What is normalization? What is its need? Explain 1NF, 2NF, 3NF, and BCNF in detail.

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity and maintain data consistency. It involves breaking down a large table into smaller, more specific tables that are related to each other through common data elements.

The need for normalization arises when a database has a large amount of redundant data. This redundancy leads to inconsistencies, and it can be challenging to update or modify data in a redundant database. Normalization helps in resolving these issues by creating a database that is organized and more manageable.

First Normal Form (1NF): A table is in 1NF if it has no repeating groups, i.e., all its attributes contain atomic values. For example, if a table has a column that contains multiple values separated by commas, it is not in 1NF.

Second Normal Form (2NF): A table is in 2NF if it is in 1NF and all non-key attributes are dependent on the entire primary key. For example, if a table has a composite primary key, all non-key attributes should be dependent on the entire primary key, not just on a part of it.

Third Normal Form (3NF): A table is in 3NF if it is in 2NF and all non-key attributes are independent of each other. For example, if a table has attributes that are dependent on each other, they should be split into separate tables.

Boyce-Codd Normal Form (BCNF): A table is in BCNF if it is in 3NF and every determinant is a candidate key. A determinant is an attribute that determines the value of another attribute. For example, in a table where employee_id determines employee_name, employee_id is the determinant.

Explain entity relationship diagram and Explain all types of relationships with examples?

An entity-relationship (ER) diagram is a graphical representation of entities and their relationships to each other. It is used to design a database schema and represents the logical structure of the database.

There are three types of relationships between entities in an ER diagram:

One-to-One (1:1) Relationship: A one-to-one relationship exists when each record in Table A corresponds with exactly one record in Table B and vice versa. For example, an employee can have only one social security number, and a social security number can only be assigned to one employee. One-to-Many (1:N) Relationship: A one-to-many relationship exists when each record in Table A can correspond to many records in Table B, but each record in Table B corresponds to only one record in Table A. For example, a customer can have many orders, but each order can only belong to one customer. Many-to-Many (M:N) Relationship: A many-to-many relationship exists when each record in Table A can correspond to many records in Table B, and vice versa. For example, a student can enroll in many courses, and each course can have many students enrolled.

Which are different types of joins in SQL? Give examples.

There are four main types of joins in SQL:

Inner Join: Returns only the rows that match in both tables. Left Join: Returns all the rows from the left table and matching rows from the right table. If there are no matching rows in the right table, it returns NULL. Right Join: Returns all the rows from the right table and matching rows from the left table. If there are no matching

rows in the left table, it returns NULL. Full Outer Join: Returns all the rows from both tables, with NULLs for any missing matches. Examples:

```
Inner Join:
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;

Left Join:
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;

Right Join:
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
RIGHT JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;

Full Outer Join:
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

What is stored procedure? What are advantages of stored procedure?

A stored procedure is a set of SQL statements that are stored in the database and can be called by applications. Stored procedures are used to perform complex database operations that would be difficult to do with a single SQL statement. They are compiled once and can be executed many times, which makes them faster than individual SQL statements.

Advantages of stored procedures include:

Reusability: Stored procedures can be reused by different applications and users.

Security: Stored procedures can be granted permissions to certain users, making the database more secure.

Performance: Stored procedures are compiled once and executed many times, which makes them faster than individual SQL statements.

Modularity: Stored procedures can be modified independently of the application code, which makes maintenance easier.

Encapsulation: Stored procedures can encapsulate complex business logic, making it easier to maintain and update.

How stored function is different than stored procedure?

A stored function is a type of stored procedure that returns a single value. Stored functions are used to perform calculations or return values that can be used in other SQL statements. Stored procedures, on the other hand, can return multiple values or no values at all.

What is difference between CHAR, VARCHAR and TEXT? Does data type matter while creating index?

CHAR, VARCHAR, and TEXT are data types in SQL that are used to store character strings. The main differences between them are:

CHAR: Fixed-length character strings. The maximum length is specified when the table is created. If the string is shorter than the maximum length, it is padded with spaces.

VARCHAR: Variable-length character strings. The maximum length is specified when the table is created, but the actual length of the string can be shorter.

TEXT: Variable-length character strings. The maximum length is not specified when the table is created, but it is limited by the maximum size of the database.

The data type does matter when creating an index. For example, creating an index on a CHAR column is different than creating an index on a VARCHAR column. The index size will be larger for CHAR columns because the data is padded with spaces. VARCHAR columns are more efficient for indexing because the index size is smaller.

ACID properties of transactions ensure that database transactions are processed in a reliable and consistent manner. ACID stands for Atomicity, Consistency, Isolation, and Durability.

Atomicity: Transactions are atomic units of work, which means they are treated as a single, indivisible action. Transactions either succeed completely, or they fail and leave no trace of their partial completion. If a transaction is interrupted or terminated for any reason, the database must restore it to its pre-transaction state.

Consistency: Transactions must ensure that the data remains in a consistent state before and after the transaction. This means that all data must meet specific constraints and validation rules. If a transaction violates these rules, it must be rolled back.

Isolation: Transactions must be isolated from each other, so the actions of one transaction don't affect another. Isolation can be achieved by using locks, and it helps prevent data corruption, concurrency issues, and other errors.

Durability: Once a transaction is committed, the changes it made should be permanent and recoverable, even if the system fails. The changes must be written to permanent storage and made durable.

Transactions are used to maintain the integrity of the database by ensuring that the data remains consistent even in the event of failures. Transactions are useful when multiple users access the database simultaneously, as they ensure that data is accessed and modified in a safe and consistent manner.

Table-level locking and row-level locking are two different types of locking mechanisms in a database.

Table-level locking: In table-level locking, the database locks the entire table when a transaction modifies any row in that table. This type of locking is simple and easy to implement, but it can lead to performance issues

and concurrency problems when multiple transactions try to access the same table simultaneously.

Row-level locking: In row-level locking, the database locks only the specific row that a transaction is modifying, rather than the entire table. This type of locking is more granular and efficient than table-level locking, as it allows multiple transactions to modify different rows in the same table at the same time. However, it requires more overhead and can be more complex to implement.

Optimistic locking and pessimistic locking are two strategies for managing concurrent access to data.

Optimistic locking: In optimistic locking, multiple transactions can access and modify the same data simultaneously, but they must first check that no other transactions have made changes since they last accessed the data. If another transaction has modified the data, the current transaction is rolled back and must retry the operation with the new data. Pessimistic locking: In pessimistic locking, a transaction locks the data it wants to modify, preventing any other transactions from accessing or modifying it until the lock is released. This approach is more reliable and ensures data consistency, but it can lead to performance issues and concurrency problems when multiple transactions try to lock the same data simultaneously.

To find the third-highest salary from the EMP table, we can use subqueries as follows:

```
SELECT DISTINCT sal
FROM EMP e1
WHERE 3 = (
    SELECT COUNT(DISTINCT sal)
    FROM EMP e2
    WHERE e2.sal >= e1.sal
);
```

Alternatively, we can use the RANK() function to rank the salaries and then select the third-highest ranked salary:

```
SELECT sal
FROM (
    SELECT sal, RANK() OVER (ORDER BY sal DESC) as rank
    FROM EMP
) ranked
WHERE rank = 3;
```

To print the name of an employee and his manager, and also the manager's manager, we need to join the EMP table with itself multiple times. Here is an example query:

```
SELECT e.ename AS employee_name, m.ename AS manager_name, gm.ename AS
grand_manager_name
FROM emp e
JOIN emp m ON e.mgr = m.empno
JOIN emp gm ON m.mgr = gm.empno;
This query joins the EMP table with itself twice to get the names of the
```

```
employee's manager and the manager's manager. It uses the mgr column to join the
tables.
```

11. What is use of views? How to limit DML operations on views to the given criteria?

Views are virtual tables that are based on the result of a SELECT statement. They are used to simplify complex queries and to provide controlled access to the underlying tables. Views can be used to limit DML operations on views by using the WITH CHECK OPTION clause. The WITH CHECK OPTION clause prevents inserts, updates, and deletes that do not conform to the view's filter condition.

Here is an example of how to create a view with the WITH CHECK OPTION clause:

sql Copy code CREATE VIEW sales_staff AS SELECT * FROM employees WHERE department = 'Sales' WITH CHECK OPTION; This view only shows the sales staff, and the WITH CHECK OPTION clause prevents any updates that do not conform to the filter condition (i.e., updates that are not for the Sales department).

What is normalization? Explain 1-NF, 2-NF, 3-NF and BCNF in detail.

Normalization is a process of organizing the data in a database to minimize redundancy and dependency. There are several normal forms that define the level of normalization:

First Normal Form (1NF): A relation is in 1NF if it has no repeating groups or arrays, and if every attribute in the relation is atomic (i.e., indivisible).

Second Normal Form (2NF): A relation is in 2NF if it is in 1NF and every non-key attribute in the relation is fully dependent on the primary key.

Third Normal Form (3NF): A relation is in 3NF if it is in 2NF and there are no transitive dependencies between non-key attributes.

Boyce-Codd Normal Form (BCNF): A relation is in BCNF if it is in 3NF and every determinant is a candidate key.

What is NoSQL database? What are its advantages and limitations?

NoSQL databases are non-relational databases that use different data models from the traditional relational databases. They are designed to handle large volumes of unstructured or semi-structured data that cannot be easily organized into tables. Some of the advantages of NoSQL databases are:

Scalability: NoSQL databases are designed to scale horizontally, which means they can handle large volumes of data and traffic.

Flexibility: NoSQL databases can store data in different formats, such as JSON or XML, and can be used for a wide range of applications.

Performance: NoSQL databases are often faster than traditional relational databases for certain types of data and queries.

However, NoSQL databases also have some limitations, such as:

Lack of ACID transactions: NoSQL databases often sacrifice the ACID properties for scalability and performance, which can lead to data inconsistencies and integrity issues.

Limited query language: NoSQL databases often have a limited query language compared to traditional relational databases, which can make it harder to perform complex queries.

Explain BASE transactions and CAP theorem.

BASE stands for Basically Available, Soft State, Eventually Consistent. It is a concept used in NoSQL databases where the focus is on availability and partition tolerance over consistency. In BASE, the data is always available, and the system operates on a soft state which is allowed to change at any time. Eventual consistency is maintained, which means that the data will eventually become consistent across all nodes in the database.

CAP theorem, on the other hand, states that it is impossible for a distributed system to provide all three of the following guarantees: Consistency, Availability, and Partition Tolerance. Consistency means that all nodes in the database see the same data at the same time, Availability means that the system is always available for requests, and Partition Tolerance means that the system continues to function even when network partitions occur.

In practice, when designing a distributed system, one needs to choose two out of the three guarantees. For example, in the case of NoSQL databases, the focus is on Availability and Partition Tolerance, which leads to eventual consistency (as opposed to strong consistency). In contrast, traditional relational databases prioritize consistency over availability and partition tolerance.

Explain types of NoSQL databases. Where they can be used?

There are several types of NoSQL databases, each designed to handle specific types of data and use cases. The most common types of NoSQL databases are:

Key-Value Databases: These databases store data as key-value pairs. They are simple and can handle high volumes of data with ease. Examples include Redis and Riak.

Document Databases: These databases store data as JSON or XML documents, which can be nested and hierarchical. They are useful for handling unstructured data and can be used for content management systems or e-commerce applications. Examples include MongoDB and Couchbase.

Column-Family Databases: These databases store data in columns, rather than rows. They are useful for handling large amounts of data and are often used for big data and analytics. Examples include Cassandra and HBase.

Graph Databases: These databases are designed to handle complex relationships between data points. They are useful for social networking and recommendation engines. Examples include Neo4j and OrientDB.

NoSQL databases can be used in a wide range of applications, including social networking, e-commerce, content management, big data analytics, and more. They are particularly useful for handling large amounts of unstructured data that traditional relational databases may struggle to handle.

Write queries to perform CRUD operations on mongo database.

Here are some examples of CRUD operations in MongoDB:

```
Insert document:
db.myCollection.insert({name: "John", age: 30, email: "john@example.com"})
```

```
Find document:
db.myCollection.find({name: "John"})

Update document:
db.myCollection.update({name: "John"}, {$set: {age: 35}})

Delete document:
db.myCollection.remove({name: "John"})
```

These are some of the basic CRUD operations in MongoDB. There are many more advanced operations that can be performed, such as indexing and aggregation, but these are beyond the scope of this answer.