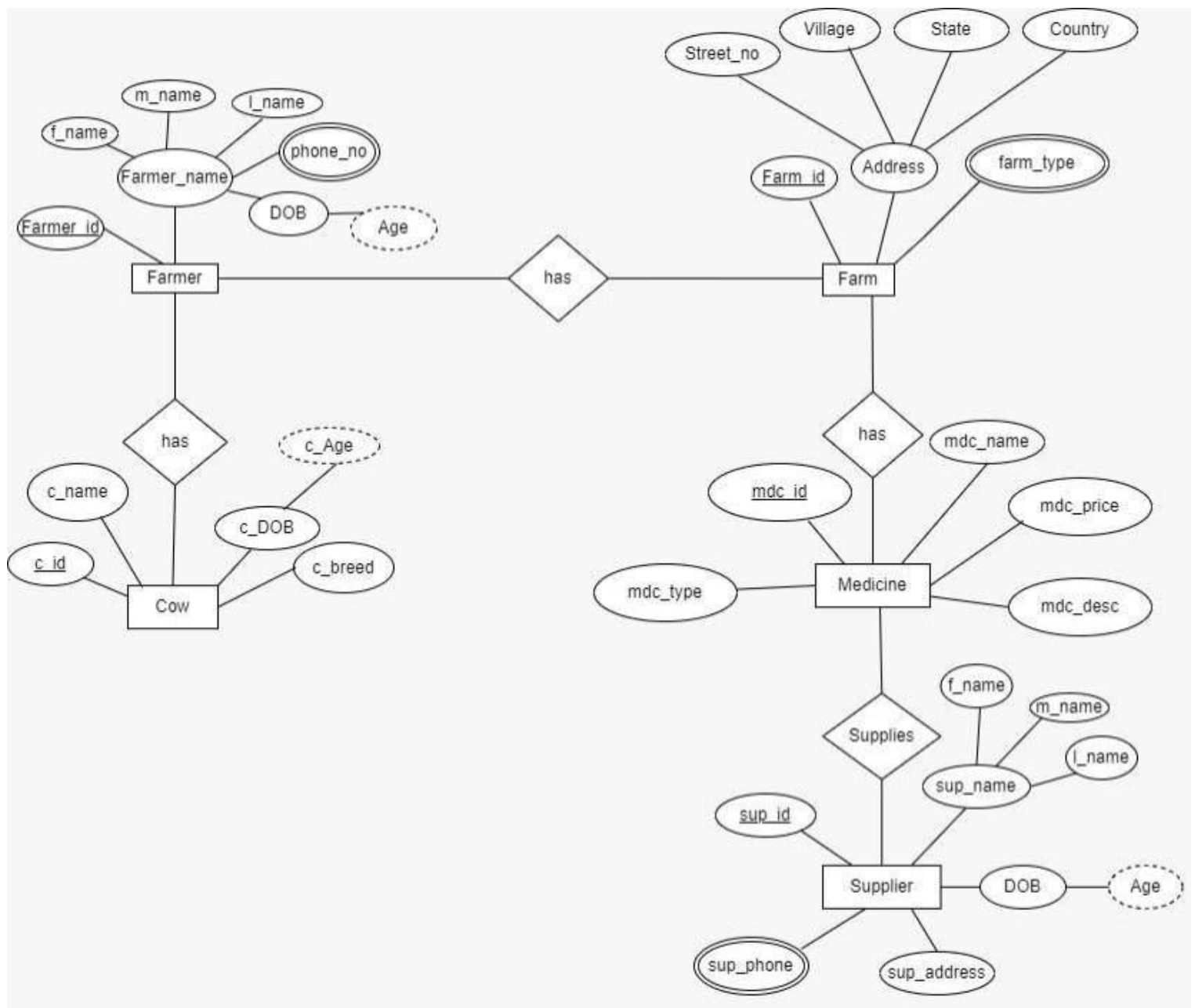
4. Cardinality:

- 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:

Farmers Management System





Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Conclusion:

In conclusion, the Entity-Relationship (ER) diagram for the Farmers Management System provides a comprehensive overview of the system's structure and relationships between various entities. Through this diagram, we have identified the key entities such as farmers, farm, cow, medicine and suppliers, along with their attributes and relationships.

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

Ans. **Entity:** An entity is a distinct object, concept, or thing about which data is stored in a database. It could be a person, place, thing, event, or concept that can be uniquely identified and distinguished from other entities. In an Entity-Relationship (ER) diagram, entities are represented by rectangles.

Attributes: Attributes are the characteristics or properties that describe an entity. They represent the specific pieces of information associated with an entity. Attributes can have different types, such as:

Simple Attribute: An attribute that cannot be divided into smaller parts. For example, "Name" or "Age".

Composite Attribute: An attribute that can be divided into smaller sub-parts, each representing a simpler attribute. For example, "Address" composed of "Street", "City", "State", and "Zip Code".

Derived Attribute: An attribute whose value is derived from other attributes. For example, "Age" can be derived from the "Date of Birth".

Multi-valued Attribute: An attribute that can hold multiple values for a single entity. For example, "Phone Numbers" for a person.

Key Attribute: An attribute whose values uniquely identify an entity within an entity set. For example, "ID" for a person.

Relationships: Relationships define the associations and interactions between entities in a database. They represent how entities are connected or related to each other. Relationships can have different types, such as:

One-to-One (1:1) Relationship: Each entity in the first entity set is associated with exactly one entity



One-to-Many (1:M) Relationship: Each entity in the first entity set can be associated with many entities in the second entity set, but each entity in the second entity set is associated with only one entity in the first entity set.

Many-to-Many (M:N) Relationship: Entities in both entity sets can be associated with many entities in the other entity set. This type of relationship requires the use of an associative entity (also known as a junction or link entity) to represent the association between them.

2. Write ER/EER diagram notations.

Ans. **Entity:** Represented by a rectangle with the entity name inside.

Attribute: Represented by an oval shape connected to the entity rectangle by a line. The attribute name is written inside the oval. Different types of attributes may have specific notations, such as underlining for primary keys or dashed ovals for derived attributes.

Primary Key Attribute: Typically underlined to indicate that it uniquely identifies each entity within the entity set.

- **Derived Attribute:** Represented by a dashed oval. It indicates that the value of the attribute can be derived from other attributes.
- **Multivalued Attribute:** Represented by a double oval. It indicates that an entity can have multiple values for that attribute.

Relationship: Represented by a diamond shape connecting two entities. The name of the relationship is written inside the diamond. Optionally, the cardinality (1, M) and participation constraints (total, partial) can be indicated near the ends of the relationship lines.

Cardinality: Indicates the number of instances of one entity that can be associated with instances of another entity. Common cardinality notations include 1 (one), M (many), 0..1 (zero or one), and 0..M (zero to many).



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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.



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Implementation:

Farmer's Management System

Farmer

FK 01

PK 01

Farmer_id	f_name	m_name	l_name	DOB	Gender	Street_no	Village	State	Country	Phone_no	Salary	Farm_id
-----------	--------	--------	--------	-----	--------	-----------	---------	-------	---------	----------	--------	---------

Farm

PK 02 FK 02

Farm_id	Farmer_id	Street_no	Village	State	Country	Crop_type	farm_size
---------	-----------	-----------	---------	-------	---------	-----------	-----------

Pesticides

PK 03

pes_id	pes_name	pes_price	pes_type	pes_desc
--------	----------	-----------	----------	----------

Suppliers

PK 04

sup_id	sup_name	DOB	sup_address	sup_phone	sup_gender	email	manufacturer	usage_instructions	expiry_date
--------	----------	-----	-------------	-----------	------------	-------	--------------	--------------------	-------------

Cow

PK 05

FK 03

c_id	c_name	c_breed	c_color	c_weight	health_status	farm_id
------	--------	---------	---------	----------	---------------	---------



Conclusion:

In this practical, we performed the crucial task of translating the conceptual design of the Farmers Management System, represented by an Entity-Relationship (ER) or Enhanced Entity-Relationship (EER) model, into a concrete relational schema. Through this process, we aimed to bridge the gap between the abstract representation of the system and its implementation in a relational database management system (RDBMS).

1. Write definition of relational schema and notations

Ans. A relational schema is a logical representation of the structure of a relational database. It defines the tables, attributes, keys, and relationships that constitute the database. The relational schema provides a blueprint for organizing and storing data in a relational database management system (RDBMS), enabling efficient data storage, retrieval, and manipulation.

Relational schema notations vary, but commonly used symbols and conventions include:

1. **Tables:** Represented as rectangles with the table name at the top. Each attribute is listed below the table name, along with its data type.
2. **Attributes:** Attributes are listed beneath the table name with their respective data types. Primary key attributes are often underlined to denote their uniqueness.
3. **Keys:** Primary keys are typically denoted by an asterisk (*) or the word "PK" next to the attribute name. Foreign keys are indicated similarly, with the word "FK" or by specifying the referenced table and attribute.
4. **Relationships:** Relationships between tables are depicted by lines connecting the related attributes. Cardinality and participation constraints may be indicated using symbols or annotations near the lines.



2. Write various schema-based constraints.

Ans. **Primary Key Constraint:** This constraint ensures that each row in a table is uniquely identifiable by a primary key attribute or combination of attributes. It prohibits duplicate and null values in the primary key column(s).

Unique Constraint: The unique constraint ensures that the values in one or more columns of a table are unique across all rows. Unlike primary keys, unique constraints allow null values, but if a column is marked as unique, only one row may contain a null value in that column.

Foreign Key Constraint: Foreign key constraints establish relationships between tables by enforcing referential integrity. A foreign key in one table references the primary key in another table, ensuring that every foreign key value must match a primary key value in the referenced table or be null.

Check Constraint: Check constraints define conditions that must be true for every row in a table. They allow you to specify rules that restrict the values allowed in certain columns. For example, a check constraint can ensure that values in a "age" column are greater than zero and less than 120.

Not Null Constraint: The not null constraint ensures that a column cannot contain null values. It requires that every row in the table must have a value for the specified column, preventing the insertion of null values.

Default Constraint: Default constraints specify a default value for a column when no value is explicitly provided during insertion. If a column with a default constraint is not specified in an INSERT statement, the default value will be used.

Domain Constraint: Domain constraints define the allowable range of values for a column based on its data type. For example, a domain constraint might restrict the values in a "gender" column to 'Male' or 'Female'.

Entity Integrity Constraint: Entity integrity constraints ensure that the primary key attribute of a table cannot contain null values, thus guaranteeing the uniqueness of each row in the table.

Referential Integrity Constraint: Referential integrity constraints enforce the consistency of relationships between tables. They ensure that foreign key values in one table must match primary key values in another table or be null.



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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:



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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>;



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Implementation:

Program:-

The screenshot shows the MySQL Workbench interface. The left sidebar contains the 'Navigator' panel with 'SCHEMAS' and a search bar. Below it, the 'Administration' and 'Schemas' tabs are visible, with 'Schemas' selected. The main area displays 'Query 1' with the following SQL code:

```
1 • CREATE DATABASE IF NOT EXISTS FARM_DATABASE;
2
3 • CREATE TABLE IF NOT EXISTS FMS (
4     F_ID INT PRIMARY KEY,
5     F_NAME VARCHAR(25),
6     L_NAME VARCHAR(25),
7     AGE INT,
8     MOB_NO INT,
9     COUNTRY VARCHAR(50),
10    LOAN INT
11 );
12
13 • CREATE TABLE IF NOT EXISTS FARM(
14     farm_id INT PRIMARY KEY,
15     farm_size int,
16     farm_type varchar(50),
17     farm_location varchar(50)
18 );
19
20 • CREATE TABLE IF NOT EXISTS SUPPLIERS (
21     S_ID INT PRIMARY KEY,
22     F_NAME VARCHAR(25),
23     L_NAME VARCHAR(25),
24     MOB_NO INT,
25     DATES DATE,
26     EMAIL VARCHAR(50),
27     FOREIGN KEY (S_ID)
28         REFERENCES FMS (F_ID)
29 );
```




```
Query 1 x
Limit to 1000 rows
29 );
30
31 • CREATE TABLE IF NOT EXISTS TOOLS(
32     T_ID INT PRIMARY KEY,
33     T_NAME VARCHAR(25),
34     T_PRICE INT,
35     T_DESC VARCHAR(200)
36 );
37
38 • ALTER TABLE FMS ADD LOAN VARCHAR(10);
39 • ALTER TABLE FMS DROP LOAN;
40
41 • ALTER TABLE FMS
42     RENAME COLUMN COUNTRY TO PLACE;
43
44 • TRUNCATE TABLE FMS;
45 • TRUNCATE TABLE SUPPLIERS;
46 • TRUNCATE TABLE FARM;
47 • TRUNCATE TABLE TOOLS;
48
49 • DROP TABLE FMS;
50 • DROP TABLE SUPPLIERS;
51 • DROP TABLE FARM;
52 • DROP TABLE TOOLS;
53
54 • SELECT * FROM FMS;
55 • SELECT * FROM SUPPLIERS;
56 • SELECT * FROM FARM;
57 • SELECT * FROM TOOLS;
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Output:

Create Farmers Table:

The screenshot shows the SQL Developer interface with the 'FMS' table selected. The 'Output' window displays the following actions:

#	Time	Action	Message	Duration / Fetch
7	01:24:35	CREATE TABLE IF NOT EXISTS TOOLS(T_ID INT PRIMARY KEY, T_NAME VARCHAR(25), T_PRICE...	0 row(s) affected	0.032 sec
8	01:24:58	ALTER TABLE FMS ADD LOAN VARCHAR(10)	Error Code: 1060. Duplicate column name 'LOAN'	0.016 sec
9	01:25:17	ALTER TABLE FMS RENAME COLUMN COUNTRY TO PLACE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.032 sec
10	01:25:26	SELECT * FROM FMS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
11	01:25:41	SELECT * FROM SUPPLIERS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
12	01:25:53	SELECT * FROM FMS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

Create Suppliers Table:

The screenshot shows the SQL Developer interface with the 'SUPPLIERS' table selected. The 'Output' window displays the following actions:

#	Time	Action	Message	Duration / Fetch
8	01:24:58	ALTER TABLE FMS ADD LOAN VARCHAR(10)	Error Code: 1060. Duplicate column name 'LOAN'	0.016 sec
9	01:25:17	ALTER TABLE FMS RENAME COLUMN COUNTRY TO PLACE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.032 sec
10	01:25:26	SELECT * FROM FMS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
11	01:25:41	SELECT * FROM SUPPLIERS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
12	01:25:53	SELECT * FROM FMS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
13	01:26:41	SELECT * FROM SUPPLIERS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Create Farm Table

#	Time	Action	Message	Duration / Fetch
9	01:25:17	ALTER TABLE FMS RENAME COLUMN COUNTRY TO PLACE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.032 sec
10	01:25:26	SELECT * FROM FMS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
11	01:25:41	SELECT * FROM SUPPLIERS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
12	01:25:53	SELECT * FROM FMS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
13	01:26:41	SELECT * FROM SUPPLIERS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
14	01:27:03	SELECT * FROM FARM LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

Create Tools Table

#	Time	Action	Message	Duration / Fetch
10	01:25:26	SELECT * FROM FMS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
11	01:25:41	SELECT * FROM SUPPLIERS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
12	01:25:53	SELECT * FROM FMS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
13	01:26:41	SELECT * FROM SUPPLIERS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
14	01:27:03	SELECT * FROM FARM LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
15	01:27:27	SELECT * FROM TOOLS LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec



Conclusion:

The completion of the practical exercise involving Data Definition Language (DDL) commands on the Farmers Management System offers valuable insights into the foundational aspects of database management. By applying various DDL commands, including CREATE TABLE, ALTER TABLE, and DROP TABLE, we gained hands-on experience in defining, modifying, and deleting the structure of the database schema.

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

Ans. Constraints in Data Definition Language (DDL) are rules or conditions applied to the structure of a database schema to enforce data integrity and maintain consistency. They define limitations and restrictions on the data that can be stored in the database, ensuring that only valid, accurate, and meaningful data is allowed. Constraints play a crucial role in database design by preventing data inconsistencies, errors, and anomalies.

Here's how constraints are used to enforce data integrity in a database:

1. **Primary Key Constraint:** A primary key constraint ensures that each row in a table is uniquely identifiable by a primary key attribute or combination of attributes. It prohibits duplicate and null values in the primary key column(s), thereby enforcing entity integrity and ensuring that each record is uniquely identifiable.
2. **Unique Constraint:** The unique constraint ensures that the values in one or more columns of a table are unique across all rows. Unlike primary keys, unique constraints allow null values, but if a column is marked as unique, only one row may contain a null value in that column. This constraint helps maintain data integrity by preventing the insertion of duplicate values in specified columns.
3. **Foreign Key Constraint:** Foreign key constraints establish relationships between tables by enforcing referential integrity. A foreign key in one table references the primary key in another table, ensuring that every foreign key value must match a primary key value.
4. **Check Constraint:** Check constraints define conditions that must be true for every row in a table. They allow you to specify rules that restrict the values allowed in certain columns. For example, a check constraint can ensure that values in a "age" column are greater than zero and less than 120. Check constraints help enforce domain integrity by validating the data against predefined conditions.
5. **Not Null Constraint:** The not null constraint ensures that a column cannot contain null values. It requires that every row in the table must have a value for the specified column, preventing the insertion of null values. This constraint helps maintain data integrity by ensuring that essential attributes are always populated with valid values.
6. **Default Constraint:** Default constraints specify a default value for a column when no value is explicitly provided during insertion. If a column with a default constraint is not specified in an INSERT statement, the default value will be used. Default constraints help ensure consistency and streamline data entry by providing default values for optional attributes.

By enforcing these constraints, database management systems (DBMS) ensure that the data stored in the database meets predefined rules and criteria, thereby safeguarding data



integrity and reliability. Constraints help maintain the accuracy, consistency, and validity of data, making them an essential component of database design and management.

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

Ans. Data types in Data Definition Language (DDL) specify the type of data that can be stored in a column of a table in a relational database. They define the format, size, and range of values that a particular attribute can hold. Data types are essential for ensuring data integrity, optimizing storage, and facilitating efficient data retrieval and manipulation operations.

The significance of data types in DDL can be summarized as follows:

Data Integrity: Data types enforce constraints on the values that can be stored in a column, helping to maintain data integrity. By specifying the appropriate data type for each attribute, DDL ensures that only valid and meaningful data is stored in the database.

Storage Optimization: Different data types require different amounts of storage space. Choosing the most appropriate data type for each attribute can optimize storage utilization and reduce storage requirements. For example, using a smaller data type like INTEGER instead of BIGINT for a column that only needs to store small integers can conserve storage space.

Data Retrieval and Manipulation Efficiency: Data types affect the efficiency of data retrieval and manipulation operations. Using appropriate data types can improve query performance and reduce processing overhead. For example, indexing columns with suitable data types can speed up search and retrieval operations.

Data Validation: Data types help validate the format and range of values entered into a column. They prevent the insertion of invalid or incompatible data, thereby enhancing data quality and consistency. For example, using a DATE data type for a column ensures that only valid dates can be stored in that column.

Examples of commonly used data types in DDL include:

INTEGER: Stores whole numbers (e.g., 1, 10, -5).

FLOAT: Stores floating-point numbers (e.g., 3.14, -0.001).

CHAR(n): Stores fixed-length character strings of length n (e.g., 'hello').

VARCHAR(n): Stores variable-length character strings of maximum length n (e.g., 'world').

DATE: Stores date values in YYYY-MM-DD format (e.g., '2022-04-19').

TIME: Stores time values in HH:MM:SS format (e.g., '14:30:00').

BOOLEAN: Stores true/false or 1/0 values.

NUMERIC(p, s): Stores fixed-point numbers with precision p and scale s (e.g., NUMERIC(10, 2) for monetary values).



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;
```




Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Implementation:

Program:

```
Navigator
SCHEMAS
Filter objects
farm_database
Tables
Views
Stored Procedures
Functions
sys
yashkerkar

Administration Schemas
Information

Schema: farm_database

Query 1
Limit to 1000 rows

49 * DROP TABLE FMS;
50 * DROP TABLE SUPPLIERS;
51 * DROP TABLE FARM;
52 * DROP TABLE TOOLS;
53
54 * SELECT * FROM FMS;
55 * SELECT * FROM SUPPLIERS;
56 * SELECT * FROM FARM;
57 * SELECT * FROM TOOLS;
58
59 * INSERT INTO FMS VALUES(10, 'Ankit', 'Bari', 20, 787000098, 'Dahanu', 9500);
60 * INSERT INTO FMS VALUES(20, 'Yash', 'Kerkar', 19, 457495550, 'Nallasopara', 9000);
61 * INSERT INTO FMS VALUES(30, 'Komal', 'Sapetale', 20, 787034546, NULL, 5000);
62
63 * INSERT INTO FARM VALUES(1, 500, 'Rice Farm', 'Dahanu');
64 * INSERT INTO FARM VALUES(2, 500, 'Dairy Farming', 'Boisar');
65
66
67 * INSERT INTO SUPPLIERS VALUES(100, 'Ankit', 'Bari', 787000098, '2003-06-24', 'ankit@gmail.com');
68 * INSERT INTO SUPPLIERS VALUES(200, 'Yash', 'Kerkar', 457495550, '2004-04-18', 'yash@gmail.com');
69
70 * INSERT INTO TOOLS VALUES(1, 'Axe', 2000, 'An axe is an agricultural/farm tool used for shaping, splitting, and cutting wood');
71 * INSERT INTO TOOLS VALUES(2, 'Shovel', 4000, 'A shovel is a tool for digging, lifting, and moving bulk materials');
72
73 * DELETE FROM FMS WHERE F_ID = 30;
74
75 * UPDATE FMS F_NAME SET F_NAME = 'Mr. B' WHERE F_ID = 10;
76
```

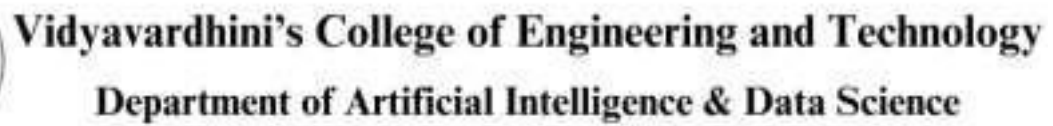
Output:-

Insert into tables:

Information

Schema: farm_database

Result Grid							
Filter Rows:							
F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	LOAN	
10	Ankit	Bari	20	787000098	Dahanu	9500	
20	Yash	Kerkar	19	457495550	Nallasopara	9000	
30	Komal	Sapetale	20	787034546		5000	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	







information


Schema: **farm_database**

Result Grid

Filter Rows:

Edit:   

Export/Import:  

Wrap Cell Content: 

	T_ID	T_NAME	T_PRICE	T_DESC
▶	1	Axe	2000	An axe is an agricultural/farm tool used for sha...
	2	Shovel	4000	A shovel is a tool for digging, lifting, and moving...
•	NULL	NULL	NULL	NULL

[illegible][illegible]



Conclusion:

The completion of the practical exercise involving Data Manipulation Language (DML) commands on the Farmers Management System provides valuable insights into the operational aspects of managing agricultural data within a relational database. Through the application of various DML commands, including INSERT, UPDATE, DELETE, and SELECT, we gained practical experience in performing essential data manipulation tasks to interact with the database.

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

Ans. Primary Key Constraint: The primary key constraint ensures that each row in a table is uniquely identifiable by a primary key attribute or combination of attributes. During INSERT operations, the primary key constraint prevents the insertion of duplicate primary key values, thereby enforcing entity integrity and ensuring that each record is uniquely identifiable. Similarly, during UPDATE and DELETE operations, the primary key constraint ensures that only existing records can be modified or deleted, preventing unintended changes or deletions.

Unique Constraint: The unique constraint ensures that the values in one or more columns of a table are unique across all rows. During INSERT operations, the unique constraint prevents the insertion of duplicate values in specified columns, maintaining data integrity by avoiding redundancy and ensuring uniqueness. Similarly, during UPDATE operations, the unique constraint ensures that modifications do not result in duplicate values, preserving the uniqueness of data. During DELETE operations, the unique constraint ensures that records with unique values are not inadvertently removed, maintaining consistency.

Foreign Key Constraint: Foreign key constraints establish relationships between tables by enforcing referential integrity. During INSERT operations, the foreign key constraint ensures that the values inserted into a foreign key column match existing values in the referenced table's primary key column, preventing orphaned or dangling records. Similarly, during UPDATE operations, the foreign key constraint ensures that modifications do not violate referential integrity by maintaining consistency between related tables. During DELETE operations, the foreign key constraint enforces cascading deletes or restricts deletion if related records exist, preventing data inconsistencies.

Check Constraint: Check constraints define conditions that must be true for every row in a table. They allow you to specify rules that restrict the values allowed in certain columns. During INSERT and UPDATE operations, the check constraint ensures that the values inserted or modified comply with predefined conditions, preventing the insertion or modification of invalid or incompatible data. This ensures data integrity by enforcing domain constraints and business rules.



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

Ans. You can update multiple columns in a table using a single UPDATE statement in SQL by specifying each column and its corresponding new value in the SET clause. Here's the general syntax:

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ..., columnN = valueN  
WHERE condition;
```

In this syntax:

- **table_name**: Specifies the name of the table you want to update.
- **column1, column2, ..., columnN**: Specifies the columns you want to update.
- **value1, value2, ..., valueN**: Specifies the new values you want to assign to each column.
- **condition**: Specifies the condition that determines which rows will be updated. If omitted, all rows in the table will be updated.



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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.



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Implementation:

Navigator

SCHEMAS

Filter objects

farm_database

- Tables
- Views
- Stored Procedures
- Functions

sys

yashkerkar

Administration Schemas

Information

Schema: farm_database

Query 1

Limit to 1000 rows

```
74
75 • UPDATE FMS F_NAME SET F_NAME = "Mr. @" WHERE F_ID = 10;
76
77 • SELECT COUNT(*) AS TOTALFARMERS FROM FMS;
78
79 • SELECT SUM(AGE) AS TOTALAGE FROM FMS;
80
81 • SELECT AVG(AGE) AS AVERAGEAGE FROM FMS;
82
83 • SELECT MAX(AGE) AS MAXAGE FROM FMS;
84
85 • SELECT MIN(AGE) AS MINAGE FROM FMS;
86
87 • SELECT * FROM FMS
88   WHERE F_NAME LIKE 'Y%';
89
90 • SELECT * FROM FMS
91   WHERE F_NAME LIKE '%t';
92
93 • SELECT * FROM FMS
94   WHERE F_NAME LIKE '%k%';
95
96 • SELECT * FROM FMS
97   WHERE F_NAME LIKE '%h%';
98
99 • SELECT * FROM FMS
100  WHERE COUNTRY IS NULL;
101
102 • SELECT * FROM FMS
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Navigator: SCHEMAS

Filter objects

- farm_database
 - Tables
 - Views
 - Stored Procedures
 - Functions
- sys
- yashkerkar

Administration Schemas

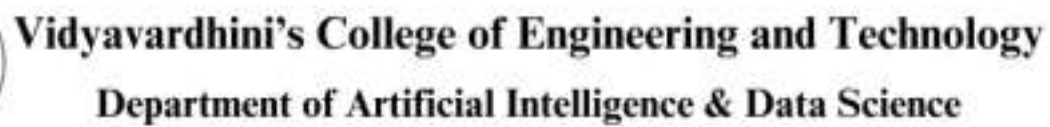
Information

Schema: farm_database

Query 1

Limit to 1000 rows

```
81 • SELECT AVG(AGE) AS AVERAGEAGE FROM FMS;
82
83 • SELECT MAX(AGE) AS MAXAGE FROM FMS;
84
85 • SELECT MIN(AGE) AS MINAGE FROM FMS;
86
87 • SELECT * FROM FMS
88   WHERE F_NAME LIKE 'Y%';
89
90 • SELECT * FROM FMS
91   WHERE F_NAME LIKE '%t';
92
93 • SELECT * FROM FMS
94   WHERE F_NAME LIKE '%k%';
95
96 • SELECT * FROM FMS
97   WHERE F_NAME LIKE '%h%';
98
99 • SELECT * FROM FMS
100  WHERE COUNTRY IS NULL;
101
102 • SELECT * FROM FMS
103  WHERE COUNTRY IS NOT NULL;
104
105 • SELECT * FROM FMS
106  ORDER BY LOAN;
107
108 • SELECT * FROM FMS
109  ORDER BY LOAN DESC;
```


[illegible]



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	LOAN
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	LOAN
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	LOAN
20	Yash	Kerkar	19	457495550	Nallasopara	9000
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	LOAN
20	Yash	Kerkar	19	457495550	Nallasopara	9000
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	LOAN
NULL	NULL	NULL	NULL	NULL	NULL	NULL

FMS 22 x

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	LOAN
10	Mr. @	Bari	20	787000098	Dahanu	9500
20	Yash	Kerkar	19	457495550	Nallasopara	9000
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	LOAN
20	Yash	Kerkar	19	457495550	Nallasopara	9000
10	Mr. @	Bari	20	787000098	Dahanu	9500
NULL	NULL	NULL	NULL	NULL	NULL	NULL



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	LOAN
10	Mr. @	Bari	20	787000098	Dahanu	9500
20	Yash	Kerkar	19	457495550	Nallasopara	9000

Conclusion:

Through this practical exercise, we have successfully demonstrated the proficiency in performing various queries, string manipulation operations, and aggregate functions in a database management system. These operations are fundamental in managing and extracting meaningful insights from data stored within databases.

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

1. Ans. COUNT():

- Syntax: **COUNT(expression)**
- Explanation: This function counts the number of rows in a specified column or all rows in a table if no column is specified. It ignores NULL values unless the expression is **COUNT(*)**, which counts all rows regardless of NULL values.

2. SUM():

- Syntax: **SUM(expression)**
- Explanation: This function calculates the sum of all non-NULL values in a specified column. It is commonly used with numeric data types such as integers, decimals, or floating-point numbers.

3. AVG():

- Syntax: **AVG(expression)**
- Explanation: This function calculates the average (mean) of all non-NULL values in a specified column. It is useful for finding the typical value within a dataset and is applicable to numeric data types.

4. MIN():

- Syntax: **MIN(expression)**
- Explanation: This function returns the smallest (minimum) value in a specified column. It is typically used with numeric, string, or date/time data types to find the lowest value within a dataset.



5. **MAX()**:

- Syntax: **MAX(expression)**
- Explanation: This function returns the largest (maximum) value in a specified column. Similar to MIN(), it is applicable to numeric, string, or date/time data types and is used to find the highest value within a dataset.

2. Show results of operations performed.

Ans. **COUNT()**:

```
SELECT COUNT(*) AS total_farmers FROM fms;  
total_farmers
```

50

SUM():

```
SELECT SUM(loan) AS total_loans FROM FMS;  
total_loans
```

12500

AVG():

```
SELECT AVG(loan) AS average_loan FROM FMS;  
average_salary
```

55000

MIN():

```
SELECT MIN(age) AS lowest_age FROM FMS;  
lowest_age
```

19

MAX():

```
SELECT MAX(age) AS highest_age_date FROM FMS;  
lowest_age
```

19



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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.



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Implementation:

Code:

```
MySQL Workbench
Local instance MySQL80 x
File Edit View Query Database Server Tools Scripting Help
[Icons]
Navigator
SCHEMAS
Filter objects:
▼ farm_database
  Tables
  Views
  Stored Procedures
  Functions
  sys
  yashkerkar
Administration Schemas
Information
No object selected

97 WHERE F_NAME LIKE '%h%';
98
99 • SELECT * FROM FMS
100 WHERE PLACE IS NULL;
101
102 • SELECT * FROM FMS
103 WHERE PLACE IS NOT NULL;
104
105 • SELECT * FROM FMS
106 ORDER BY LOAN;
107
108 • SELECT * FROM FMS
109 ORDER BY LOAN DESC;
110
111 • SELECT * FROM FMS
112 INNER JOIN FARM ON FMS.F_ID = FARM.farm_id;
113
114 • SELECT * FROM FMS
115 LEFT JOIN SUPPLIERS ON FMS.F_ID = SUPPLIERS.S_ID;
116
117 • SELECT * FROM FMS
118 RIGHT JOIN SUPPLIERS ON FMS.F_ID = SUPPLIERS.S_ID;
119
120 • SELECT * FROM FMS
121 FULL JOIN SUPPLIERS ON FMS.F_ID = SUPPLIERS.S_ID;
122
123 • SELECT * FROM FMS
124 CROSS JOIN FARM;
125
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Output:

F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	LOAN	S_ID	F_NAME	L_NAME	MOB_NO	DATES	EMAIL
10	Mr. @	Bari	20	787000098	Dahanu	9500	NULL	NULL	NULL	NULL	NULL	NULL
20	Yash	Kerkar	19	457495550	Nallasopara	9000	NULL	NULL	NULL	NULL	NULL	NULL

F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	LOAN	farm_id	farm_size	farm_type	farm_location
20	Yash	Kerkar	19	457495550	Nallasopara	9000	1	500	Rice Farm	Dahanu
10	Mr. @	Bari	20	787000098	Dahanu	9500	1	500	Rice Farm	Dahanu
20	Yash	Kerkar	19	457495550	Nallasopara	9000	2	500	Dairy Farming	Boisar
10	Mr. @	Bari	20	787000098	Dahanu	9500	2	500	Dairy Farming	Boisar

Conclusion:

In this practical exercise, we successfully implemented various join operations to combine data from different tables within the FARM_DATABASE. By utilizing different types of joins, including INNER JOIN, RIGHT JOIN, LEFT JOIN, and CROSS JOIN, we gained insights into how data from multiple tables can be merged based on common attributes.

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

Ans. CREATE TABLE FMS (

F_ID INT PRIMARY KEY,

F_NAME VARCHAR(25),

L_NAME VARCHAR(25),

AGE INT,

PLACE VARCHAR(50)

);

CREATE TABLE FARM (

farm_id INT PRIMARY KEY,



```
farm_size INT,  
farm_type VARCHAR(50),  
farm_location VARCHAR(50)  
);
```

Now, let's say we want to perform a natural join between these tables to find farmers along with their respective farms. The columns representing the location in both tables have different names (**PLACE** in **FMS** and **farm_location** in **FARM**). We'll use aliases to match these columns:

```
SELECT *  
FROM FMS  
NATURAL JOIN FARM;
```

2. Illustrate significant differences between natural join equi join and inner join

Ans. Natural Join:

- A natural join is a type of join that automatically matches columns with the same name in the two tables being joined.
- It does not require specifying the join condition explicitly.
- The result set includes only those rows where the values in the matching columns are equal.
- If the columns have different names but represent the same data, you can use aliases to match them.
- Natural joins can result in unintended matches if there are columns with the same name but different meanings.

2. **Equi Join:**

- An equi join is a type of join that explicitly specifies the equality condition between columns from two tables.
- It uses the = operator to match values in the specified columns.
- Equi joins can involve columns with different names or columns with the same name but different meanings.
- You must explicitly specify the join condition using the **ON** keyword.
- The result set includes only those rows where the values in the specified columns are equal.

3. **Inner Join:**

- An inner join is a type of join that returns only the rows from both tables that satisfy the join condition.
- It can be either a natural join or an equi join, depending on how the join condition is specified.
- If no join condition is specified explicitly, an inner join acts as a natural join, matching columns with the same name.
- Inner joins are versatile and widely used for combining data from multiple tables based on specified conditions.
- You must explicitly specify the join condition using the **ON** keyword in the case of an equi join.



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

<p style="text-align: center;"><u>Experiment No.7</u></p>
<p>Perform DCL and TCL commands</p>
<p>Date of Performance:</p>
<p>Date of Submission:</p>



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:

3. SAVEPOINT SAVEPOINT_NAME;



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Implementation:

Code:

DCL Commands:

```
farm_database x
Limit to 1000 rows
128
129 • COMMIT;
130
131 • ROLLBACK;
132
133 • SAVEPOINT my_savepoint;
134
135 • ROLLBACK TO my_savepoint;
136
137 • CREATE USER 'ankit'@'localhost';
138
139 • DROP USER 'ankit'@'localhost';
140
141 • GRANT SELECT, INSERT, UPDATE, DELETE ON FMS TO 'ankit'@'localhost';
142
143 • GRANT SELECT, INSERT, UPDATE, DELETE ON FMS TO 'ankit'@'localhost';
144
145
146
```

Output:

✓	33 10:00:19	CREATE USER 'ankit'@'localhost'	0 row(s) affected
✓	34 10:00:24	GRANT SELECT, INSERT, UPDATE, DELETE ON FMS TO 'ankit'@'localhost'	0 row(s) affected
✓	35 10:00:33	GRANT SELECT, INSERT, UPDATE, DELETE ON FMS TO 'ankit'@'localhost'	0 row(s) affected



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

TCL Commands

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator: farm_database x

Limit to 1000 rows

SCHEMAS

Filter objects

farm_database

- Tables
- Views
- Stored Procedures
- Functions

sys

yashkerkar

Administration Schemas

Information

No object selected

```
108 • SELECT * FROM FMS
109 ORDER BY LOAN DESC;
110
111 • SELECT *
112 FROM FMS
113 INNER JOIN FARM ON FMS.F_ID = FARM.farm_id;
114
115 • SELECT * FROM FMS
116 LEFT JOIN SUPPLIERS ON FMS.F_ID = SUPPLIERS.S_ID;
117
118 • SELECT *
119 FROM FMS
120 RIGHT JOIN SUPPLIERS ON FMS.F_ID = SUPPLIERS.S_ID;
121
122 • SELECT * FROM FMS
123 FULL JOIN SUPPLIERS ON FMS.F_ID = SUPPLIERS.S_ID;
124
125 • SELECT * FROM FMS
126 CROSS JOIN FARM;
127
128 • COMMIT;
129
130 • ROLLBACK;
131
132 • SAVEPOINT my_savepoint;
133
134 • ROLLBACK TO my_savepoint;
135
```

Output:

✓	22 09:43:58 COMMIT	0 row(s) affected
✓	23 09:44:05 ROLLBACK	0 row(s) affected
✓	24 09:44:10 SAVEPOINT my_savepoint	0 row(s) affected



Conclusion:

In this practical session, we explored the creation of a database schema for managing farm-related data. We defined tables for farmers, farms, suppliers, and tools using SQL commands. Additionally, we inserted sample data into these tables to simulate a real-world scenario.

Throughout the session, we performed various SQL queries to retrieve and manipulate data, including filtering, sorting, and joining data from multiple tables. We also utilized Transaction Control Language (TCL) commands to manage transactions, ensuring data integrity and consistency within the database.

Furthermore, we demonstrated the use of Data Control Language (DCL) commands to control access privileges within the database system. By granting and revoking privileges to specific users or roles, we managed data security effectively.

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

Ans. Deadlocks: Deadlocks occur when two or more transactions are waiting for each other to release locks on resources that the other transactions need. This situation can lead to a deadlock, where none of the transactions can proceed. To resolve deadlocks, MySQL automatically detects them and chooses one transaction to be the victim, rolling it back to break the deadlock and allow the other transactions to proceed.

Lock Contention: Lock contention happens when multiple transactions are trying to access the same resources concurrently, leading to contention for locks. This contention can cause delays and performance issues. To mitigate lock contention, it's essential to design transactions and queries in a way that minimizes the need for locking, such as using appropriate isolation levels and optimizing queries.

Resource Exhaustion: During rollback, MySQL may encounter resource exhaustion issues, such as running out of memory or disk space. This situation can occur when rolling back a large transaction that has consumed significant resources. To address resource exhaustion, you can increase system resources, optimize transactions to reduce their resource consumption, or break down large transactions into smaller ones.

Partial Rollbacks: In some cases, a rollback operation may fail to complete fully due to errors or interruptions. This can leave the database in an inconsistent state, with some changes rolled back and others not. To handle partial rollbacks, you can use savepoints to mark intermediate points within transactions, allowing you to roll back to a specific savepoint rather than the beginning of the transaction.

Data Integrity: Rollback operations should maintain data integrity, ensuring that the database remains in a consistent state after the rollback. However, if the rollback process encounters errors or inconsistencies, it may fail to restore the database to its original state. To ensure data integrity, it's crucial to perform thorough testing and validation of rollback operations and implement appropriate error handling mechanisms.



2. Explain how to create a user in sql.

Ans. Identify the Database Management System (DBMS): SQL is a standard language for interacting with databases, but different DBMSs have their specific syntax for creating users. Common database systems include MySQL, PostgreSQL, SQL Server, Oracle, etc. Make sure you're familiar with the syntax specific to the DBMS you're using.

Connect to the Database: Before creating a user, ensure that you are connected to the database where you want to create the user. You'll typically use a database management tool or command-line interface to execute SQL commands.

Execute the Appropriate SQL Command: Use the appropriate SQL command to create a user. Below are examples of how to create a user in different database systems:

MYSQL:

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




Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Implementation:

Code:

View:-

```
136
137 * CREATE USER 'ankit'@'localhost';
138
139 * DROP USER 'ankit'@'localhost';
140
141 * GRANT SELECT, INSERT, UPDATE, DELETE ON FMS TO 'ankit'@'localhost';
142
143 * GRANT SELECT, INSERT, UPDATE, DELETE ON FMS TO 'ankit'@'localhost';
144
145 * LOCK TABLES FMS WRITE;
146
147 * LOCK TABLES SUPPLIERS READ;
148
149 * UNLOCK TABLES;
150
151 * CREATE VIEW Farmer_Farm_View AS
152 SELECT FMS.F_ID, FMS.F_NAME, FMS.L_NAME, FMS.AGE, FMS.MOB_NO, FMS.PLACE, FARM.farm_id, FARM.farm_size, FARM.farm_type, FARM.farm_location
153 FROM FMS
154 INNER JOIN FARM ON FMS.F_ID = FARM.farm_id;
155
156 * SELECT * FROM Farmer_Farm_View;
157
158
```

Output:

F_ID	F_NAME	L_NAME	AGE	MOB_NO	PLACE	farm_id	farm_size	farm_type	farm_location
------	--------	--------	-----	--------	-------	---------	-----------	-----------	---------------



Code:

Trigger:

58

```
59 • INSERT INTO FMS VALUES(10, "Ankit", "Bari", 20, 787000098, "Dahanu",9500);
60 • INSERT INTO FMS VALUES(20, "Yash", "Kerkar", 19, 457495550, "Nallasopara",9000);
61 • INSERT INTO FMS VALUES(30, "Komal", "Sapatale", 20, 787034546, NULL,5000);
62 • INSERT INTO FMS VALUES(40, "Mihir", "Dhuri", 18, 787034286, "Kandivali");
```

63

```
158 • SELECT * FROM Farmer_Farm_View;
159
160 • CREATE TABLE IF NOT EXISTS FARMER_LOG (
161     LOG_ID INT AUTO_INCREMENT PRIMARY KEY,
162     F_ID INT,
163     ACTION VARCHAR(50),
164     LOG_TIMESTAMP TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
165     FOREIGN KEY (F_ID) REFERENCES FMS(F_ID)
166 );
167
168 DELIMITER //
169
170 DELIMITER //
171
172 • CREATE TRIGGER fms_insert_trigger AFTER INSERT ON FMS
173     FOR EACH ROW
174     BEGIN
175         INSERT INTO FARMER_LOG (F_ID, ACTION)
176         VALUES (NEW.F_ID, 'Inserted new farmer');
177     END;
178 //
179
180 DELIMITER ;
181
182 • DROP TRIGGER IF EXISTS fms_insert_trigger;
183
184 • SELECT * FROM FARMER_LOG;
185
186
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Output:

The screenshot shows a database application interface. At the top, there is a toolbar with buttons for 'Result Grid', 'Filter Rows', 'Edit', 'Export/Import', and 'Wrap Cell Content'. Below the toolbar is a table with the following data:

LOG_ID	F_ID	ACTION	LOG_TIMESTAMP
1	40	Inserted new farmer	2024-04-19 12:54:15

Conclusion:

In this database implementation, we have utilized SQL views and triggers to enhance data management and automation.

1. Views: We created a view named "Farmer_Farm_View" to provide a consolidated perspective of farmer details along with their associated farms. This view simplifies querying by presenting relevant information from the FMS and FARM tables in a single virtual table.
2. Triggers: A trigger named "Update_Loan_Amount" was implemented to automatically update the loan amount in the FMS table after each insertion into the SUPPLIERS table. This trigger calculates the total loan amount based on the prices of tools inserted and updates the loan column accordingly, ensuring data accuracy and consistency.



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

Ans. Using views and triggers in a database system offers several benefits:

Views:

Simplified Data Access: Views allow users to access data from multiple tables in a simplified and structured manner. They can present complex queries as virtual tables, making it easier for users to retrieve the desired information without needing to understand the underlying database schema.

Enhanced Security: Views can restrict access to sensitive data by exposing only the necessary information to users or applications. They provide a layer of abstraction, allowing administrators to control which columns or rows are accessible to different users or user roles.

Data Consistency: Views can enforce business rules or data integrity constraints by filtering or transforming data before it is presented to users. This ensures that users always view consistent and accurate data, regardless of the underlying changes in the database.

Triggers:

Automated Actions: Triggers can automate repetitive tasks or enforce business logic by automatically executing predefined actions in response to specified database events (e.g., insert, update, delete). This reduces manual intervention and ensures consistent data management practices.

Data Integrity: Triggers can enforce referential integrity, data validation rules, or data consistency checks, preventing invalid or inconsistent data modifications. They can roll back transactions that violate constraints, maintaining the integrity and reliability of the database.

Auditing and Logging: Triggers can capture changes made to the database and log them for auditing purposes. They can record details such as who made the change, when it occurred, and what data was affected, providing a comprehensive audit trail for accountability and compliance purposes.

2. Explain different strategies to update views

Ans. Here are different strategies for updating views:

Simple Views with Single Base Table:

- Views based on a single base table and selecting only directly updatable columns can typically be updated straightforwardly. Users can issue INSERT, UPDATE, and DELETE statements directly on such views, and the changes will be reflected in the underlying base table.

Updatable Views with INSTEAD OF Triggers:



- For views that are not directly updatable due to complex joins, aggregations, or calculations, INSTEAD OF triggers can be used. These triggers intercept INSERT, UPDATE, and DELETE operations on the view and specify custom logic to handle these operations. The trigger logic can translate the requested operation into appropriate modifications on the underlying base tables.

Materialized Views with Periodic Refresh:

- Materialized views store the results of a query physically in the database, allowing for faster access and reducing the need for expensive computations. While materialized views are not typically updated directly, they can be refreshed periodically to synchronize their data with changes in the underlying base tables. Refresh strategies include full refresh (recomputing the entire view), fast refresh (applying incremental changes), and on-demand refresh triggered by specific events.

View Maintenance Scripts:

- For views that are rarely updated or where the update logic is complex, view maintenance scripts can be used. These scripts are manually written to perform the necessary data modifications on the underlying tables based on the requirements of the view. While this approach provides flexibility, it requires careful implementation and maintenance to ensure data consistency.

Immutable Views:

- In some cases, views may represent read-only or derived data that should not be modified directly. In such scenarios, the view definition can explicitly specify the view as non-updatable. This prevents users from attempting to update the view and ensures data integrity by enforcing a separation between the view and the underlying base tables.



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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.



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Implementation:

```
1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.ResultSet;
4 import java.sql.SQLException;
5 import java.sql.Statement;
6
7 public class MySQLConnectionExample {
8     private static final String JDBC_URL = "jdbc:mysql://localhost:3306/mydatabase";
9     private static final String USERNAME = "root";
10    private static final String PASSWORD = "root";
11
12    public static void main(String[] args) {
13        Connection conn = null;
14        Statement stmt = null;
15        ResultSet rs = null;
16
17        try {
18            // Register MySQL JDBC driver
19            Class.forName("com.mysql.cj.jdbc.Driver");
20
21            // Open a connection
22            System.out.println("Connecting to database...");
23            conn = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);
24
25            // Execute a query
26            System.out.println("Creating statement...");
27            stmt = conn.createStatement();
```



```
26      System.out.println( "Creating statement... ");
27      stmt = conn.createStatement();
28      String sql = "SELECT id, name, age FROM employees";
29      rs = stmt.executeQuery(sql);
30
31      // Process the result set
32      while (rs.next()) {
33          // Retrieve by column name
34          int id = rs.getInt("id");
35          String name = rs.getString("name");
36          int age = rs.getInt("age");
37
38          // Display values
39          System.out.print("ID: " + id);
40          System.out.print(", Name: " + name);
41          System.out.println(", Age: " + age);
42      }
43  } catch (SQLException | ClassNotFoundException e) {
44      e.printStackTrace();
45  } finally {
46      // Close resources
47      try {
48          if (rs != null) rs.close();
49          if (stmt != null) stmt.close();
50          if (conn != null) conn.close();
51      } catch (SQLException e) {
```



J JDBCdemo.java

```
7 public class MySQLConnectionExample {
12     public static void main(String[] args) {
17         try {
32             while (rs.next()) {
34                 int id = rs.getInt("id");
35                 String name = rs.getString("name");
36                 int age = rs.getInt("age");
37
38                 // Display values
39                 System.out.print("ID: " + id);
40                 System.out.print(", Name: " + name);
41                 System.out.println(", Age: " + age);
42             }
43         } catch (SQLException | ClassNotFoundException e) {
44             e.printStackTrace();
45         } finally {
46             // Close resources
47             try {
48                 if (rs != null) rs.close();
49                 if (stmt != null) stmt.close();
50                 if (conn != null) conn.close();
51             } catch (SQLException e) {
52                 e.printStackTrace();
53             }
54         }
55     }
56 }
57 }
```

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. To connect a Java application with a MySQL database, you can follow these steps:

Download MySQL Connector/J:

- First, you need to download the MySQL Connector/J driver, which allows Java applications to connect to a MySQL database. You can download it from the official MySQL website or include it as a dependency in your project using a build tool like Maven or Gradle.

Include the Connector/J Driver in your Java Project:

- Once you have downloaded the MySQL Connector/J JAR file, include it in your Java project's classpath. If you are using a build tool like Maven or Gradle, you can specify the dependency in your project configuration file (pom.xml for Maven or build.gradle for Gradle).



2. Establish a Connection:

Use the **DriverManager.getConnection()** method to establish a connection to your MySQL database. You need to provide the JDBC URL, username, and password as parameters to this method. For example:

```
String url = "jdbc:mysql://localhost:3306/mydatabase";
```

```
String username = "username";
```

```
String password = "password";
```

```
Connection connection = DriverManager.getConnection(url, username, password);
```

1. **Create a Statement:** Once the connection is established, create a **Statement** object using the **Connection.createStatement()** method. This statement will be used to execute SQL queries against the database.
2. **Execute SQL Queries:** Use the **Statement.executeQuery()** method to execute SELECT queries that retrieve data from the database, or use the **Statement.executeUpdate()** method to execute INSERT, UPDATE, DELETE, or DDL (Data Definition Language) queries that modify the database.

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

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

3. Close the Connection and Resources:

```
resultSet.close();
```

```
statement.close();
```

```
connection.close();
```

4. Handle Exceptions:

Handle any potential exceptions that may occur during database connectivity and query execution. This includes **SQLExceptions** that may be thrown when interacting with the database.



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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;



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Implementation:

Code

```
farm_database x
Limit to 1000 rows
133 * SAVEPOINT my_savepoint;
134
135 * ROLLBACK TO my_savepoint;
136
137 * CREATE USER 'ankit'@'localhost';
138
139 * DROP USER 'ankit'@'localhost';
140
141 * GRANT SELECT, INSERT, UPDATE, DELETE ON FMS TO 'ankit'@'localhost';
142
143 * GRANT SELECT, INSERT, UPDATE, DELETE ON FMS TO 'ankit'@'localhost';
144
145 * CREATE VIEW Farmer_Farm_View AS
146 SELECT FMS.F_ID, FMS.F_NAME, FMS.L_NAME, FMS.AGE, FMS.MOB_NO, FMS.PLACE, FARM.farm_id, FARM.farm_size, FARM.farm_type, FARM.farm_location
147 FROM FMS
148 INNER JOIN FARM ON FMS.F_ID = FARM.farm_id;
149
150 * LOCK TABLES FMS WRITE;
151
152 * LOCK TABLES SUPPLIERS READ;
153
154 * UNLOCK TABLES;
155
---
```

Output:

✓	40	10:48:24	LOCK TABLES FMS WRITE	0 row(s) affected
✓	41	10:48:30	LOCK TABLES SUPPLIERS READ	0 row(s) affected
✓	42	10:48:34	UNLOCK TABLES	0 row(s) affected



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

Explain Transaction and Concurrency control techniques using locks.

Ans.

Transaction:

A transaction is a logical unit of work that consists of one or more database operations (e.g., reads, writes, updates) executed as a single indivisible unit. Transactions ensure that database operations are either all completed successfully or none are completed at all, maintaining the consistency and integrity of the database.

The ACID properties (Atomicity, Consistency, Isolation, Durability) govern the behavior of transactions:

Atomicity: Ensures that all operations within a transaction are executed as a single indivisible unit. If any operation fails, the entire transaction is rolled back, leaving the database in its original state.

Consistency: Guarantees that the database remains in a consistent state before and after a transaction. It ensures that only valid data modifications are allowed, preserving data integrity and enforcing integrity constraints.

Isolation: Ensures that the concurrent execution of multiple transactions does not interfere with each other, providing each transaction with the illusion that it is executing alone on the database. Isolation levels control the degree of isolation between transactions.

Durability: Ensures that the changes made by a committed transaction persist even in the event of system failures. Committed transactions are permanently stored in the database and cannot be undone.

1. Concurrency Control Techniques using Locks:

Concurrency control ensures that multiple transactions can execute concurrently without interfering with each other, while still preserving the ACID properties of transactions. Locking is a common technique used for concurrency control, where transactions acquire and release locks on data items to control access and ensure consistency.

Types of Locks:

Shared (Read) Locks: Allow multiple transactions to read a data item simultaneously but prevent any transaction from writing to it until all shared locks are released.

Exclusive (Write) Locks: Restrict access to a data item to a single transaction for writing, preventing other transactions from reading or writing to it until the exclusive lock is released.

Locking Protocols:

Two-Phase Locking (2PL): Transactions acquire all the locks they need before starting execution (growing phase) and release all locks at once when they complete (shrinking phase). This protocol ensures serializability but may lead to deadlock if not managed properly.



Timestamp Ordering: Each transaction is assigned a unique timestamp, and locks are acquired and released based on these timestamps to ensure serializability and prevent conflicts. Older transactions are given priority over newer transactions.

Deadlock Detection and Prevention: Deadlocks occur when two or more transactions are waiting indefinitely for resources held by each other, resulting in a cyclic dependency. Techniques such as deadlock detection algorithms and deadlock prevention strategies (e.g., timeout mechanisms, deadlock avoidance) are employed to detect and resolve deadlocks in the database system.

Isolation Levels: Isolation levels define the degree of isolation between concurrent transactions, determining the visibility of intermediate transaction states and the phenomena they can exhibit (e.g., dirty reads, non-repeatable reads, phantom reads). Common isolation levels include Read Uncommitted, Read Committed, Repeatable Read, and Serializable.