1. Types of commands and their examples.

**Data Manipulation Language (DML)** commands are used to manipulate data in the database. Examples include:

SELECT: Retrieve data from a table.

INSERT: Insert new data into a table.

UPDATE: Modify existing data in a table.

DELETE: Delete data from a table.

**Data Definition Language (DDL)** commands are used to define the structure of the database. Examples include:

CREATE: Create a new table, view, or other database object.

ALTER: Modify the structure of an existing table.

DROP: Delete a table or other database object.

Data Control Language (DCL) commands are used to control access to the database. Examples include:

GRANT: Grant privileges and permissions to users.

REVOKE: Revoke privileges and permissions from users.

Transaction Control Language (TCL) commands are used to manage transactions in the database. Examples include:

COMMIT: Save changes made within a transaction.

ROLLBACK: Undo changes made within a transaction.

SAVEPOINT: Set a savepoint within a transaction.

**What is Normalization and denormalization?**

Normalization is the process of organizing data in a database to minimize redundancy and dependency. It involves dividing large tables into smaller, more manageable tables and establishing relationships between them. The main goal of normalization is to eliminate data anomalies and improve data integrity.

Denormalization is the opposite of normalization. It involves combining tables and reintroducing redundancy to improve performance in certain scenarios. Denormalization is used when there is a need for faster data retrieval, and it sacrifices some aspects of data integrity and storage efficiency in favor of performance.

**Explain 1NF, 2NF, 3NF?**

First Normal Form (1NF): In 1NF, the data is organized into tables, and each column contains only atomic (indivisible) values. There should be no repeating groups, and each row should be uniquely identifiable. Example: A table with separate columns for each student's subject grades.

Second Normal Form (2NF): In 2NF, the table is in 1NF, and all non-key attributes are dependent on the entire primary key. There should be no partial dependencies, where an attribute depends on only a part of the primary key. Example: A table with separate tables for students and their respective subject grades.

Third Normal Form (3NF): In 3NF, the table is in 2NF, and there are no transitive dependencies. Non-key attributes should depend only on the primary key, not on other non-key attributes. Example: A table with separate tables for students, subjects, and grades, with dependencies properly defined.

**Share a use case where you had to do denormalization in a database?**

A use case for denormalization could be in a reporting system where there is a need for faster data retrieval. By combining multiple normalized tables into a single denormalized table, the reporting queries can run more efficiently, improving performance. However, it's important to consider the trade-offs in terms of data redundancy and maintenance complexity.

**What is a 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 enforces entity integrity and ensures that each row has a unique identifier. Example: A "user\_id" column in a "Users" table.

Foreign Key: A foreign key is a column or a set of columns in a table that refers to the primary key in another table. It establishes a relationship between two tables and enforces referential integrity. It ensures that the values in the foreign key column(s) match the values in the primary key column(s) of the referenced table. Example: A "user\_id" column in an "Orders" table that refers to the "user\_id" column in the "Users" table.

**What is an alternate and candidate key?**

Alternate Key: An alternate key is a candidate key that is not chosen to be the primary key. In other words, it is a unique identifier for a table, but it is not selected as the primary means of identifying records. Example: In a "Users" table, both the "email" and "username" columns could be alternate keys if they are unique and can be used to identify records.

Candidate Key: A candidate key is a column or a set of columns that can uniquely identify records in a table. It meets the criteria for being a primary key but is not necessarily chosen as the primary key. Example: In a "Users" table, the "user\_id" column could be a candidate key if it is unique and can identify records uniquely.

**What are window functions?**

Window functions are a set of functions in SQL that perform calculations across a set of rows and return a result for each row. They allow you to compute values based on a subset of rows within a window defined by a specific ordering or partitioning of the data. Window functions are often used for tasks like calculating running totals, ranking rows, and calculating moving averages.

**Explain Ranking Functions?**

Ranking functions are a type of window functions that assign a rank or position to each row within a specific ordering of the data. Some commonly used ranking functions are:

RANK(): Assigns a unique rank to each distinct row, leaving gaps when there are ties.

DENSE\_RANK(): Assigns a unique rank to each distinct row, without any gaps in the ranking for ties.

ROW\_NUMBER(): Assigns a unique number to each row, without any consideration for ties.

**Types of Joins?**

INNER JOIN: Returns only the matching rows from both tables based on the specified join condition.

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

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

FULL JOIN: Returns all rows from both tables. If there is no match, NULL values are returned for columns in the non-matching table.

Use case when self-join is required:

A common use case for self-join is when you have a table that contains hierarchical or recursive data, such as an employee table where each employee has a manager who is also an employee. By joining the table to itself using the appropriate join condition, you can retrieve information about employees and their managers.

**What is a subquery?**

A subquery is a query nested within another query. It can be used to retrieve data that depends on the results of another query. The result of the subquery is then used as a condition or value in the outer query.

**What is a correlated subquery?**

A correlated subquery is a subquery that refers to a column from the outer query. It uses values from the outer query to determine the result of the subquery. Correlated subqueries are evaluated for each row of the outer query, which can impact performance compared to non-correlated subqueries.

**What is a CTE (Common Table Expression)?**

A Common Table Expression (CTE) is a temporary named result set that can be used within a SQL statement. It allows you to define a query block and reference it multiple times within the same query. CTEs enhance the readability and reusability of complex SQL queries.

**Find third highest employee based on salary?**

SELECT employee\_name, salary

FROM employees

ORDER BY salary DESC

LIMIT 1 OFFSET 2;

**Find third highest employee based on salary?**

**SELECT employee\_name, salary, department**

**FROM employees**

**WHERE department = 'department\_name'**

**ORDER BY salary DESC**

**LIMIT 1 OFFSET 2;**

**How to find duplicate values in a single column?**

SELECT column\_name, COUNT(\*)

FROM table\_name

GROUP BY column\_name

HAVING COUNT(\*) > 1;

**How to find duplicate values in a multiple column?**

SELECT column1, column2, COUNT(\*)

FROM table\_name

GROUP BY column1, column2

HAVING COUNT(\*) > 1;

**What are ACID properties?**

ACID stands for Atomicity, Consistency, Isolation, and Durability. These properties are the key characteristics that ensure reliability and data integrity in database transactions:

Atomicity: A transaction is treated as a single, indivisible unit of work. It either completes successfully, or all changes are rolled back.

Consistency: A transaction brings the database from one valid state to another. It ensures that data follows predefined rules and constraints.

Isolation: Concurrent transactions are isolated from each other, and their intermediate results are not visible to other transactions until they are committed.

Durability: Once a transaction is committed, its changes are permanent and will survive any subsequent system failures.

**Difference between UNION and UNION ALL?**

UNION: UNION combines the result sets of two or more SELECT statements into a single result set. Duplicate rows are removed from the final result set. The column names and data types must match in the corresponding positions of each SELECT statement.

UNION ALL: UNION ALL also combines the result sets of two or more SELECT statements into a single result set. However, it does not remove duplicate rows. All rows from each SELECT statement are included in the final result set, regardless of duplicates.

Difference between primary key and unique key:

A primary key is a column or a set of columns that uniquely identifies each row in a table. It must be unique and cannot contain NULL values. There can be only one primary key per table.

A unique key is a column or a set of columns that enforces uniqueness but allows NULL values. It ensures that the values in the key column(s) are unique among all rows in the table, but multiple NULL values are allowed. A table can have multiple unique keys.

**Difference between TRUNCATE and DELETE?**

TRUNCATE: TRUNCATE removes all rows from a table, resetting the table to its initial state. It is a DDL statement and is faster than DELETE because it does not generate individual row-level delete operations. TRUNCATE cannot be rolled back.

DELETE: DELETE removes one or more rows from a table based on a condition. It is a DML statement and generates individual row-level delete operations. DELETE can be rolled back if executed within a transaction.

**SQL query execution order?**

The order of execution in a SQL query is as follows:

FROM: Specifies the table(s) to retrieve data from.

JOIN: Joins the specified tables based on the join conditions.

WHERE: Filters the rows based on the specified conditions.

GROUP BY: Groups the rows based on the specified columns.

HAVING: Filters the groups based on the specified conditions.

SELECT: Selects the columns to be included in the result set.

DISTINCT: Removes duplicate rows from the result set (if specified).

ORDER BY: Sorts the result set based on the specified columns.

LIMIT/OFFSET: Limits the number of rows returned or skips a specified number of rows (if specified).

UNION/INTERSECT/EXCEPT: Combines or compares result sets (if specified).