1)WHAT IS RDMS?

The software used to store, manage, query, and retrieve data stored in a relational database is called a relational database management system (RDBMS).

==========

2)What is mysql?

MySQL is an open-source relational database management system (RDBMS)

==========

3)What are the types of languages in DBMS?

- **DQL**: Data Query Language
  e.g. SELECT.

- **DML**: Data Manipulation Language
  e.g. INSERT, UPDATE, DELETE.

- **DDL**: Data Definition Language
  e.g. CREATE, ALTER, DROP, RENAME.

- **DCL**: Data Control Language
  e.g. CREATE USER, GRANT, REVOKE.

- **TCL**: Transaction Control Language
  e.g. SAVEPOINT, COMMIT, ROLLBACK.

==========

4)What Is DQL Explain with Example?

DQL language Only fire Queries in **database Used To fetch record from database. It does not update anything or delete any record in database and also it has no role in structure too.**

Select queries comes under this language. Which is very powerful command. That is In simple It won't change anything in the database related to data and structure. and fetches only relevant data we need.

Example:

1)If we have Employee table and we need to fetch all record then we just fire query like

Select * from employee.

2)If we have same table and we want all record on the basis of the given employee_id like if we need a record of employee_id 210 then we fire query like

Select * from Employee where employee_id=210.

==========

5)What is the purpose of DML Language With example?

Data manipulation language makes the user able to manipulate data in a relational database.

MANIPULATION : NEED TO TO CHNAGE

We can perform the following operations using DML language:

COMMANDS : INSERT / UPDATE / DELETE

1)Insert data into the database through the INSERT command.

1.1)Specify both the column names and the values to be inserted:

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

Example:

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

1.2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

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

2)Update data in the database through the UPDATE command.

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

3)Delete data from the database through the DELETE command.

```
DELETE FROM table_name WHERE condition;
```

NEVER EVER TOUCH THE STRUCTURE OF THE TABLE

INTERACT WITH THE DATABASE FOR

PROVIDING DATA

MODIFYING THE DATA

REMOVING THE DATA

Example

1)INSERT INTO Employee VALUES (111, ‘George’, ‘HR’,45000);

2)UPADTE Employee set salary=50000 where id=84;

3)DELETE from Employee Where id = 200;

==========

6)What Is DDL Explain In Details?

Data definition language (DDL) refers to the set of SQL commands that can create and manipulate the structures of a database. It Deals with schema objects such as databases, tables, and views.

• **DDL**: Data Definition Language
e.g. CREATE, ALTER, DROP, RENAME

1)CREATE :

The CREATE TABLE statement is used to create a new table in a database.

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

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

Example:

The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

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

Create Table Using Another Table

A copy of an existing table can also be created using CREATE TABLE.

The new table gets the same column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

Syntax

```
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

The following SQL creates a new table called "TestTables" (which is a copy of the "Customers" table):

Example

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

2)ALTER :

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 - ADD Column

1.To add a column in a table, use the following syntax:

ALTER TABLE *table_name*
ADD *column_name datatype*;

The following SQL adds an "Email" column to the "Customers" table:

Example:
ALTER TABLE Customers
ADD Email varchar(255);

ALTER TABLE - DROP COLUMN

2.To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE *table_name*
DROP COLUMN *column_name*;

The following SQL deletes the "Email" column from the "Customers" table:

Example

ALTER TABLE Customers
DROP COLUMN Email;

ALTER TABLE - ALTER/MODIFY COLUMN

3.To change the data type of a column in a table, use the following syntax:

My SQL

ALTER TABLE *table_name*
MODIFY COLUMN *column_name datatype*;

3)DROP :

The DROP TABLE statement is used to drop an existing table in a database.

Syntax:

DROP TABLE *table_name*;

4)RENAME :

used to rename database name and table name;

Example**:**

ALTER TABLE table_name RENAME TO new_table_name

==========

7)What Is DTL Explain in Details?

Transactional Control Commands

Transactional control commands are only used with the **DML Commands** such as - INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

TCL: Transaction Control Language
e.g. SAVEPOINT, COMMIT, ROLLBACK

The following commands are used to control transactions.

1)COMMIT − to save the changes.

USED TO PUSH THE DATA FROM TEMP MEORY TO THE PHYSICAL LOCATION OF THE DATABASE

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

Example 1)

Following is an example which would delete those records from the table which have age = 25 and then COMMIT the changes in the database.

```
SQL> DELETE FROM CUSTOMERS
  WHERE AGE = 25;
SQL> COMMIT;
```

After this command we get the updated record of customers table

2) ROLLBACK − to roll back the changes

USED TO ROLLBACK THE CHANGE NOT MOVED TO THE DATABASE DATA IS IN TEMP MEMORY CAN BE ROLLEDBACK

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

Following is an example, which would delete those records from the table which have the age = 25 and then ROLLBACK the changes in the database.

```
SQL> DELETE FROM CUSTOMERS
  WHERE AGE = 25;
SQL> ROLLBACK;
```

After this command we get the same earlier record of customers table

3)SAVEPOINT –

creates points within the groups of transactions in which to ROLLBACK.

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

==========

8)What Is DCL?

DCL is short name of Data Control Language which includes commands such as GRANT and mostly concerned with rights, permissions and other controls of the database system.

1)GRANT - allow users access privileges to the database(PROVIDE PERMISSION)

2)REVOKE - withdraw users access privileges given by using the GRANT command(REMOVE PERMISSION)

==========

9)Difference between drop truncate delete?

	Delete	Truncate	Drop
1.	Delete is a DML command	truncate is DDL command.	Drop is DDL command
2.	delete statement can be used for deleting the specific data	Truncate can be used to delete the entire data of the table keep the structure as it is.	DROP table query removes one or more table data as well as table structure.
3.	Syntax: Delete table tablename; Delete from tablename where ?	Syntax: Truncate table tablename;	Syntax: Drop table tablename;
4.	It can be rollback before commit.	Truncate can't rollback	Drop not rollback
5.	It is slower as compare to truncate	It is faster	It is faster
6.	Delete statement use where clause to delete particular records	Truncate can't use where clause to delete particular data.	Drop can't use where clause.
7.	Eg. Delete from Employee; Delete From Employee where employee_id =300;	Eg. Truncate table Employee;	Eg. Drop Table Employee; Drop database acts;

==========

10)What is a difference between Commit, Rollback and Savepoint ?

COMMIT: Ends the current transaction by making all pending data changes permanent.

ROLLBACK: Ends the current transaction by discarding all pending data changes.

SAVEPOINT: Divides a transaction into smaller parts. You can rollback the transaction till a particular named savepoint.

==========

11)SQL CONSTRAINTS

SQL constraints are used to specify rules for data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level.

Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

1.NOT NULL - Ensures that a column cannot have a NULL value

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

Example:

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

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

2.UNIQUE - Ensures that all values in a column are different

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

Example:

The following SQL creates a UNIQUE constraint 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,
    UNIQUE (ID)
);
```

3.PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

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).

Example -1)

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)
);
```

Example -2)

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

Note: In the example below there is only ONE PRIMARY KEY (PK_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

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

4.FOREIGN KEY - Prevents actions that would destroy links between tables

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.

Persons Table(Parent Table)

Having Columns PersonId, FirstName, LastName, age

Orders Table(Child table)

Having Columns OrderId,OrderNumber,PersonId

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

Example 2)

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

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

Example 2)

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

```
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)
);
```

5.CHECK - Ensures that the values in a column satisfies a specific condition

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

Example:

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

6.DEFAULT - Sets a default value for a column if no value is specified

The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

Example 1)

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

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

Example 2)

The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT GETDATE()
);
```

7.CREATE INDEX - Used to create and retrieve data from the database very quickly

The CREATE INDEX statement is used to create indexes in tables.

==========

12)PRIMARY KEY –

A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

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).

Example-1)

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)
);
```

Example-2)

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

Note: In the example below there is only ONE PRIMARY KEY (PK_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

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

real life example:-

It is a unique identifier such as,

1)such as a driver license number,

2)telephone number (including area code),

3)vehicle identification number (VIN).

==========

13)FOREIGN KEY –

Prevents actions that would destroy links between tables

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.

Persons Table(Parent Table)

Having Columns PersonId, FirstName, LastName, age

Orders Table(Child Table)

Having Columns OrderId,OrderNumber,PersonId

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

Example 1)

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

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

Example 2)

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

```
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)
)
```

==========

14)Difference between primary & foreign key.

Comparison Basis	Primary Key	Foreign Key
Basic	It is used to identify each record into the database table uniquely.	It is used to links two tables together. It means the foreign key in one table refers to the primary key of another table.
NULL	The primary key column value can never be NULL.	The foreign key column can accept a NULL value.
Count	A table can have only one primary key.	A table can have more than one foreign key.
Duplication	The primary key is a unique attribute; therefore, it cannot stores duplicate values in relation.	We can store duplicate values in the foreign key column.
Indexing	The primary key is a clustered index by default, which means it is indexed automatically.	A foreign key is not a clustered index by default. We can make clustered indexes manually.
Deletion	The primary key value can't be removed from the table. If you want to delete it, then make sure the referencing foreign key does not contain its value.	The foreign key value can be removed from the table without bothering that it refers to the primary key of another table.
Insertion	We can insert the values into the primary key column without any limitation, either it present in a foreign key or not.	The value that is not present in the column of a primary key cannot be inserted into the referencing foreign key.
Temporary table	The primary key constraint can be defined on the temporary tables.	A foreign key constraint cannot be defined on the temporary tables.
Relationship	It cannot create a parent-child relationship in a table.	It can make a parent-child relationship in a table.

==========

15)Difference between primary unique & foreign key.

	Primary key	Unique key	Foreign key
1.	Primary key is a column of table Which uniquely identifies the rows in that table.	Unique key constraints also uniquely identifies the rows in that table.	It refers to the field in a table which is the primary key of another table.
2.	Primary Key does not allow NULL Values.	Unique Key is like Primary Key, the only difference Unique allows single NULL value.	Foreign Key can have multiple NULL values
3.	Each Table can have only one Primary Key.	Table can have multiple Unique Keys	Table can have multiple Foreign Keys.
4.	Primary Key creates Clustered Index by default.	Unique Key creates Non-Clustered Index by default.	Foreign Key does not create any index by default
5.	Primary key can be created on temporary tables and table variables.	Unique key can be created on temporary tables and table variables.	Foreign key cannot be created on temporary tables and table variables.

==========

16)KEYS IN DATABASE:-

Types of Keys in DBMS

There are broadly seven types of keys in DBMS:

1)Primary Key

2)Candidate Key

3)Super Key

4)Foreign Key

5)Composite Key

6)Alternate Key

7)Unique Key

1. Primary Key

A primary key is a column of a table or a set of columns that helps to identify every record present in that table uniquely.There can be only one primary Key in a table. Also, the primary Key cannot have the same values repeating for any row.

Every value of the primary key has to be different with no repetitions.

The PRIMARY KEY (PK) constraint put on a column or set of columns will not allow

them to have any null values or any duplicates.

One table can have only one primary key constraint.

Any value in the primary key cannot be changed by any foreign keys (explained below) which refer to it.

2. Super Key

Super Key is the set of all the keys which help to identify rows in a table uniquely.

This means that all those columns of a table than capable of

identifying the other columns of that table uniquely will all be considered super keys.

Super Key is the superset of a candidate key (explained below).

The Primary Key of a table is picked from the super key set to be made the table’s identity attribute.

3. Candidate Key

Candidate keys are those attributes that uniquely identify rows of a table.

The Primary Key of a table is selected from one of the candidate keys.

So, candidate keys have the same properties as the primary keys explained above.

There can be more than one candidate keys in a table.

4. Alternate Key

As stated above, a table can have multiple choices for a primary key;

however, it can choose only one. So, all the keys

which did not become the primary Key are called alternate keys.

5. Foreign Key

Foreign Key is used to establish relationships between two tables.

A foreign key will require each value in a column or set of columns

to match the Primary Key of the referential table.

Foreign keys help to maintain data and referential integrity.

6. Composite Key

Composite Key is a set of two or more attributes that help identify each tuple

in a table uniquely. The attributes in the set may not be unique when considered separately.

However, when taken all together, they will ensure uniqueness.

7. Unique Key

Unique Key is a column or set of columns that uniquely identify each record in a table.

All values will have to be unique in this Key.

A unique Key differs from a primary key because it can have only one null value,

whereas a primary Key cannot have any null values.

==========

17)What are the Inbuild Functions In MySql?

1) NUMERIC FUNCTION

2) STRING FUNCTION

3) DATE FUNCTION

---

18)Explain Functions

1)Numeric Functions/Aggregate Functions

I] ROUND() :(NUMERIC FUNCTIONS)

WHICH RETURN THE ROUND OFF VALUE PROVIDED

SELECT ROUND(23.666); -> 24

SELECT ROUND(29.8999,2); -> 29.90

SELECT ROUND(333.88888,-2); ->300

SELECT ROUND(2987.56,-1); -> 2990

==========

II] CEIL() :(NUMERIC FUNCTIONS)

RETURNS THE UPPER INTEGER VALUES BASED ON THE DECIMAL PROVIDED

SELECT CEIL(23.77); ---→24

==========

III] FLOOR() :(NUMERIC FUNCTIONS)

RETURNS THE NEAREST LOWER VALUE

34.78 : 34

35.23 : 35

SELECT FLOOR(67.99); ---68

==========

IV) TRANCTAE() :(NUMERIC FUNCTIONS)

TRUNCATE FUNCTION HAS 2 PARAMETERS

BASED ON THE 2ND PARAMETER IT REMOVES THE VALUES AFTER THE NUMBER PROVIDED IN THE 2ND PARAMETER

mysql> SELECT TRUNCATE(345.567,1); ----345.5

mysql> ; SELECT TRUNCATE(345.567,2); ------345.56

mysql> SELECT TRUNCATE(345.567,0); ----345

mysql> SELECT TRUNCATE(345.567,-2); --- 300

==========

V] DIV() : NUMERIC FUNCTIONS

returrns value without decimals

SELECT 34 DIV 11; ----3

==========

VI] MOD() :NUMERIC FUNCTIONS

Returns Remainder

MOD : MODULUS

mysql> SELECT 34 MOD 5; ----4

mysql> SELECT MOD(23,2);---1

==========

VII] SQRT() :NUMERIC FUNCTIONS

VIII]SQRT :

SQUARE ROOT OF 4 : 2

SELECT SQRT(16); : returns 4

SELECT SQRT(9); : returns 3

==========

2)String Functions

1) LENGTH: RETUNS THE LENGTH

SELECT LENGTH('LOKESH JAWALE'); ---13

==========

2) UPPER :

STRINGS IN TO UPPER CASE

==========

3) LOWER :

STRING TO LOWER CASE

==========

4) CONCAT

COMBINE TWO STRING

MOST OF THE CASES REQUIRED FOR NAMES

PRINT ON MOBILES BILL OR ELECTRICITY BILLS

RAHUL BANSAL

RAHUL, BANSAL

BANSAL, RAHUL

CONCAT(ACCEPT STRINGS WHICH NEED TO CONCAT);

==========

5) SUBSTRING : RETURNS THE SUBSTRING FROM THE MAIN STRING

3 PARAMTERS

1ST PARAM : ORINAL STRING

2ND PARAM : START OF THE STRING

3RD PARAM : LENGHT AFTER 2ND PARAM

mysql> SELECT SUBSTR('LOKESH',3,2); ----KE

==========

6)TRIM :

HAVE SPACES STORED IN THE BASES

TRIM(STRINGS) : REMOVES THE SPACES FROM THE FRONT N BACK OF THE STRING

mysql> SELECT TRIM(' LOKESH JAWALE '); ---LOKESH JAWALE

==========

7) LPAD :

LEFT PADDING

TO MAKE THE COMPETE STRING OF THE SAME LENGTH

3 PARAM

1ST PARAM : VALUE 5

2ND PARAM : LENGTH REQUIRED FOR THE COMPLETE STRING 10

3RD PARAM : REST 5 POSITION REPLACED WITH THE 3RD PARAMETER STRING

SELECT LPAD('LOKESH',12,'&'); ----&&&&&&LOKESH

==========

8) RPAD

RIGHT PADDING

mysql> SELECT RPAD('JAWALE',9,'*');----- JAWALE***

==========

3)DATE FUNCTIONS

1) NOW()

RETURNS TODAYS DATETIME IN FORMAT YYYY-DD-MM HH:MM:SS

SELECT NOW();

==========

2) CURDATE()

RETUNRS DATE WITHOUT TIME

==========

3) CURTIME()

RETURNS TIME WITHOUT DATE

==========

4) DATE_FORMAT

DATE INTO SPECIFIC FORMT BASED ON THE REQUIREMENTS

==========

5) DAY_ADD()

==========

6) DAY_SUB()

==========

7) YEAR, MONTH, DAY

EXTRACT YEAR , MONTH AND DAY FROM THE DATE

EX: SELECT EMPLOYEE_ID,YEAR(HIRE_DATE) AS JOINING YEAR FROM EMP LIMIT 10;

==========

8) STRING TO DATE

STR_TO_DATE(STRING,PATTERNS);

Select str_to_date('30-01-1987','%d-%m-%Y')

1987-01-30

|

==========

9)DATEDIFF()

TWO DATES

DATEDIFF(NOW(),HIRE_DATE)

EXMPLE:

mysql> SELECT TRUNCATE(DATEDIFF(NOW(),HIRE_DATE)/365,0) "EXPERIENCE" FROM EMPLOYEES LIMIT

---

4) SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

AVG() - Returns the average value

COUNT() - Returns the number of rows

FIRST() - Returns the first value

LAST() - Returns the last value

MAX() - Returns the largest value

MIN() - Returns the smallest value

SUM() - Returns the sum

==========

19)Explain MYSQL VS NO SQL

Key Areas	SQL	NoSQL
Type of database	Relational Database	Non-relational Database
Schema	Pre-defined Schema	Dynamic Schema
Database Categories	Table based Databases	Document-based databases, Key-value stores, graph stores, wide column stores
Complex Queries	Good for complex queries	Not a good fit for complex queries
Hierarchical Data Storage	Not the best fit	Fits better when compared to SQL
Scalability	Vertically Scalable	Horizontally Scalable
Language	Structured Query language	Unstructured Query language
Online Processing	Used for OLTP	Used for OLAP
Base Properties	Based on ACID Properties	Based on CAP Theorem
External Support	Excellent support is provided by all SQL vendors	Rely on community support.

==========

20)what if we mention null=null condition on joins

it's not possible to join on NULL values in SQL Server like you might expect,

we need to be creative to achieve the results we want.

One option is to make our AccountType column NOT NULL and set some other default value

==========

21)What are the different types of SQL operators?

Operators are the special keywords or special characters reserved for performing particular operations. They are also used in SQL queries. We can primarily use these operators within the WHERE clause of SQL commands. It's a part of the command to filters data based on the specified condition. The SQL operators can be categorized into the following types:

Arithmetic operators:

These operators are used to perform mathematical operations on numerical data. The categories of this operators are

addition (+), subtraction (-), multiplication (*), division (/), remainder/modulus (%), etc.

Logical operators:

These operators evaluate the expressions and return their results in True or False. This operator includes

ALL, AND, ANY, ISNULL, EXISTS, BETWEEN, IN, LIKE, NOT, OR, UNIQUE.

Comparison operators:

These operators are used to perform comparisons of two values and check whether they are the same or not. It includes

equal to (=),

not equal to (!= or <>),

less than (<), less than or equal to (<=),

greater than(>)

greater than or equal to (>=),

not less than (!), etc.

Bitwise operators:

It is used to do bit manipulations between two expressions of integer type. It first performs conversion of integers into binary bits and then applied operators such as

AND (& symbol),

OR (|, ^),

NOT (~), etc.

Compound operators:

These operators perform operations on a variable before setting the variable's result to the operation's result. It includes

Add equals (+=),

subtract equals (-=),

multiply equals (*=),

divide equals (/=),

modulos equals (%=), etc.

String operators:

These operators are primarily used to perform concatenation and pattern matching of strings. It includes

+ (String concatenation),

+= (String concatenation assignment),

% (Wildcard),

[] (Character(s) matches),

[^] (Character(s) not to match),

_ (Wildcard match one character), etc.

==========

22)WHAT IS PATTERN MATCHING

Pattern matching is the process of checking

whether a specific sequence of characters/tokens/data exists among the given data.

Which SQL operator performs pattern matching?

LIKE operator(Logical Operator)

==========

23)WHAT IS IN KEYWORD/OPERATOR

The SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

EXAMPLES:

The following SQL statement selects all customers that are located in "Germany", "France" or "UK":

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

==========

24)Data Types In DBMS

1)String Data Types

CHAR(size)

A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1

VARCHAR(size)

A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters - can be from 0 to 65535

BINARY(size)

Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1

VARBINARY(size)

Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes.

TINYBLOB For BLOBs (Binary Large Objects).

Max length: 255 bytes

TINYTEXT

Holds a string with a maximum length of 255 characters

TEXT(size)

Holds a string with a maximum length of 65,535 bytes

BLOB(size)

For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data

MEDIUMTEXT

Holds a string with a maximum length of 16,777,215 characters

LONGTEXT

Holds a string with a maximum length of 4,294,967,295 characters

LONGBLOB

For BLOBs (Binary Large Objects).

Holds up to 4,294,967,295 bytes of data

ENUM(val1, val2, val3, ...)

A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them

SET(val1, val2, val3, ...)

A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET .

==========

2)NUMERIC DATA TYPES

BIT(size)

A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1.

TINYINT(size)

A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255)

BOOL

Zero is considered as false, nonzero values are considered as true.

BOOLEAN Equal to BOOL

SMALLINT(size) A small integer.

Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255)

MEDIUMINT(size)

A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255)

INT(size)

A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)

INTEGER(size)

Equal to INT(size)

BIGINT(size)

A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255)

FLOAT(size, d)

A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions

FLOAT(p)

A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE()

DOUBLE(size, d)

A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter

3)Date and Time Data Types

DATE

A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'

DATETIME(fsp)

A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time

TIMESTAMP(fsp)

A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition

TIME(fsp)

A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'

YEAR

A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000.

==========

25)DIFFERENCE BETWEEN HAVING AND WHERE CLAUSE?

Comparison Basis	WHERE Clause	HAVING Clause
Definition	It is used to perform filtration on individual rows.	It is used to perform filtration on groups.
Basic	It is implemented in row operations.	It is implemented in column operations.
Data fetching	The WHERE clause fetches the specific data from particular rows based on the specified condition	The HAVING clause first fetches the complete data. It then separates them according to the given condition.
Aggregate Functions	The WHERE clause does not allow to work with aggregate functions.	The HAVING clause can work with aggregate functions.
Act as	The WHERE clause acts as a pre-filter.	The HAVING clause acts as a post-filter.
Used with	We can use the WHERE clause with the SELECT, UPDATE, and DELETE statements.	The HAVING clause can only use with the SELECT statement.
GROUP BY	The GROUP BY clause comes after the WHERE clause.	The GROUP BY clause comes before the HAVING clause.

==========

26)ACID PROPERTIES IN DBMS

DBMS is the management of data that should remain integrated when any changes are done in it. It is because if the integrity of the data is affected, whole data will get disturbed and corrupted. Therefore, to maintain the integrity of the data, there are four properties described in the database management system, which are known as the ACID properties.

The ACID properties are meant for the transaction that goes through a different group of tasks, and there we come to see the role of the ACID properties.which are known as the ACID properties.

1)Atomicity

i) By this, we mean that either the entire transaction takes place at once or doesn't happen at all.

ii)There is no midway i.e. transactions do not occur partially.

iii)Each transaction is considered as one unit and either runs to completion or is not executed at all.

iv)It involves the following two operations.

—Abort: If a transaction aborts, changes made to database are not visible.

—Commit: If a transaction commits, changes made are visible.

Atomicity is also known as the 'All or nothing rule'.

2)Consistency

i) This means that integrity constraints must be maintained so that the database is consistent before and after the transaction.

ii) It refers to the correctness of a database.

3)Isolation

i) This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state.

ii) Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transactions written to memory or has been committed.

iii)This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

4)Durability:

i) This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs.

ii)These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

iii)The ACID properties, in totality, provide a mechanism to ensure correctness and consistency of a database in a way such that each

transaction is a group of operations that acts a single unit, produces consistent results, acts in isolation from other operations and updates that it makes are durably stored.

==========

27)What is normalization?

Normalization is a way of organizing the data in the database.

Normalization is used to remove redundant data from the database and to store non-redundant data into it.

Refer the link for better understanding.
https://beginnersbook.com/2015/05/normalization-in-dbms/

Normalization rules divides larger tables into smaller tables and links them using relationships.

Without Normalization in SQL, we may face many issues such as

1. ***Insertion anomaly***: It occurs when we cannot insert data to the table without the presence of another attribute
2. ***Update anomaly***: It is a data inconsistency that results from data redundancy and a partial update of data.
3. ***Deletion Anomaly***: It occurs when certain attributes are lost because of the deletion of other attributes.

Database normal forms:

1NF :

- Each table cell should contain a single value.
- Each record needs to be unique.

First normal form (1NF)

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212 9900012222
103	Ron	Chennai	7778881212

104	Lester	Bangalore	9990000123 8123450987

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is **not in 1NF** as the rule says “each attribute of a table must have atomic (single) values”, the emp_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123

104	Lester	Bangalore	8123450987

2NF :

- Be in 1NF
- Single Column Primary Key

Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38

333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}
Non prime attribute: teacher_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "**no** non-prime attribute is dependent on the proper subset of any candidate key of the table".

To make the table complies with 2NF we can break it in two tables like this:
teacher_details table:

teacher_id	teacher_age
111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables comply with Second normal form (2NF).

3NF :

- Be in 2NF
- Has no transitive functional dependencies

Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

An attribute that is not part of any candidate key is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on

Candidate Keys: {emp_id}
Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

BCNF:

Even when a database is in 3rd Normal Form, still there would be anomalies resulted if it has more than one **Candidate** Key.
Sometimes is BCNF is also referred as **3.5 Normal Form.**

4NF
5NF
6NF

==========

28)What is Denormalization?

Denormalization is a database optimization technique in which we add redundant data to one or more tables. This can help us avoid costly joins in a relational database. Note that denormalization does not mean not doing normalization. It is an optimization technique that is applied after doing normalization.

Denormalization in Databases When we normalize tables, we break them into multiple smaller tables. So when we want to retrieve data from multiple tables, we need to perform some kind of join operation on them. In that case, we use the denormalization technique that eliminates the drawback of normalization.

Denormalization is a technique used by database administrators to optimize the efficiency of their database infrastructure. This method allows us to add redundant data into a normalized database

QUE)

Pros of Denormalization

The following are the advantages of denormalization:

1. Enhance Query Performance

Fetching queries in a normalized database generally requires joining a large number of tables, but we already know that the more joins, the slower the query. To overcome this, we can add redundancy to a database by copying values between parent and child tables,

minimizing the number of joins needed for a query.

2. Make database more convenient to manage

A normalized database is not required calculated values for applications. Calculating these values on-the-fly will take a longer time, slowing down the execution of the query. Thus, in

denormalization, fetching queries can be simpler because we need to look at fewer tables.

3. Facilitate and accelerate reporting

Suppose you need certain statistics very frequently. It requires a long time to create them from live data and slows down the entire system. Suppose you want to monitor client revenues over a certain year for any or all clients. Generating such reports from live data will require "searching" throughout the entire database, significantly slowing it down.

QUE)

Cons of Denormalization

The following are the disadvantages of denormalization:

i)It takes large storage due to data redundancy.

ii)It makes it expensive to updates and inserts data in a table.

iii)It makes update and inserts code harder to write.

iv)Since data can be modified in several ways, it makes data inconsistent.

Hence, we'll need to update every piece of duplicate data.

v)It's also used to measure values and produce reports.

vi) We can do this by using triggers, transactions, and/or procedures for all operations that must be performed together.

.

-==========

29)NORMALIZATION VS DENORMALIZATION

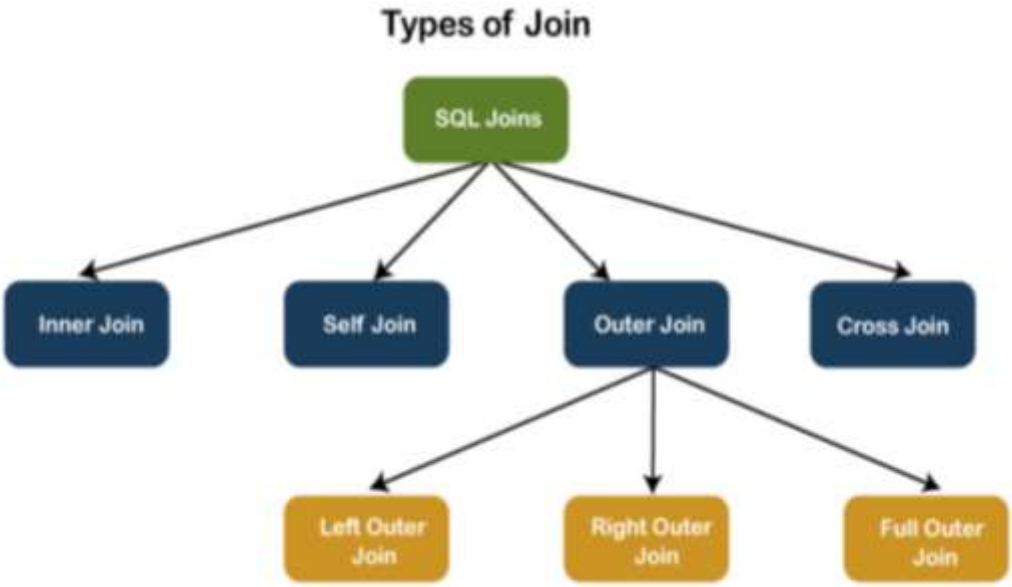
Sr. No.	Key	Normalization	Denormalization
1	Implementation	Normalization is used to remove redundant data from the database and to store non-redundant and consistent data into it.	Denormalization is used to combine multiple table data into one so that it can be queried quickly.
2	Focus	Normalization mainly focuses on clearing the database from unused data and to reduce the data redundancy and inconsistency.	Denormalization on the other hand focus on to achieve the faster execution of the queries through introducing redundancy.
3	Number of Tables	During Normalization as data is reduced so a number of tables are deleted from the database hence tables are lesser in number.	On another hand during Denormalization data is integrated into the same database and hence a number of tables to store that data increases in number.
4	Memory consumption	Normalization uses optimized memory and hence faster in performance.	On the other hand, Denormalization introduces some sort of wastage of memory.
5	Data integrity	Normalization maintains data integrity i.e. any addition or deletion of data from the table will not create any mismatch in the relationship of the tables.	Denormalization does not maintain any data integrity.
6	Where to use	Normalization is generally used where number of insert/update/delete operations are performed and joins of those tables are not expensive.	On the other hand Denormalization is used where joins are expensive and frequent query is executed on the tables.

-==========

30) What are the different types of joins in SQL?

Joins are used to merge two tables or retrieve data from tables. It depends on the relationship between tables.

the following are the different types of joins used in SQL:



1)Inner Join:

Returns records that have matching values in both tables

2)Self-join:

Returns records that have matching values in same (self) table.

3)Outer Join

1. **LEFT OUTER JOIN**: Return all records from the left table, and the matched records from the right table
2. **RIGHT OUTER JOIN**: Return all records from the right table, and the matched records from the left table
3. **FULL OUTER JOIN**: Return all records when there is a match in either left or right table.

4)Cross join / Cartesian Join –

The absence of join condition it takes **Cartesian** product of table. That is no of rows in table A & no of rows in table B

5)Natural join:

Returns records that have matching values in both tables.

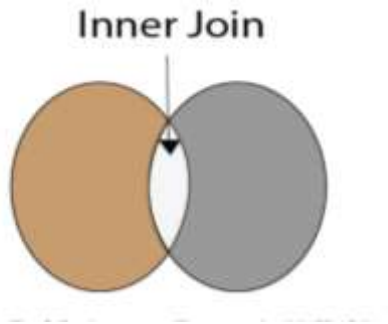
6)Non-Equi join:
Returns records base on non-matching values in both tables.

7)Self-join:
Returns records that have matching values in same (self) table.
-==========

31) What is INNER JOIN in SQL?

Inner join returns only those records from the tables that match the specified condition and hides other rows and columns. In simple words, it fetches rows when there is at least one match of rows between the tables is found. INNER JOIN keyword joins the matching records from two tables. It is assumed as a default join, so it is optional to use the INNER keyword with the query.

The below visual representation explain this join more clearly



The following syntax illustrates the INNER JOIN:

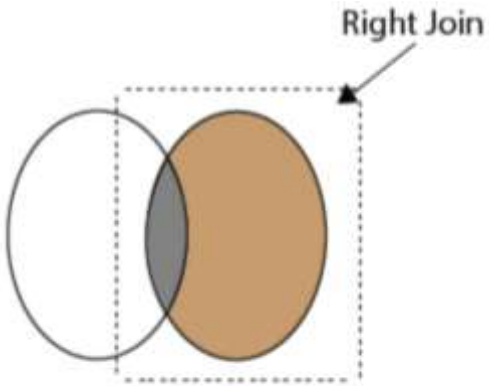
```
SELECT column_lists
 FROM table1
 INNER JOIN table2 ON join_condition1
INNER JOIN table3 ON join_condition2
```

-==========

32) What is the Right JOIN in SQL?

The Right join is used to retrieve all rows from the right-hand table and only those rows from the other table that fulfilled the join condition. It returns all the rows from the right-hand side table even though there are no matches in the left-hand side table. If it finds unmatched records from the left side table, it returns a Null value. This join is also known as Right Outer Join.

The below visual representation explain this join more clearly:



The following syntax illustrates the RIGHT JOIN:

SELECT colum_lists

FROM table1

RIGHT JOIN table2

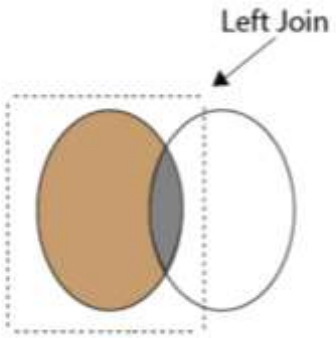
ON join_condition;

-==========

33) What is Left Join in SQL?

The Left Join is used to fetch all rows from the left-hand table and common records between the specified tables. It returns all the rows from the left-hand side table even though there are no matches on the right-hand side table. If it will not find any matching record from the right side table, then it returns null. This join can also be called a Left Outer Join.

The following visual representation explains it more clearly:



The following syntax illustrates the RIGHT JOIN:

SELECT colum_lists

FROM table1

LEFT JOIN table2

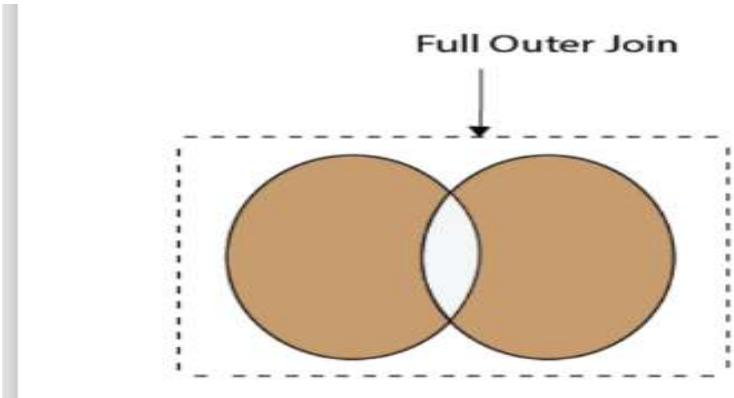
ON join_condition

-==========

34) What is Full Join in SQL?

The Full Join results from a combination of both left and right join that contains all the records from both tables. It fetches rows when there are matching rows in any one of the tables. This means it returns all the rows from the left-hand side table and all the rows from the right-hand side tables. If a match is not found, it puts NULL value. It is also known as FULL OUTER JOIN.

The following visual representation explains it more clearly



The following syntax illustrates the FULL JOIN:

```
SELECT * FROM table1
 FULL OUTER JOIN table2
ON join_condition;
```

-==========

35) What is self-join and what is the requirement of self-join?

A SELF JOIN is used to join a table with itself. This join can be performed using table aliases, which allow us to avoid repeating the same table name in a single sentence. It will throw an error if we use the same table name more than once in a single query without using table aliases. A SELF JOIN is required when we want to combine data with other data in the same table itself. It is often very useful to convert a hierarchical structure to a flat structure.

The following syntax illustrates the SELF JOIN:

```
SELECT column_lists
FROM table1 AS T1, table1 AS T2
 WHERE join_conditions;
```

Example:

If we want to get retrieve the student_id and name from the table where student_id is equal, and course_id is not equal, it can be done by using the self-join:

```
SELECT  s1.student_id, s1.name
FROM student AS s1, student s2
 WHERE s1.student_id=s2.student_id
 AND s1.course_id<>s2.course_id;
```

-==========

36) EXPLAIN PROCEDURE

i) A stored procedure is a sequence of statement or a named PL/SQL block which performs one or more specific functions. It is similar to a procedure in other programming languages.

ii)It is stored in the database and can be repeatedly executed.

iii)It is stored as schema object. It can be nested, invoked and parameterized.

Syntax:

```
DELIMITER &&
CREATE PROCEDURE procedure_name [[IN | OUT | INOUT] parameter_name d
atatype [, parameter datatype]) ]
BEGIN
   Declaration_section
   Executable_section
END &&
DELIMITER ;
```

que-1)

How to call a stored procedure?

CALL procedure_name (parameter(s))

que-2)

What are the advantages of stored procedure?

Modularity, extensibility, reusability, Maintainability and one time compilation.

-==========

37) FUNCTION (STORED FUNCTION)

A stored function in MySQL is a set of SQL statements that perform some task/operation and return a single value. It is one of the types of stored programs in MySQL.

The stored function is almost similar to the procedure in MySQL, but it has some differences that are as follows:

i)The function parameter may contain only the IN parameter

ii)but can't allow specifying this parameter, while the procedure can allow IN, OUT, INOUT parameters.

iii)The stored function can return only a single value defined in the function header.

iv)The stored function may also be called within SQL statements.

v)It may not produce a result set.

The syntax of creating a stored function in MySQL is as follows:

```
DELIMITER $$
CREATE FUNCTION fun_name(fun_parameter(s))
RETURNS datatype
[NOT] {Characteristics}
fun_body;
```

-==========

38). Which are Different Parameters in Procedure and Functions?

1) IN type parameter: These types of parameters are used to send values to stored procedures.

2) OUT type parameter: These types of parameters are used to get values from stored
procedures. This is similar to a return type in functions.

3) INOUT parameter: These types of parameters are used to send values and get values from stored procedures.

-==========

38. Stored procedure vs stored function

	Functions	Procedures
1.	A function has a return type and returns a value.	A procedure does not have a return type.
2.	You cannot use a function with Data Manipulation queries. Only Select queries are allowed in functions.	You can use DML queries such as insert, update, select etc… with procedures.
3.	You cannot call stored procedures from a function	You can call a function from a stored procedure.
4.	A function does not allow output parameters	A procedure allows both input and output parameters.
5.	You cannot manage transactions inside a function.	You can manage transactions inside a function.

==========

39. DBMS VS RDBMS

DBMS	RDBMS
DBMS applications store **data as file**.	RDBMS applications store **data in a tabular form**.
Normalization is not present in DBMS	**Normalization is** present in RDBMS.
DBMS uses file system to store data, so there will be **no relation between the tables**.	in RDBMS, data values are stored in the form of tables, so a **relationship** between these data values will be stored in the form of a table as well.
DBMS **does not support distributed database**.	RDBMS **supports distributed database**.
DBMS is meant to be for small organization and **deal with small data**. it supports **single user**.	RDBMS is designed to **handle large amount of data**. It supports **multiple users**.
Examples of DBMS are file systems, **xml** etc.	Example of RDBMS are **mysql**, **postgre**, **sql server**, **oracle** etc

==========

40) What is the difference between SQL and PL/SQL

SQL	PL/SQL
• It is a structured Query Language.	• It is a Procedural Structured Query Language.
• It does not support the use of Control Structures.	• Various Control Structures can be used like while loop and for loop.
• Data variables are absent.	• Data variables are present.
• It executes a single query at a time.	• It executes a block of statements at a time.
• It is a declarative language.	• It is a procedural language.
• It can be embedded in PL/SQL.	• It can not be embedded in SQL.
• It is used to write DML, DDL statements, and queries.	• It is accustomed to writing functions, program blocks, packages, and procedures triggers.
• It is a Data-oriented language.	• It is an application-oriented language.

==========

41) Explain 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.

Example:

1)The following SQL creates a view that shows all customers from Brazil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

2)The following SQL creates a view that selects every product in the "Products" table with a price higher than the average price:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

==========

42) Explain Triggers?

A trigger is a stored procedure in 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.

A trigger uses the special table to keep a copy of the row which we have just inserted, deleted or modified.

QUE-1)

What is a trigger in PL/SQL?

A trigger is a PL/SQL program which is stored in the database.

It is executed immediately before or after the execution of INSERT, UPDATE, and DELETE commands.

Types Of Triggers:

We can define the maximum six types of actions or events in the form of triggers:

1. **Before Insert:** It is activated before the insertion of data into the table.
2. **After Insert:** It is activated after the insertion of data into the table.
3. **Before Update:** It is activated before the update of data in the table.
4. **After Update:** It is activated after the update of the data in the table.
5. **Before Delete:** It is activated before the data is removed from the table.
6. **After Delete:** It is activated after the deletion of data from the table.

When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked.

Benefits of Triggers

i) Triggers can be written for the following purposes −

ii) Generating some derived column values automatically

iii) Enforcing referential integrity

iv) Event logging and storing information on table access

v)Auditing

vi) Synchronous replication of tables

vii)Imposing security authorizations

viii) Preventing invalid transactions

QUE-2)

What is the maximum number of triggers, you can apply on a single table?

12 triggers.

QUE-3)

Which command is used to delete a trigger?

DROP TRIGGER command.

==========

43). What is Cursor?

Cursor is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML operations on Table by User.

Cursors are used to store Database Tables.

There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors.

1. **Implicit Cursors:**

Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.

2. **Explicit Cursors :**

Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

==========

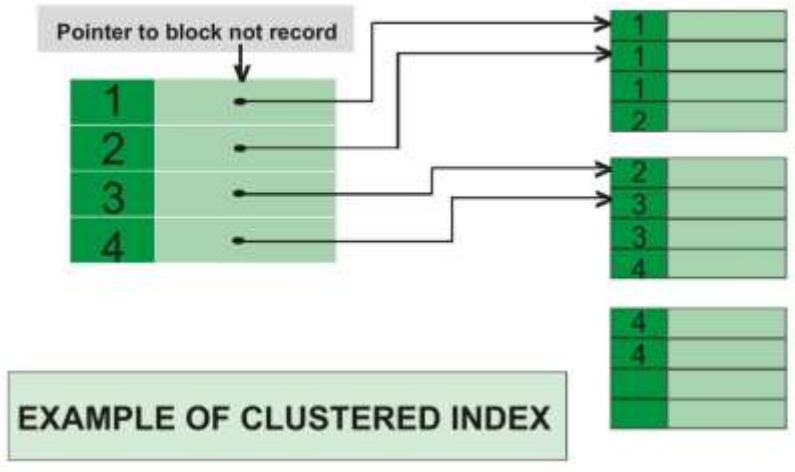
44) What is indexing?

Indexing is a data structure technique which allows you to quickly retrieve records from a database file. An Index is a small table having only two columns. **The first column comprises** a copy of the primary or candidate key of a table. **Its second column contains** a set of pointers for holding the address of the disk block where that specific key value stored.

Advantage: reduced i/o cost.

There are two types of indexing:

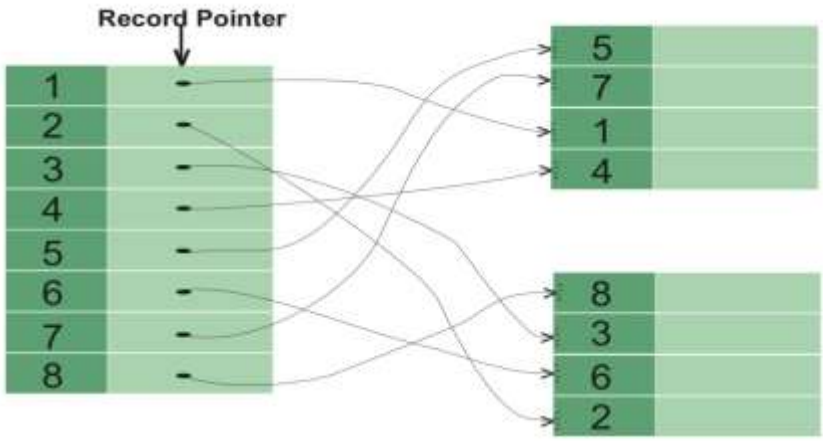
1. **Clustered Index:** A clustered index determines the physical order of data in a table. For this reason a table can have only one clustered index.
   Primary key constraint create clustered indexes automatically if no index already exists on the table.
   Eg. Clustered index is as same as dictionary where the data is arranged by alphabetical order.
   In clustered index, index contains pointer to block but not direct data.



2. **Non- Clustered Index:** Non-Clustered Index is similar to the index of a book. The index of a book consists of a chapter name and page number, if you want to read any topic or chapter then you can directly go to that page by using index of that book. No need to go through each and every page of a book.

   The data is stored in one place, and index is stored in another place. Since, the data and non-clustered index is stored separately, then you can have multiple non-clustered index in a table.

   In non-clustered index, index contains the pointer to data.



EXAMPLE OF NON-CLUSTERED INDEX

==========

45)Types Of Indexs

Some examples of the purposes and benefits are:

1)Unique indexes

enforce the constraint of uniqueness in your index keys.

2)Bidirectional indexes

allow for scans in both the forward and reverse directions.

3)Clustered indexes

can help improve the performance of queries that traverse the table in key order.

4)Expression-

based indexes efficiently evaluate queries with the indexed expression.

5)Unique and non-unique indexes

Unique indexes are indexes that help maintain data integrity

by ensuring that no two rows of data in a table have identical key values.

6)Clustered and non-clustered indexes

7)Partitioned and nonpartitioned indexes

Partitioned data can have indexes that are partitioned and nonpartitioned.

==========

Suppose u have 1 millions records in our db then how to fetch all records

It's depends on your requirement. You have to write an optimized query. In the table you can do indexing for better performance. Or you can divide the single table into multiple tables on the basis of different criteria like date created, alphabetical, etc.

Queries:

EMPNO	SAL

1. **Find the second highest salary of employee**

   Select max(sal) from emp
   where sal not in (select max(sal) from emp);

   or……..

   Select max(sal) from emp
   where sal <(select max(sal) from emp);

2. **Display the highest payed employee in each department**

   Select max(sal), deptno
   from emp
   group by deptno;

3. **Find number of employee in each department**

   Select count(*) ,deptno
   from emp
   group by deptno;

4. **Display alternate records in sql**

   //odd no records

```
select * from
(select empno, ename, sal,  rownum rn
 from emp
order by rn)
where mod (rn, 2) != 0;

//even no records

select * from
(select empno, ename, sal,  rownum rn
 from emp
order by rn)
where mod (rn, 2) = 0;
```

5. Find duplicate values and its frequency of a column

```
select ename , count(*)
from emp
group by ename
having count(*)>1;
```

6. Display employee whose name starts with m

```
select ename from emp
where ename like ‘m%’;
```

7. Name ends with n

```
Select ename from emp
Where ename like ‘%n’;
```

8. Having name in anyposition in name

```
Select ename from emp
Where ename like ‘%m%’;
```

9. Having name does not contain m anywhere

```
Select ename from emp
Where ename NOT like '%m%';
```

10. Display names of all emplyoees whose name contains exactly 4 letters

```
Select ename from emp
Where ename like '____';
```

11. Display names of all emplyoees whose name contains the second letter as 'L' & forth letter as ' M'

```
Select ename from emp
Where ename like '_L%';

Select ename from emp
Where ename like '___M%';
```

12. Display the employee names and hire dates for the employees joined at the month of December

```
Select hiredate, ename from emp
Where hiredate like '%DEC%';
```

13. Display names of all emplyoees whose name contains exactly 2 ' L's

```
Select ename from emp
Where ename like '%LL%';
```

14. Display names of employee whose name starts with 'J' and ends with 'S'

```
Select ename from emp
Where ename like 'J%S';
```

15. Display n^{th} row in sql

rownum cannot deal with greater than or equal to operater
……
……

//display 4^{th} row

```
Select * from emp
Where rownum<=4
Minus
Select * from emp
Where rownum<=3
```

Or…….

```
select * from
(select rownum r, ename, sal from emp)
where r=4;
```
…….

To print all columns of 4^{th} row

```
Select * from(select rownum r, emp.* from emp)
Where r=4;
```
……
……

To print 2^{nd} , 3^{rd} , 7^{th} record

```
Select * from (select rownum r, emo.* from emp)
Where r in(2,3,7);
```

16. Display employees who are working in location Chicago from emp and dept table

```
Select ename, sal, d.deptno,dname,loc
```

```
From emp e, dept d
Where e.deptno=d.deptno and LOC='CHICAGO';
```

17. Display the department name and total salaries from each department

```
Select dname, sum(sal)
from emp e, dept d
where e.deptno=d.deptno
group by deptno;
```

18. First 1st & last nth rows

```
Select * from(select rownum r, ename, sal from emp)
Where r=1 or r = (select count(*) from emp);
```

19. Display last two rows of the table

```
Select * from emp
Minus
Select  * from emp
Where rownum <=(select count(*)-2 from emp;
```

Or……
……

```
Select * from (select rownum r , ename, sal from emp)
Where r>(select count(*)-2 from emp);
```

For 1st two numbers

```
Select * from (select rownum r , ename, sal from emp)
Where r>(select count(*)-2 from emp)  or r in(1,2);
```

20. Nth highest salary

1st 3 highest salary

Select * from(select distinct sal from emp order by sal desc)
Where rownum<=3;

==========