SQL

1. Types of commands and their examples.
   a. **Data Definition Language (DDL):**
      i. **Create**
      ii. **Drop**
      iii. **alter**
   b. **Data Manipulation Language (DML):**
      i. **Select**
      ii. **Insert**
      iii. **Update**
      iv. **delete**
   c. **Data Control Language (DCL):**
      i. **Grant**
      ii. Revoke
   d. **Transaction Control Language (TCL):**
      i. **Commit**
      ii. **Rollback**
2. What is Normalization and denormalization?
   a. **Normalization:** Normalization is a process in database design that eliminates data redundancy and improves data integrity by organizing data into multiple related tables. It aims to reduce data duplication and dependency, ensuring that each piece of information is stored in only one place. Normalization follows a set of rules (known as normal forms) to design a well-structured and efficient database schema.
   b. **Denormalization:** Denormalization is the process of intentionally adding redundancy to a relational database design. It involves combining multiple tables and duplicating data to optimize query performance. Denormalization is often used in situations where read performance is critical, but at the cost of increased storage usage and potential data consistency risks.
3. Explain 1NF, 2NF, 3NF.
   a. **First Normal Form (1NF):** First Normal Form (1NF) is the foundational level of normalization. It sets the basic rules for organizing data in a database table. To be in 1NF, a table must meet the following criteria:
      i. Atomic values: Each column should contain only atomic (indivisible) values. This means that each value in a column should represent a single data element rather than a collection or a set of values.

        ii. Unique column names: Every column in the table should have a unique name, avoiding any ambiguity and ensuring that each column holds distinct data.

    b. **Second Normal Form (2NF):** Second Normal Form (2NF) builds upon 1NF and deals with the concept of functional dependency. To meet 2NF, a table must satisfy the following conditions:
        i. Be in 1NF.
        ii. Have a primary key defined.
        iii. All non-key columns depend on the whole primary key, not just part of it.
        iv. In simpler terms, 2NF ensures that each non-key column depends on the entire primary key rather than just a portion of it. It avoids partial dependencies within a table.

    c. **Third Normal Form (3NF):** Third Normal Form (3NF) extends the concept of functional dependency introduced in 2NF. A table is in 3NF if it satisfies the following conditions:
        i. Be in 2NF.
        ii. No transitive dependencies exist. Transitive dependency occurs when a non-key column depends on another non-key column rather than directly on the primary key.
        iii. In simple terms, 3NF ensures that all non-key columns depend only on the primary key and not on each other

4. Share use case where you had to do denormalization in database.
    a. One use case where denormalization may be necessary is in a reporting system or data warehouse. Let's say we have a normalized database schema for an e-commerce platform that includes separate tables for customers, orders, and products. The tables are structured in a way that enforces data integrity and reduces redundancy. However, when it comes to generating complex reports or analytics on large datasets, normalization can pose performance challenges due to the need for joins and aggregations across multiple tables. In such a scenario, denormalization can be applied to create a separate reporting table or view that combines the necessary information from various normalized tables. This denormalized structure can optimize query performance by reducing the need for extensive joins and simplifying the retrieval of data.

5. What is primary key and foreign key?
    a. **Primary Key**: A primary key is a unique identifier for each record in a table. It ensures the uniqueness and integrity of the data in a table. The primary key can consist of one or more columns, and its values

cannot be null. It is used to uniquely identify each record and is typically used for indexing and referencing purposes.

b. **Foreign Key**: A foreign key is a column or a set of columns in a table that references the primary key of another table. It establishes a relationship between two tables, known as a parent-child relationship. The foreign key value in one table is used to match the primary key value in another table, creating a link between the two tables. Foreign keys enforce referential integrity, ensuring that data in the child table is consistent with the data in the parent table.

6. what is alternate and candidate key?
    a. **Alternate Key**: An alternate key is a candidate key that is not selected as the primary key. In a table, multiple candidate keys may exist, and one of them is chosen as the primary key. The rest of the candidate keys become alternate keys. Alternate keys have unique values that can be used to identify records, but only one candidate key becomes the primary key.
    b. **Candidate Key**: A candidate key is a column or set of columns that can uniquely identify each record in a table. It is a potential candidate for being the primary key. A table can have multiple candidate keys, and the primary key is selected from these candidates. A candidate key must be unique and non-null, ensuring data integrity.
7. What are window functions?
    a. **Window Functions in SQL**: Window functions in SQL allow you to perform calculations on a subset or window of data within a table, as opposed to the entire table. These functions operate on a specific range or partition of rows defined by the window specification. Window functions help in performing complex calculations such as running totals, moving averages, rank assignments, and more. The syntax for window functions typically includes the OVER clause, which defines the window specification.
8. Explain Ranking Functions?
    a. **Ranking Functions in SQL**: Ranking functions in SQL assign a rank or position to each row within a result set based on a specified criterion. They can be used to retrieve the top or bottom records, identify duplicates, or assign a rank based on a certain attribute of data. Common ranking functions include RANK , DENSE_RANK , ROW_NUMBER , and NTILE .
9. Types of Joins?
    a. Inner join
    b. Outer join
    c. Left outer join
    d. Right outer join

    e. Cross join
    f. Self-join
10. Use case when self-join is required.
    a. A self-join in SQL is used when you need to join a table with itself, typically by correlating rows based on a common attribute or relationship within the same table. This approach can be useful in various scenarios, including Hierarchy or Parent-Child Relationships: When you have a table that represents hierarchical data, such as an employee table with a foreign key referencing the same table to represent managers or supervisors. A self-join can be used to retrieve information about employees and their respective managers.
11. What is subquery?
    a. A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within clauses, most commonly in the WHERE clause. It is used to return data from a table, and this data will be used in the main query as a condition to further restrict the data to be retrieved.
12. What is coorelated subquery?
    a. A correlated subquery is SQL, which is a subquery that depends on the outer query for its results. It is a type of subquery where the inner query references a column from the outer query, creating a relationship or correlation between the two queries. This allows the inner query to be executed repeatedly for each row processed by the outer query. The result of the corelated subquery is usually used to filter or evaluate conditions in the outer query.
13. What is CTE?
    a. In SQL, CTE stands for common table expression. A CTE is a temporary named result set that can be used within select, insert, update or delete statements. It allows you to define a query that can be referenced multiple times within a single SQL statement. ACTE is defined using the "WITH" keyword, followed by a name for the CTE and the query that defines it. Once defined, CTE can be referenced and treated like a table within the rest of the SQL statement.
14. Find third highest employee based on salary?
    a. Select employee_name, salary from (select employee_name, salary, DENSE_RANK () OVER (order by salary desc) as salary_rank from employees) as ranked_employees where salary_rank = 3;
15. Find third highest employee based on salary per department?
16.
    a. SELECT department, employee_name, salary FROM (SELECT department, employee_name, salary, DENSE_RANK () OVER (PARTITION BY department ORDER BY salary DESC) AS salary_rank FROM employees) AS ranked_employees WHERE salary_rank = 3;
17. How to find duplicate values in a single column?
    a. SELECT column_name, COUNT(column_name) AS count FROM table_name GROUP BY column_name HAVING COUNT(column_name) > 1;
18. How to find duplicate values in a multiple column?
    a. SELECT column1, column2, ..., columnN, COUNT(*) AS count FROM table_name GROUP BY column1, column2, ..., columnN HAVING COUNT(*) > 1;
19. What are ACID properties?

a. ACID is an acronym that stands for Atomicity, Consistency, Isolation, and Durability. These properties define the key characteristics of a reliable transactional database system.
    i. Atomicity: Atomicity ensures that a transaction is treated as a single, indivisible unit of work. It means that either all the operations within a transaction are executed successfully, or none of them are. If any part of the transaction fails, the entire transaction is rolled back, and the database returns to its original state.
    ii. Consistency: Consistency ensures that a transaction brings the database from one valid state to another. It enforces integrity constraints and rules defined on the database. In other words, the data remains consistent and follows all predefined rules throughout the execution of a transaction.
    iii. Isolation: Isolation guarantees that concurrent transactions do not interfere with each other. Each transaction operates as if it is executing in isolation, even when multiple transactions are executing concurrently. Isolation prevents uncommitted data from being visible to other transactions and avoids conflicts that may arise due to concurrent access.
    iv. Durability: Durability ensures that once a transaction is committed, its effects are permanent and will survive any subsequent failures, such as power outages or system crashes. The changes made by the committed transaction are stored in non-volatile storage, typically disk or flash memory, so that they can be recovered and restored in the event of a failure.

20. Diff between union and union all
21. The main difference between UNION and UNION ALL in SQL is how they handle duplicate rows during the combination of multiple result sets.
    a. UNION:
        i. The UNION operator combines the result sets of two or more SELECT statements into a single result set.
        ii. It removes duplicate rows from the final result set. If there are any rows that appear in multiple result sets, only one instance of the row will be included in the final result.
        iii. The columns in the SELECT statements must be of the same data type and in the same order.
        iv. The number of columns in each SELECT statement must be the same.
    a. UNION ALL:
    b. The UNION ALL operator also combines the result sets of multiple SELECT statements into a single result set.
    c. It does not remove duplicate rows from the final result set. All rows from each SELECT statement are included in the result, including any duplicates.
    d. The columns in the SELECT statements must be of the same data type and in the same order, similar to UNION.
    e. The number of columns in each SELECT statement must be the same.
22. Diff between primary key and unique key
    a. Primary Key:
        i. A primary key is a column or set of columns that uniquely identifies each row in a table.
        ii. It must have a unique value for each row and cannot contain null values.

iii. There can be only one primary key defined for a table.
iv. The primary key constraint automatically creates a clustered index in many database systems, physically organizing the data based on the key's values.
v. Primary keys are typically used as the basis for foreign key relationships between tables.

b. Unique Key:
c. A unique key is a column or set of columns that ensures each value is unique within a table.
d. Unlike a primary key, a unique key can have one or more null values, as long as the combination of non-null values remains unique.
e. A table can have multiple unique keys defined, each enforcing a different unique constraint.
f. Unique keys do not automatically create a clustered index, but they may create a non-clustered index in some database systems.
g. Unique keys are useful for enforcing uniqueness on columns that do not serve as the primary key but still need to have distinct values.

23. Diff between truncate and delete
    a. TRUNCATE:
        i. TRUNCATE is a Data Definition Language (DDL) statement.
        ii. TRUNCATE removes all rows from a table, effectively deleting all data.
        iii. It operates much faster than the DELETE statement, especially for large tables, as it deallocates the data pages without logging individual row deletions.
        iv. TRUNCATE resets any associated identity column values to their initial seed value.
        v. It cannot be rolled back, and the data deleted by TRUNCATE cannot be recovered.
        vi. TRUNCATE does not activate any triggers associated with the table.
        vii. Permissions required to use TRUNCATE are typically higher than those required for DELETE.
    b. DELETE:
        i. DELETE is a Data Manipulation Language (DML) statement.
        ii. DELETE removes specific rows or a subset of rows from a table based on a specified condition.
        iii. It removes rows one by one, generating individual log entries for each deletion, which can impact performance for large tables.
        iv. DELETE can be rolled back if executed within a transaction, allowing for recovery of the deleted data.
        v. DELETE triggers, if defined, are activated for each deleted row
        vi. Permissions required to use DELETE are typically lower than those required for TRUNCATE.
24. SQL query execution order.
    a. FROM clause: The query begins by identifying the table or tables from which the data will be retrieved.
    b. WHERE clause: The WHERE clause is applied next to filter the rows based on specified conditions. Rows that do not meet the conditions are excluded from the result set.

c. GROUP BY clause: If a GROUP BY clause is present, the rows are grouped based on the specified columns. This is often used in conjunction with aggregate functions like SUM, COUNT, AVG, etc.
d. HAVING clause: If a HAVING clause is present, it is applied to filter the grouped rows based on specified conditions. Similar to the WHERE clause, it excludes rows that do not meet the conditions.
e. SELECT clause: The SELECT clause is used to specify the columns that will be included in the result set. It operates on the filtered and grouped rows from the previous steps.
f. ORDER BY clause: If an ORDER BY clause is present, it sorts the result set based on the specified columns and their sort order (ascending or descending).
g. LIMIT/OFFSET clause: If a LIMIT clause (or equivalent) is specified, it restricts the number of rows returned. An OFFSET clause can be used to skip a certain number of rows from the beginning.