FAQ’s: with explanation:

1. Types of commands and their examples.

Mainly 4 types

DDL – Data Definition Language – create, alter (add/modify/drop), drop

DML – Data Manipulation Language – insert, update, delete

DCL – Data Control Language – grant, revoke

TCL – Transaction Control Language – commit, rollback

1. Data Definition Language (DDL): Used for defining and managing database objects.

CREATE: Creates new database objects (e.g., tables, indexes).

ALTER: Modifies the structure of existing database objects.

DROP: Deletes database objects.

Syntax:

Create table tablename (columnname datatype);

Alter table tablename add columnname datatype;

Drop table tablename;

2. Data Manipulation Language (DML): Used for manipulating data in the database.

INSERT: Adds new records into a table.

UPDATE: Modifies existing records in a table.

DELETE: Removes records from a table.

Syntax:

Insert into tablename (columnnames) values (values for all columns);

Update tablename set columnname = value where condition;

Delete from tablename where condition;

3. Data Control Language (DCL): Used for controlling access to data within the database.

GRANT: Gives specific privileges to users or roles.

REVOKE: Removes specific privileges from users or roles.

Syntax:

Grant privilege\_name on tablename to username;

Revoke privilege\_name on tablename to username;

4. Transaction Control Commands: Used for managing transactions in the database.

COMMIT: Saves all changes made during the current transaction.

ROLLBACK: Reverts all changes made during the current transaction.

Syntax:

Commit;

Rollback;

2.What is Normalization and denormalization?

Normalization is the process of organizing a database to reduce redundancy and dependency by dividing large tables into smaller, related tables and defining relationships between them. The main objectives of normalization are to minimize redundancy, prevent update anomalies, and ensure data integrity. Normalization typically involves applying a series of normalization forms, such as First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), and so on, each with specific rules to eliminate different types of anomalies.

Denormalization, on the other hand, is the process of deliberately adding redundancy to a database schema to improve read performance or simplify queries. Denormalization involves reintroducing redundant data into one or more tables, which can lead to faster query execution times by reducing the need for joins and simplifying complex queries. However, denormalization can also increase the risk of data inconsistency and update anomalies, so it should be carefully considered and balanced against the performance benefits.

3. Explain 1NF, 2NF, 3NF.

First Normal Form (1NF):

In 1NF, data is organized into tables where each column contains atomic (indivisible) values, and each row represents a unique record. There should be no repeating groups or arrays within a table. All entries in each column must be of the same data type.

Example: Splitting a table containing multiple phone numbers for a person into separate rows, each with a single phone number, achieves 1NF.

Second Normal Form (2NF):

In 2NF, a table is in 1NF, and all attributes (non-key columns) are fully functionally dependent on the primary key. If a table has a composite primary key, each non-key attribute must be functionally dependent on the entire composite key, not just part of it.

Example: A table with order details where the order number and product code together form the primary key. The quantity and price fields are dependent on both the order number and product code.

Third Normal Form (3NF):

In 3NF, a table is in 2NF, and all non-key attributes are dependent only on the primary key, and not on other non-key attributes. There should be no transitive dependencies, meaning no non-key attribute should depend on another non-key attribute.

Example: A table with employee details where the employee's department is dependent on the employee's ID, not on the employee's name.

4. Share use case where you had to do denormalization in database.

Consider an e-commerce website with a large catalog of products. The database schema might include separate tables for products, categories, and orders. In a normalized database design, product information such as name, description, price, etc., would be stored in the "Products" table, while category information would be stored in a separate "Categories" table. When a user visits the website to browse products within a category, the system would typically need to perform joins between the "Products" and "Categories" tables to retrieve relevant product information.

However, in scenarios where the website experiences heavy traffic and frequent product browsing, performing joins for every user request can result in slower response times and increased server load. In such cases, denormalization can be used to improve query performance.

5.What is primary key and foreign key?

Primary Key:

A primary key is a column or a set of columns in a table that uniquely identifies each row in that table. It must contain unique values for each row, and it cannot contain NULL values. There can be only one primary key in a table. Primary keys are typically used to enforce entity integrity, ensuring that each row in a table is uniquely identifiable.

Foreign Key:

A foreign key is a column or a set of columns in a table that establishes a link or relationship between data in two tables. It refers to the primary key of another table, thus creating a parent-child relationship between the two tables. A foreign key constraint ensures referential integrity, meaning that values in the foreign key column must match values in the primary key column of the referenced table or be NULL. A table can have multiple foreign keys, each referring to a different table. Foreign keys are often used to enforce relational integrity and maintain consistency between related tables in a database.

6. What is alternate and candidate key?

Candidate Key:

A candidate key is a set of one or more columns in a table that uniquely identifies each row within that table. Like a primary key, a candidate key must satisfy two properties: uniqueness and irreducibility.

Uniqueness: Each combination of values in the candidate key must uniquely identify a row in the table.

Irreducibility: No subset of the candidate key should have the uniqueness property. In other words, removing any column from the candidate key would result in the loss of uniqueness.

In a table, there can be multiple candidate keys.

Alternate Key:

An alternate key is a candidate key that is not chosen as the primary key for the table. When one candidate key is selected as the primary key, the remaining candidate keys become alternate keys. Although they are not designated as the primary key, alternate keys still have the property of uniquely identifying rows within the table.

7.What are window functions?

Window functions, also known as analytic functions, are a powerful feature in SQL that allow you to perform calculations across a set of rows related to the current row within a query result set. These functions operate on a "window" of rows defined by a set of rows related to the current row.

ROW\_NUMBER(): Assigns a unique sequential integer to each row within a partition.

RANK(), DENSE\_RANK(): Computes the rank of each row within a partition, optionally handling ties differently.

LEAD() and LAG(): Access data from subsequent or preceding rows within the partition.

SUM(), AVG(), MIN(), MAX(): Calculate aggregate values over the window.

NTILE(): Distributes rows into a specified number of groups or "tiles".

8.Explain Ranking Functions? GIven a small table , write the output.

Ranking functions in SQL are used to assign a rank or a sequential number to each row within a partition of a result set. These functions help in analyzing data by providing insights into the relative positions of rows based on certain criteria.

ROW\_NUMBER(): Assigns a unique sequential integer to each row within a partition.

RANK(): Assigns a unique rank to each distinct row, with equal values receiving the same rank. Gaps can occur if there are ties.

DENSE\_RANK(): Similar to RANK(), but without gaps. Rows with equal values receive the same rank, and the next rank in sequence is incremented by one, irrespective of the number of tied rows.

SELECT

customer\_id,

order\_date,

product\_id,

ROW\_NUMBER() OVER (PARTITION BY customer\_id ORDER BY order\_date) AS row\_number,

RANK() OVER (PARTITION BY customer\_id ORDER BY order\_date) AS rank,

DENSE\_RANK() OVER (PARTITION BY customer\_id ORDER BY order\_date) AS dense\_rank

FROM

Sales;



9.Types of Joins? With example and usecase. All the number of records return and exact records.

INNER JOIN:

Returns rows from both tables that have matching values in the specified columns. If there is no match, those rows are excluded from the result set.

LEFT JOIN (or LEFT OUTER JOIN):

Returns all rows from the left table and the matched rows from the right table. If there is no match, NULL values are returned for the columns from the right table.

RIGHT JOIN (or RIGHT OUTER JOIN):

Returns all rows from the right table and the matched rows from the left table. If there is no match, NULL values are returned for the columns from the left table.

FULL JOIN (or FULL OUTER JOIN):

Returns all rows when there is a match in either the left or right table. If there is no match, NULL values are returned for the columns from the table without a match.

CROSS JOIN:

Returns the Cartesian product of the two tables, i.e., all possible combinations of rows from both tables. There is no specific condition for matching rows.

CREATE TABLE table\_a (id INT, name VARCHAR(50) );

CREATE TABLE table\_b (id INT, color VARCHAR(50));

INSERT INTO table\_a VALUES (1, 'apple');

INSERT INTO table\_a VALUES (1, 'apple');

INSERT INTO table\_a VALUES (1, 'apple');

INSERT INTO table\_a VALUES (1, 'apple');

INSERT INTO table\_a VALUES (2, 'Banana');

INSERT INTO table\_a VALUES (2, 'Banana');

INSERT INTO table\_a VALUES (3, 'Guava');

INSERT INTO table\_b VALUES (1, 'Red');

INSERT INTO table\_b VALUES (2, 'Yellow');

INSERT INTO table\_b VALUES (3, 'Green');

1. LEFT JOIN:

SELECT table\_a.id, table\_a.name, table\_b.color

FROM table\_a

LEFT JOIN table\_b ON table\_a.id = table\_b.id;



2. RIGHT JOIN

SELECT table\_a.id, table\_a.name, table\_b.color

FROM table\_a

RIGHT JOIN table\_b ON table\_a.id = table\_b.id;



3. INNER JOIN

SELECT table\_a.id, table\_a.name, table\_b.color

FROM table\_a

INNER JOIN table\_b ON table\_a.id = table\_b.id;



4. FULL OUTER JOIN:

SELECT table\_a.id, table\_a.name, table\_b.color

FROM table\_a

FULL OUTER JOIN table\_b ON table\_a.id = table\_b.id;



5. CROSS JOIN:

SELECT table\_a.id, table\_a.name, table\_b.color

FROM table\_a

CROSS JOIN table\_b;







10. Use case when self-join is required.

Hierarchical Data:

Suppose you have a table representing an organizational structure, where each row contains information about an employee and their manager's ID. You might need to retrieve details about both the employees and their managers. In this case, you can use a self-join to join the table to itself, matching each employee's manager ID with another employee's ID. This allows you to construct a hierarchical view of the organization, showing relationships between employees and their managers.

Comparing Records:

Consider a scenario where you have a table storing data about employees, including their performance ratings or sales figures over time. You might want to compare an employee's current performance with their past performance or with that of their peers. By performing a self-join on the same table, you can match each employee's records with those of others, allowing you to analyse trends, identify top performers, or detect anomalies within the dataset. In both cases, a self-join provides a way to relate rows within the same table, enabling you to query and analyse data in a variety of ways.

11.What is subquery?

A subquery, also known as a nested query or inner query, is a query nested within another SQL statement, such as SELECT, INSERT, UPDATE, or DELETE. Subqueries are enclosed within parentheses and can be used in various parts of a SQL statement, including the WHERE clause, FROM clause, HAVING clause, or even in the SELECT list.

Subqueries are primarily used to retrieve data from one or more tables based on specific conditions defined in the outer query. The result of the subquery can be used by the outer query to further filter, join, or manipulate data. Subqueries can return a single value, a single row, multiple rows, or an entire result set, depending on the context in which they are used.

12.What is corelated subquery?

A correlated subquery is a type of subquery in SQL where the inner query references one or more columns from the outer query. Unlike a regular subquery, which can run independently, a correlated subquery depends on the outer query for its execution.

Correlated subqueries are typically used when you need to evaluate conditions from the outer query within the inner query. This allows for dynamic filtering or comparison based on the current row being processed by the outer query.

13.What is CTE?

CTE stands for Common Table Expression. It's a temporary named result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. CTEs are defined using the WITH keyword and offer a way to write more readable and modular SQL queries, especially for complex queries that involve multiple steps or recursion.

Example:

WITH SalesPerRegion AS (

SELECT Region, SUM(SalesAmount) AS TotalSales

FROM Sales

GROUP BY Region

)

SELECT \* FROM SalesPerRegion;

In this example, "SalesPerRegion" is the name of the CTE, which calculates the total sales amount per region. This CTE is then referenced in the subsequent SELECT statement to retrieve the results.

14.What is derived table?

A derived table, also known as an inline view or subselect, is a virtual table that is derived from the result of a SELECT statement. It's typically used within the FROM clause of a SQL query and behaves like a regular table, but its definition exists only for the duration of that query's execution.

Derived tables are useful when you need to perform complex calculations or transformations on existing data before using it in the main query. They can also help simplify queries by breaking down complex logic into smaller, more manageable parts.

Example:

SELECT \*

FROM (

SELECT product\_id, SUM(quantity) AS total\_quantity

FROM orders

GROUP BY product\_id

) AS product\_summary;

In this example, the inner SELECT statement calculates the total quantity of each product from the "orders" table. The result of this inner query is then treated as a derived table named "product\_summary," which is used in the outer SELECT statement to retrieve the product IDs and their total quantities.

15.Find third highest employee based on salary?

SELECT employee\_id, employee\_name, salary

FROM employees

ORDER BY salary DESC

LIMIT 1 OFFSET 2;

-- This query orders the employees by their salary in descending order

-- Then skips the first two rows using OFFSET 2.

-- Finally, it limits the result to only one row, which represents the third-highest-paid employee.

16.Find third highest employee based on salary per department?

WITH RankedEmployees AS (

SELECT employee\_id, employee\_name, salary, department\_id,

ROW\_NUMBER() OVER (PARTITION BY department\_id ORDER BY salary DESC) AS rank

FROM employees

)

SELECT employee\_id, employee\_name, salary, department\_id

FROM RankedEmployees

WHERE rank = 3;

-- We use a Common Table Expression (CTE) named RankedEmployees to assign a rank to each employee within their department based on their salary.

-- The ROW\_NUMBER() function is used with the PARTITION BY clause to partition the result set by department and order the employees within each department by their salary in descending order.

-- Finally, we select the employees where the rank is equal to 3, indicating the third-highest-paid employee per department.

17.How to find duplicate values in a single column?

SELECT column\_name, COUNT(\*) AS duplicate\_count

FROM table\_name

GROUP BY column\_name

HAVING COUNT(\*) > 1;

This query groups the rows by the values in the specified column, counts the occurrences of each value, and then filters for groups with a count greater than 1, indicating duplicates. It returns the duplicate values along with their counts.

18.How to find duplicate values in a multiple column?

SELECT column1, column2, ..., COUNT(\*) AS duplicate\_count

FROM table\_name

GROUP BY column1, column2, ...

HAVING COUNT(\*) > 1;

This query groups the rows by the combination of values in the specified columns, counts the occurrences of each combination, and then filters for groups with a count greater than 1, indicating duplicates. It returns the duplicate combinations of values along with their counts.

19.What are ACID properties? give example for each property

ACID properties

ACID is an acronym for Atomicity, Consistency, Isolation, and Durability.

Atomicity: Ensures that either all the DML Statements (i.e. insert, update, delete) inside a transaction are completed successfully or all of them are rolled back.

Ex: Consider a banking transaction where you transfer money from your savings account to your checking account. Atomicity ensures that if the funds are successfully debited from your savings account, they will be credited to your checking account. If there's an error during the transaction, such as insufficient funds, neither the debit nor the credit will occur.

Consistency: Ensures that the database data is in a consistent state before the transaction started and also left the data in a consistent state after the transaction is completed.

Ex: Imagine a database where you're updating a customer's information. Let's say the customer's age must be greater than 18 due to legal constraints. If you try to update the age to 17, the database system will maintain consistency by rejecting the update, ensuring that the database remains in a valid state.

Isolation: Ensures that the intermediate state of a transaction is invisible to other transactions. The Data modifications made by one transaction must be isolated from the data modifications made by all other transactions.

Ex: Suppose two users are simultaneously attempting to update the same bank account balance. Isolation ensures that each transaction is executed independently, without interference from the other. For example, if one user is withdrawing money while the other is depositing, both transactions will be processed without one affecting the outcome of the other.

Durability: Ensures that once the transaction is successfully completed, then the changes made to the database will be permanent. Even if there is a system failure or power failure or any abnormal changes, it should safeguard the committed data.

Ex: After a successful transaction, such as updating a customer's address, the changes are permanently stored in the database. Even in the event of a system crash or failure, the changes will be preserved. When the system is restored, the updated address will still be present, ensuring that data remains intact and recoverable.

20.Diff between union and union all

The main difference between UNION and UNION ALL lies in how they handle duplicate rows:

UNION:

- Eliminates duplicate rows from the combined result set.

- Performs a distinct operation, which means it only includes unique rows in the final result.

- Slower compared to UNION ALL due to the overhead of removing duplicates.

UNION ALL:

- Includes all rows from both result sets, including duplicates.

- Does not perform any duplicate removal operation.

- Faster than UNION because it does not incur the overhead of identifying and eliminating duplicates.

21.Diff between primary key and unique key

Primary Key:

- A primary key is a column or a set of columns that uniquely identifies each row in a table.

- It does not allow null values because it must uniquely identify each record.

- There can be only one primary key constraint defined for a table.

- Automatically creates a unique index on the column(s) it's applied to.

- Used to establish relationships between tables in a database.

- Generally, it's the preferred way to identify records within a table.

Unique Key:

- A unique key constraint ensures that all values in a column or a set of columns are unique.

- Unlike primary keys, unique keys can contain null values, but only one null value is allowed per column.

- Multiple unique key constraints can be defined within a table.

- Doesn't automatically create an index, but you can manually create one for performance reasons.

- Often used to enforce business rules or constraints that require certain columns to have unique values but are not used for identifying records uniquely.

22.Diff between truncate and delete

The DELETE statement is used when we want to remove some or all of the records from the table, while the TRUNCATE statement will delete entire rows from a table. DELETE is a DML command as it only modifies the table data, whereas the TRUNCATE is a DDL command.

23.Diff between havin and where

WHERE clause:

- The WHERE clause is used to filter rows based on individual row values.

- It is applied before any grouping of data.

- Conditions specified in the WHERE clause are applied to each individual row.

- It cannot be used with aggregate functions.

Example: SELECT \* FROM students WHERE age > 18; This query selects all students whose age is greater than 18.

HAVING clause:

- The HAVING clause is used to filter groups of rows based on aggregate values (e.g., COUNT, SUM, AVG).

- It is applied after data has been grouped using the GROUP BY clause.

- Conditions specified in the HAVING clause are applied to groups of rows, not individual rows.

- It can be used only with aggregate functions.

Example: SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 5000; This query selects departments where the average salary is greater than 5000.

24.SQL query execution order.

FROM: The tables or views specified in the FROM clause are accessed and joined, creating a working set of data.

WHERE: The WHERE clause filters the rows in the working set based on specified conditions.

GROUP BY: If specified, the rows in the filtered set are then grouped into summary rows based on common values in the specified columns.

HAVING: The HAVING clause filters the grouped rows based on specified conditions.

SELECT: The SELECT clause specifies which columns or expressions to include in the query result.

ORDER BY: If specified, the rows in the result set are then sorted based on the specified column(s) or expression(s).

LIMIT/OFFSET: If specified, limits the number of rows returned and optionally skips a certain number of rows.

25.What are indexes? Types of Indexes and their differences.

Indexes in databases are data structures that enhance the speed of data retrieval operations by providing quick access paths to rows based on the values of specific columns. They act as a roadmap to efficiently locate data within a table.

1. B-tree Index:

Utilizes a balanced tree structure.

Suitable for range queries and equality comparisons.

Efficient for most types of queries.

Widely used in most relational databases.

2. Hash Index:

Utilizes a hash table data structure.

Efficient for equality comparisons but not for range queries.

May suffer from collisions, affecting performance.

Suitable for in-memory databases and for columns with low cardinality.

3. Bitmap Index:

Represents each distinct value in a column as a bitmap.

Efficient for columns with a small number of distinct values.

Suitable for columns with low cardinality and where data changes infrequently.

4.Full-Text Index:

Optimized for fast text searches on large text fields.

Enables advanced search capabilities like keyword searches and relevance ranking.

Useful for applications requiring extensive text search functionalities.

5.Spatial Index:

Specialized index for optimizing queries involving spatial data types.

Enables efficient spatial operations like distance-based searches and spatial joins.

Used in Geographic Information Systems (GIS) and location-based applications.

6.Composite Index:

Created on multiple columns.

Useful for queries involving multiple columns in the WHERE clause.

Enables efficient retrieval based on combinations of column values.

26.What is surrogate key? Give example where you used it and how.

A surrogate key is a unique identifier assigned to each record in a database table. Unlike natural keys, which are based on existing data attributes, surrogate keys are system-generated and have no inherent meaning. They are commonly used in database design to ensure uniqueness and improve performance.

Example:

Let's say you're designing a database for an online retail store. You have a table named "products" to store information about the products sold. Instead of using a natural key, such as the product name or SKU (Stock Keeping Unit), you decide to use a surrogate key as the primary key for the "products" table.

CREATE TABLE products (

product\_id INT PRIMARY KEY AUTO\_INCREMENT,

product\_name VARCHAR(100),

price DECIMAL(10,2),

description TEXT

);

In this example:

-The "product\_id" column serves as the surrogate key for the "products" table.

-It is defined as an INT data type and is designated as the primary key.

-The AUTO\_INCREMENT keyword ensures that each new record inserted into the table will automatically be assigned a unique "product\_id" value, incrementing by one.

Using a surrogate key in this scenario ensures that each product record has a unique identifier, regardless of any changes to the product name or other attributes. This improves data integrity and simplifies database operations, such as joins and updates.