-- STATEMENT TO CREATE THE CONTACT TABLE --

create database tops;

use tops;

CREATE TABLE CONTACT

(

ContactID INT,

CompanyID INT,

FirstName VARCHAR(45),

LastNmae varchar(45),

Street varchar(45),

City varchar(45),

state varchar(2),

zip varchar(10),

ismain boolean,

Email varchar(45),

Phone varchar(45)

);

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------

-- 2) statement ceate the employee table --

create table Employee

(

EmployeeID INT primary key,

FirstName VARCHAR(45),

LastNmae varchar(45),

Salary decimal(10.2),

HireDate date,

JobTittle varchar(25),

Email varchar(45),

Phone varchar(20)

);

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------

/\* statement to create contactemployee table

HINT : Use DATE as the datatype for ContactDate .it allows you to store the

date in this formal :YYYY-MM--DD(ex=2014-03-12 for march march 12 2014);

\*/

create table ContactEmployee

(

ContactEmployeeId int primary key,

ContactID int,

EmployeeID int,

ContactDate date,

Descripition varchar(100)

);

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

update Employee set phone ='215-555-8800' where EmployeeID=2;

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

-- 5) In the company table,the statement that change the name of "urban outfitters inc" to "urban outfittters" ?

create table Company

(

CompanyID int primary key,

CompanyName varchar(45),

Street varchar(45),

City varchar(45),

State varchar(2),

Zip varchar(10)

);

update Company set CompanyName ="urban outfitters" where CompanyId=14;

----------------------------------------------------------------------------------------------------------------------------------------------------------------------

-- 6) In the ContactEmployee table ,the statement that removes Dianne Connor's contact event with Jack Lee(one statement) ?

-- HINT Use the primary of the ContactEmployee table of specify the correct the correct record to remove.

set sql\_safe\_updates =0;

delete from ContactEmployee where ContactID =101 and EmployeeID = 201 ;

---------------------------------------------------------------------------------------------------------------------------------------------------------------------

/\* 7)Write the SQL SELECT query that displays the names of the employee that have have contacted Toll brothers (one statement) .Run the SQL select query in

MYSQL Workbeanch .copy the results below as well . \*/

select \* from employee left join Contact on

employee.EmployeeID =contact.ContactID;

SELECT \*

FROM employee

LEFT JOIN contact

ON contact.ContactID = employee.EmployeeID

AND contact.FirstName = employee.FirstName

AND contact.LastName = employee.LastName

AND contact.Email = employee.Email

AND contact.Phone = employee.Phone;

------------------------------------------------------------------------------------------------------------------------------------------------------------------------

-- 8) what is the significance of "%" "AND" "\_" operators in the LIKE statement ?

/\* In SQL, the LIKE operator is used to search for a specified pattern in a column. It is often used with the % and \_ wildcards to allow more flexible and

complex pattern matching.

1. % (Percent Sign) Wildcard

The % wildcard represents zero, one, or more characters. It is used to match any sequence of characters (including no characters).

Example:

SELECT \* FROM employees

WHERE name LIKE 'A%';

This will return all employees whose names start with "A", such as "Alice", "Alex", or "Avery", because % matches any number of characters.

Example:

SELECT \* FROM employees

WHERE name LIKE '%son';

This will return all employees whose names end with "son", such as "Jackson", "Mason", or "Wilson", because % can match any characters before "son".

Example :

SELECT \* FROM employees

WHERE name LIKE '%e%';

This will return all employees whose names contain the letter "e" anywhere in the name, because % matches any characters before and after "e".

2. \_ (Underscore) Wildcard

The \_ wildcard represents exactly one character. It is used to match a single character in a specific position.

Example:

SELECT \* FROM employees

WHERE name LIKE 'A\_ice';

This will return all employees whose names are 5 characters long and start with "A" and end with "ice". For example, it will match "Alice", but not "Avery" or "Alec",

because the underscore represents exactly one character.

Example:

SELECT \* FROM employees

WHERE name LIKE 'A\_\_e';

his will return all employees whose names are 4 characters long and start with "A" and end with "e". For example, it will match "Abe", but not "Alice",

because it expects exactly two characters between "A" and "e". /\*

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------

-- 9) Explain normalization in the context of database .

ans = Normalization is a method to breaking down a large ,complex table into smaller related tables and define relationship between them .It reduce the redency of data

in an table also removes inconsistency problem and unwanted risk.

1. First Normal Form (1NF):

A table is in 1NF if:

All columns contain atomic (indivisible) values. This means that each column should have a single value for each row, and the values must be of the same type.

There are no repeating groups or arrays within a column.

Example:

Student\_ID Name Subjects

1 Alice Math, Science

2 Bob English, History

In 1NF, the table should be structured like this:

Student\_ID Name Subject

1 Alice Math

1 Alice Science

2 Bob English

2 Bob History

2. Second Normal Form (2NF):

A table is in 2NF if:

It is in 1NF.

It does not have any partial dependency, meaning that all non-key columns must depend on the entire primary key, not just part of it.

For tables with a composite primary key (a key made of more than one column), this eliminates cases where non-key attributes are dependent on only a part of the key.

Example: Consider a table with a composite primary key (Student\_ID, Subject) and non-key attributes (Teacher\_Name):

Student\_ID Subject Teacher\_Name

1 Math Mr. Smith

1 Science Mrs. Johnson

2 English Mr. Lee

In this case, the teacher depends on the subject, not on the student. To bring the table into 2NF, we separate the teacher information into a different table:

Student-Subject Table:

Student\_ID Subject

1 Math

1 Science

2 English

Teacher-Subject Table:

Subject Teacher\_Name

Math Mr. Smith

Science Mrs. Johnson

English Mr. Lee

3. Third Normal Form (3NF):

A table is in 3NF if:

It is in 2NF.

There is no transitive dependency, meaning that non-key attributes do not depend on other non-key attributes.

This ensures that non-key attributes are only dependent on the primary key.

Example: Consider a table:

Employee\_ID Name Department Department\_Location

1 John HR New York

2 Alice IT San Francisco

In this case, the Department\_Location depends on the Department, not directly on the Employee\_ID. To bring the table into 3NF, we remove this transitive dependency:

Employee Table:

Employee\_ID Name Department

1 John HR

2 Alice IT

Department Table:

Department Location

HR New York

IT San Francisco

Higher Normal Forms:

While 1NF, 2NF, and 3NF are the most commonly used normal forms, there are also:

Boyce-Codd Normal Form (BCNF): A stronger version of 3NF.

Fourth Normal Form (4NF): Deals with multi-valued dependencies.

Fifth Normal Form (5NF): Deals with join dependencies and ensures that data is split in a way that no further decomposition is possible without losing information.

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------

-- 10) What does a join in MySQL mean ?

ANS = MySQL, a join is a statement that combines data from multiple tables in a database based on shared column values:

Purpose: Joins allow you to retrieve data from multiple tables with a single query.

How it works: Joins use common column values, also known as the join key, to link data from different tables.

Where it's used: Joins can be used in SELECT, UPDATE, and DELETE statements.

Types of joins: MySQL supports three basic types of joins: inner, outer, and cross. Outer joins can be further broken down into left outer and right outer joins.

Syntax: In MySQL, JOIN, CROSS JOIN, and INNER JOIN are syntactic equivalents.

Example: Here's an example of an INNER JOIN in MySQL:

Code

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate

FROM Orders.INNER JOIN Customers ON Orders

-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

11) What do you understand about DDL ,DCL , and DML in MySQL ?

ans= In MySQL (and other relational database management systems), DDL (Data Definition Language), DML (Data Manipulation Language), and DCL (Data Control Language) are three important categories of SQL commands. Each category is used for different purposes when interacting with a database.

1. DDL (Data Definition Language)

DDL is used to define and manage the structure of the database, including the creation, alteration, and deletion of database objects like tables, indexes, and schemas.

Common DDL commands include:

CREATE: Used to create database objects like tables, views, or databases.

CREATE TABLE employees (id INT, name VARCHAR(100), salary DECIMAL(10, 2));

ALTER: Used to modify an existing database object, such as a table structure.

ALTER TABLE employees ADD COLUMN department VARCHAR(50);

DROP: Used to delete database objects like tables or views.

DROP TABLE employees;

TRUNCATE: Removes all data from a table but does not delete the table structure.

TRUNCATE TABLE employees;

2. DML (Data Manipulation Language)

DML is used to manipulate and manage the data stored within the database. DML commands deal with the retrieval, insertion, updating, and deletion of data.

Common DML commands include:

SELECT: Used to retrieve data from one or more tables.

SELECT \* FROM employees WHERE department = 'HR';

INSERT: Used to add new rows of data to a table.

INSERT INTO employees (id, name, salary) VALUES (1, 'John Doe', 50000.00);

UPDATE: Used to modify existing data within a table.

UPDATE employees SET salary = 55000 WHERE id = 1;

DELETE: Used to remove rows of data from a table.

DELETE FROM employees WHERE id = 1;

3. DCL (Data Control Language)

DCL is used to control access to data stored in a database. These commands are typically used for setting permissions or granting/revoking access rights to users.

Common DCL commands include:

GRANT: Used to provide users with specific permissions or privileges.

GRANT SELECT, INSERT ON employees TO 'user1'@'localhost';

REVOKE: Used to remove previously granted privileges.

REVOKE INSERT ON employees FROM 'user1'@'localhost';

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

12) What is the role of the MySQL JOIN in clause in a query ,and what are some common types of join ?

Ans =The JOIN clause in MySQL is used to combine rows from two or more tables based on a related column between them. It is essential for querying relational databases where data is often stored across multiple tables, and you need to retrieve it in a meaningful way by linking related information.

Purpose of the JOIN Clause:

Combining Data: It allows you to combine data from multiple tables in a single result set.

Establishing Relationships: It helps in linking tables based on common columns (such as primary and foreign keys).

Efficient Queries: It helps in creating more complex and efficient queries, retrieving necessary data across multiple tables without redundancy.

Common Types of JOINs in MySQL:

INNER JOIN:

Description: The INNER JOIN returns only the rows where there is a match in both tables. If there is no match, those rows are excluded from the result set.

Syntax:

SELECT columns

FROM table1

INNER JOIN table2

ON table1.column = table2.column;

Example:

SELECT employees.name, departments.name

FROM employees

INNER JOIN departments

ON employees.department\_id = departments.id;

This returns only employees who have a matching department.

LEFT JOIN (or LEFT OUTER JOIN):

Description: The LEFT JOIN returns all rows from the left table, and the matching rows from the right table. If there is no match, NULL values are returned for columns of the right table.

Syntax:

SELECT columns

FROM table1

LEFT JOIN table2

ON table1.column = table2.column;

Example:

SELECT employees.name, departments.name

FROM employees

LEFT JOIN departments

ON employees.department\_id = departments.id;

This returns all employees, even if they don't belong to a department.

RIGHT JOIN (or RIGHT OUTER JOIN):

Description: The RIGHT JOIN returns all rows from the right table, and the matching rows from the left table. If there is no match, NULL values are returned for columns of the left table.

Syntax:

SELECT columns

FROM table1

RIGHT JOIN table2

ON table1.column = table2.column;

Example:

SELECT employees.name, departments.name

FROM employees

RIGHT JOIN departments

ON employees.department\_id = departments.id;

This will return all departments, even those without any employees.

FULL JOIN (or FULL OUTER JOIN):

Description: MySQL does not support FULL JOIN directly, but the same result can be achieved by combining LEFT JOIN and RIGHT JOIN. A FULL JOIN returns all rows from both tables, with NULLs for non-matching rows.

Example (simulated using UNION):

SELECT employees.name, departments.name

FROM employees

LEFT JOIN departments

ON employees.department\_id = departments.id

UNION

SELECT employees.name, departments.name

FROM employees

RIGHT JOIN departments

ON employees.department\_id = departments.id;

This returns all employees and departments, including those without matches.

CROSS JOIN:

Description: The CROSS JOIN produces a Cartesian product of the two tables, returning all possible combinations of rows between the tables. This join does not require a condition.

Syntax:

SELECT columns

FROM table1

CROSS JOIN table2;

Example:

SELECT employees.name, projects.name

FROM employees

CROSS JOIN projects;

This returns every combination of employee and project, regardless of any existing relationship.

SELF JOIN:

Description: A self join is a join where a table is joined with itself. This can be useful for hierarchical or recursive relationships.

Syntax:

SELECT a.column, b.column

FROM table a

JOIN table b

ON a.column = b.column;

Example:

sql

Copy code

SELECT a.employee\_name, b.employee\_name

FROM employees a

JOIN employees b

ON a.manager\_id = b.employee\_id;

This returns pairs of employees and their respective managers.