1. **Demonstration of creating database and table.**

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

2. **Demonstrate to INSERT, UPDATE, and DELETE Records in Table.**

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

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode,
Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

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

Update Single Record-

```
UPDATE Customers
SET ContactName = 'Aakash Jain', City= 'Dhule'
WHERE CustomerID = 1;
```

Update Multiple Record-

```
UPDATE Customers
SET ContactName='Amol'
WHERE Country='Jalgaon';
```

DELETE FROM *table_name* WHERE *condition*;

```
DELETE FROM Customers WHERE CustomerName='Aakash Patil';
```

```
DELETE FROM Customers;
```

3. **Demonstrate to Alter Table**

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

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE *table_name*
ADD *column_name datatype*;

```
ALTER TABLE Customers
ADD Email varchar(255);
```

ALTER TABLE - DROP COLUMN

ALTER TABLE *table_name*
DROP COLUMN *column_name*;

```
ALTER TABLE Customers
DROP COLUMN Email;
```

ALTER TABLE - ALTER/MODIFY COLUMN

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

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

```
ALTER TABLE Persons
ADD DateOfBirth date;
```

```
ALTER TABLE Persons
ALTER COLUMN DateOfBirth year;
```

```
ALTER TABLE Persons
DROP COLUMN DateOfBirth;
```

4. Defining different types of database constraint. Create table with various constraints as PRIMARY KEY, FOREIGN KEY, and CHECK & NOT NULL Constraint

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
```

```
    Age int,
    PRIMARY KEY (ID)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

SQL FOREIGN KEY Constraint

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

```
CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
    OrderNumber int NOT NULL,
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
```

```
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

SQL NOT NULL Constraint

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

5. Query based on operators and joins • Simple and nested query

SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

look at a selection from the "Orders" table:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

INNER JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2*
ON *table1.column_name* = *table2.column_name*;

SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

LEFT JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
LEFT JOIN *table2*
ON *table1.column_name* = *table2.column_name*;

SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;

SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

RIGHT JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
RIGHT JOIN *table2*
ON *table1.column_name* = *table2.column_name*;

SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;

SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Tip: FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
FULL OUTER JOIN *table2*
ON *table1.column_name* = *table2.column_name*
WHERE *condition*;

SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;

SQL Self Join

A self join is a regular join, but the table is joined with itself.

Self Join Syntax

SELECT *column_name(s)*
FROM *table1 T1, table1 T2*
WHERE *condition*;

SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;

The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

UNION Syntax

SELECT *column_name(s)* FROM *table1*
UNION
SELECT *column_name(s)* FROM *table2*;

SELECT * FROM Products

WHERE Price = 18;

SELECT * FROM Products

WHERE Price <> 18;

SELECT * FROM Products

WHERE Price BETWEEN 50 AND 60;

SELECT * FROM Products

WHERE Price > SOME (SELECT Price FROM Products WHERE Price > 20);

SELECT SupplierName

FROM Suppliers

WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20);

SELECT * FROM Customers

WHERE City = "Dhule" AND Country = "Pune";

6. Write down SQL by using i. WHERE Clause ii. GROUP BY ii. HAVING CLAUSE

The SQL WHERE Clause

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

WHERE Syntax

SELECT *column1*, *column2, ...*
FROM *table_name*
WHERE *condition*;

SELECT * FROM Customers
WHERE Country='Dhule';

SELECT * FROM Customers
WHERE CustomerID=1;

The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions

(COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

GROUP BY Syntax

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
ORDER BY *column_name(s);*

SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;

SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;

The SQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

HAVING Syntax

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
HAVING *condition*
ORDER BY *column_name(s);*

SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;

SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;

7. **Write down SQL by using i. Aggregate functions ii. Date functions iii. String functions**

The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN() Syntax

SELECT MIN(*column_name*)
FROM *table_name*
WHERE *condition*;

MAX() Syntax

SELECT MAX(*column_name*)
FROM *table_name*
WHERE *condition*;

SELECT MIN(Price) AS SmallestPrice
FROM Products;

SELECT MAX(Price) AS LargestPrice
FROM Products;

The SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criterion.

COUNT() Syntax

SELECT COUNT(*column_name*)
FROM *table_name*
WHERE *condition*;

The SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criterion.

COUNT() Syntax

SELECT COUNT(*column_name*)
FROM *table_name*
WHERE *condition*;

SUM() Syntax

SELECT SUM(*column_name*)
FROM *table_name*
WHERE *condition*;

SELECT COUNT(ProductID)

```
FROM Products;


SELECT AVG(Price)
FROM Products;

SELECT SUM(Quantity)
FROM OrderDetails;
```

SQL Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

SQL Server comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: a unique number

Note: The date types are chosen for a column when you create a new table in your database!

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```