1.Types of commands and their examples.

In SQL, commands categorized into four types:

1. Data Definition Language (DDL)

2. Data Manipulation Language (DML)

3. Data Query Language (DQL)

4. Data Control Language (DCL).

1.Data Definition Language (DDL): DDL commands are used to define, modify, and delete the structure of database objects like tables, indexes, views, etc.

a. CREATE TABLE:

CREATE TABLE employees (

id INT PRIMARY KEY,

name VARCHAR(100),

salary DECIMAL(10, 2)

);

b. ALTER TABLE:

ALTER TABLE employees ADD COLUMN department VARCHAR(50);

c. DROP TABLE: Deletes an existing table.

DROP TABLE employees;

2. Data Manipulation Language (DML):

DML commands are used to manipulate data within database objects like table

a.INSERT INTO: Inserts new records into a table.

INSERT INTO employees (id, name, salary) VALUES (1, 'John Doe', 50000);

b. UPDATE: Updates existing records in a table.

UPDATE employees SET salary = 55000 WHERE id = 1;

c. DELETE FROM: Deletes records from a table.

DELETE FROM employees WHERE id = 1;

3. Data Query Language (DQL):

DQL commands are used to retrieve data from the database.

SELECT: Retrieves data from one or more tables.

SELECT \* FROM employees;

4. Data Control Language (DCL):

DCL commands are used to control access to data within the database.

GRANT: Grants permissions to users or roles.

GRANT SELECT ON employees TO user1;

REVOKE: Revokes permissions from users or roles.

REVOKE SELECT ON employees FROM user1;

1. **What is Normalization and denormalization?**

Normalization:

Organizing data to minimize redundancy and dependency.

Achieved by breaking down large tables into smaller ones and defining relationships.

Helps ensure data integrity and simplifies maintenance.

Denormalization:

Introduces redundancy to improve performance or simplify data retrieval.

Involves combining related tables, duplicating data, or precomputing aggregated values.

Enhances read performance but can increase storage and complexity.

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

**First Normal Form (1NF):**

Requires that each column in a table must contain atomic (indivisible) values.

Eliminates repeating groups and ensures that each cell holds a single value.

Example: Splitting a column containing multiple values into separate columns or rows.

**Second Normal Form (2NF):**

Builds on 1NF by ensuring that every non-key attribute is fully functionally dependent on the primary key.

Eliminates partial dependencies by moving subsets of data into separate tables.

Example: Breaking a table into two separate tables when non-key attributes depend on only part of the primary key.

**Third Normal Form (3NF):**

Further refines the database structure by removing transitive dependencies.

Ensures that non-key attributes depend only on the primary key and not on other non-key attributes.

Example: Splitting a table into multiple tables to eliminate attributes that depend on other non-key attributes.

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

Use case: E-commerce reporting system.

Scenario: Need for fast analytics queries.

Solution: Denormalization of transactional data.

Benefits: Improved query performance for analytics.

Considerations: Increased storage, data synchronization.

1. What is primary key and foreign key?

Primary Key:

A primary key is a unique identifier for each record in a table.

It ensures that each row in the table is uniquely identifiable.

Primary keys are used to enforce entity integrity and establish relationships between tables.

Foreign Key:

A foreign key is a column or a set of columns in a table that establishes a relationship with a primary key or a unique key in another table.

It ensures referential integrity by enforcing a link between the data in two related tables.

Foreign keys are used to create relationships between tables and maintain data consistency.

1. what is alternate and candidate key?

**Candidate Key:**

A candidate key is a column or set of columns in a table that can uniquely identify each row.

It satisfies uniqueness and irreducibility criteria.

One candidate key becomes the primary key, while others are alternate keys.

**Alternate Key:**

An alternate key is any candidate key that is not chosen as the primary key.

It provides an alternative choice for uniquely identifying rows.

Alternate keys ensure data integrity and uniqueness, like the primary key.

1. What are window functions?

Perform calculations across a set of rows related to the current row.

Operate within a window frame defined by a partition or ordering.

Return a value for each row in the result set.

Examples include ROW\_NUMBER(), RANK(), SUM() OVER(), etc.

Used for analytical calculations like ranking, aggregation, moving averages, etc.

1. Explain Ranking Functions? Given a small table , write the output.

Ranking Functions:

Assign a rank to each row based on specified criteria.

Common ranking functions include ROW\_NUMBER(), RANK(), and DENSE\_RANK().

ROW\_NUMBER() assigns a unique number to each row.

RANK() assigns ranks with gaps for ties.

DENSE\_RANK() assigns consecutive ranks without gaps for ties.

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

Types of Joins:

INNER JOIN:

Returns records that have matching values in both tables.

Example:

SELECT \* FROM table1 INNER JOIN table2 ON table1.id = table2.id;

Usecase: Retrieving only the records that have related entries in both tables.

LEFT JOIN (or LEFT OUTER JOIN):

Returns all records from the left table and matching records from the right table.

Example:

SELECT \* FROM table1 LEFT JOIN table2 ON table1.id = table2.id;

Usecase: Retrieving all records from one table and related records from another table, regardless of whether a match exists.

RIGHT JOIN (or RIGHT OUTER JOIN):

Returns all records from the right table and matching records from the left table.

Example:

SELECT \* FROM table1 RIGHT JOIN table2 ON table1.id = table2.id;

Usecase: Similar to LEFT JOIN, but ensures all records from the right table are included.

FULL JOIN (or FULL OUTER JOIN):

Returns all records when there is a match in either left or right table.

Example:

SELECT \* FROM table1 FULL JOIN table2 ON table1.id = table2.id;

Usecase: Retrieving all records from both tables, regardless of whether a match exists.

CROSS JOIN:

Returns the Cartesian product of the two tables, i.e., all possible combinations of rows.

Example:

SELECT \* FROM table1 CROSS JOIN table2;

Usecase: Used to generate all possible combinations of rows from two tables.

For the number of records returned and exact records, it depends on the specific data in the tables and the join conditions used in the query. Without knowing the actual data, it's difficult to provide the exact number of records and the specific records returned for each join type. Each join type behaves differently based on the relationships between the tables and the data in them.

1. Use case when self join is required.

Use Case for Self-Join: Hierarchical Data

When you need to represent parent-child relationships within the same table.

Commonly used in organizational charts, bill of materials, network graphs, etc.

Example: Retrieving employee details along with their manager's details from an "employees" table.

1. What is subquery?

A subquery, also known as a nested query or inner query, is a query nested within another SQL query.

It is enclosed within parentheses and is used to return data that will be used in the main query's condition or calculation.

Subqueries can be used in SELECT, INSERT, UPDATE, and DELETE statements.

They can return a single value, a single row, multiple rows, or an entire result set.

Subqueries are often used to filter results, perform calculations, or retrieve data from related tables.

1. What is corelated subquery?

Nested query where the inner query references columns from the outer query.

Inner query executed repeatedly for each row processed by the outer query.

Used for filtering, calculations, or aggregates based on values from outer query.

Offers flexibility but may impact performance and query readability.

1. What is CTE?

Common Table Expression (CTE):

A temporary named result set that can be referenced within a SQL statement, usually defined using the WITH keyword.

Provides a way to simplify complex queries by breaking them down into smaller, more manageable parts.

Can be self-referencing (recursive CTE), allowing hierarchical or recursive queries.

Commonly used for recursive queries, data manipulation, and simplifying complex queries.

1. What is derived table?

A temporary table created within the scope of a SQL query.

Not stored in the database schema and exists only for the duration of the query execution.

Generated on-the-fly from the result of a subquery within the main query.

Commonly used for simplifying complex queries by breaking them into smaller, more manageable parts.

1. Find third highest employee based on salary?

To find the third highest employee based on salary:

Use the ROW\_NUMBER() window function to assign row numbers based on descending salary order.

Select rows where the row number is 3.

Example SQL query:

SELECT employee\_name, salary

FROM (

SELECT employee\_name, salary, ROW\_NUMBER() OVER (ORDER BY salary DESC) AS rn

FROM employees

) AS ranked\_employees

WHERE rn = 3;

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

To find the third highest employee based on salary per department:

Use a Common Table Expression (CTE) to assign row numbers partitioned by department based on descending salary order.

Select rows where the row number is 3.

Example SQL query:

WITH ranked\_employees AS (

SELECT

employee\_name,

salary,

department\_id,

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

FROM employees

)

SELECT

employee\_name,

salary,

department\_id

FROM ranked\_employees

WHERE rn = 3;

This query returns the name, salary, and department ID of the employee with the third highest salary within each department. Adjust the ORDER BY clause as needed for specific tie-breaking criteria.

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

SELECT column\_name

FROM table\_name

GROUP BY column\_name

HAVING COUNT(\*) > 1;

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

To find duplicate values across multiple columns:

SELECT column1, column2, ..., columnN

FROM table\_name

GROUP BY column1, column2, ..., columnN

HAVING COUNT(\*) > 1;

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

Atomicity: Transactions are treated as a single unit of work that either succeeds completely or fails completely. Example: A bank transfer deducts money from one account and adds it to another; if either deduction or addition fails, the entire transaction is rolled back.

Consistency: Transactions bring the database from one valid state to another, preserving integrity constraints. Example: In a banking system, a withdrawal transaction reduces an account balance, ensuring the resulting balance adheres to defined constraints.

Isolation: Concurrent transactions execute independently without interference, providing each transaction with a consistent view of the data. Example: Two concurrent withdrawals from the same account are processed without one transaction seeing the uncommitted changes of the other.

Durability: Committed transactions' changes are permanent and survive system failures, ensuring data persistence. Example: After a successful transfer, the transaction's changes are durably stored, persisting even if the system crashes immediately afterward.

20.Diff between union and union all

UNION is used to combine the results of two or more SELECT statements and removes duplicate rows from the result set.

It only returns distinct rows, eliminating duplicates.

It has a higher computational cost due to the need to remove duplicates.

UNION ALL:

UNION ALL also combines the results of two or more SELECT statements but includes all rows, including duplicates, in the result set.

It does not remove duplicate rows; it retains all rows from each SELECT statement.

It is faster than UNION since it does not need to check for duplicates.

21. Diff between primary key and unique key

**Primary Key:**

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

It must contain unique values and cannot contain NULL values.

There can be only one primary key constraint defined per table.

Automatically creates a unique index to enforce uniqueness and to speed up data retrieval.

**Unique Key:**

Unique key is a column or a set of columns that ensures that all values in the column(s) are unique.

Unlike primary key, a unique key column can contain NULL values, but only one NULL value is allowed.

There can be multiple unique key constraints defined per table.

Unique key constraints also create unique indexes to enforce uniqueness.

22. Diff between truncate and delete

**DELETE**:

Removes specific rows from a table based on conditions.

Generates entry in transaction log for each deleted row.

Can be rolled back.

Does not reset identity columns by default.

**TRUNCATE**:

Removes all rows from a table.

Resets identity columns to initial seed value.

Faster than DELETE, as it does not generate transaction log entries.

Cannot be rolled back.

1. Diff between having and where

**WHERE clause:**

Used to filter rows before grouping or aggregation.

Applies filtering conditions to individual rows of the base table.

Executed before the grouping and aggregation operations.

Cannot be used with aggregate functions.

**HAVING clause:**

Used to filter groups after grouping or aggregation.

Applies filtering conditions to groups formed by the GROUP BY clause.

Executed after the grouping and aggregation operations.

Can be used with aggregate functions to filter groups based on aggregate results.

1. SQL query execution order.

**FROM**: Tables are accessed and joined to form the initial result set.

**WHERE**: Rows are filtered based on specified conditions.

**GROUP BY**: Rows are grouped together based on common values.

**HAVING**: Groups are filtered based on specified conditions.

**SELECT**: Columns and expressions are evaluated.

**DISTINCT**: Duplicate rows are removed.

**ORDER BY**: Rows are sorted based on specified criteria.

**LIMIT/OFFSET**: The result set may be limited or offset as needed.

This sequence provides a structured approach to processing and manipulating data in SQL queries.

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

Indexes are like a table of contents in a book, helping to find information quickly.

They speed up database queries by storing sorted copies of columns with pointers to corresponding rows.

Types of Indexes:

Single-Column Index: Created on one column.

Composite Index: Created on multiple columns.

Unique Index: Ensures each value in the indexed column(s) is unique.

Non-Unique Index: Allows duplicate values in the indexed column(s).

Clustered Index: Physically orders table rows based on indexed column(s).

Non-Clustered Index: Creates a separate index structure.

Bitmap Index: Stores bitmaps for each possible value.

Differences:

Single-column vs. composite: Number of columns indexed.

Unique vs. non-unique: Enforcing uniqueness.

Clustered vs. non-clustered: Physical vs. separate index structure.

Bitmap index: Suitable for low-cardinality columns.

Each type serves different needs, helping databases efficiently find and retrieve data.

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

An artificially created unique identifier in a database table.

Typically an integer value generated automatically by the database system.

Used as the primary key in the table.

Ensures stable and unique identification of records, regardless of changes to other attributes.

Simplifies database operations and enhances performance for queries and joins.

Maintains data integrity and consistency within the database.