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 data redundancy and improve data integrity. Normalization ensures that data is stored in a way that minimizes duplication and inconsistencies, making it easier to maintain and update the database.

Normalization is needed to prevent data redundancy and inconsistencies that can occur when data is stored in a non-normalized database. A non-normalized database can result in data inconsistencies, making it difficult to maintain and update the database. Normalization ensures that the database is organized in a way that is consistent and easy to maintain.

There are several levels of normalization, including first normal form (1NF), second normal form (2NF), third normal form (3NF), and Boyce-Codd normal form (BCNF).

First Normal Form (1NF) First normal form requires that each column in a table contains only atomic values. That means that each column must contain only a single value, and each value must be indivisible. This eliminates repeating groups and ensures that data is organized in a way that is consistent and easy to manage.

Second Normal Form (2NF) Second normal form requires that all non-key attributes in a table are dependent on the primary key. This means that each column in a table must be directly related to the primary key, and not to any other non-key attributes in the table.

Third Normal Form (3NF) Third normal form requires that all non-key attributes in a table are not dependent on any other non-key attributes. This means that each column in a table must be directly related to the primary key, and not to any other non-key attributes in the table.

Boyce-Codd Normal Form (BCNF) Boyce-Codd normal form is a stricter form of third normal form that applies to tables with multiple candidate keys. It requires that every determinant (column that uniquely determines another column) is a candidate key.

Overall, normalization helps to ensure that data is organized in a consistent and easy to manage way, reducing data redundancy and improving data integrity

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

A stored procedure is a pre-compiled and stored SQL code that can be called by other programs or applications. It is a collection of SQL statements that are grouped together to perform a specific task or set of tasks. Stored procedures are stored in a database and can be called by multiple applications or users.

Advantages of stored procedures:

Reusability: Stored procedures can be called by multiple applications or users, which makes them reusable. This reduces the amount of code that needs to be written, tested, and maintained.

Performance: Stored procedures are pre-compiled and stored in the database, which makes them faster than ad-hoc SQL queries. They also reduce network traffic between the application and the database server, which improves performance.

Security: Stored procedures can be used to restrict access to sensitive data by allowing only authorized users to execute them. This helps to maintain the security of the database.

Maintenance: Stored procedures make it easier to maintain and update the database. Changes to the database schema or business logic can be made in a single stored procedure, which reduces the risk of introducing errors.

Scalability: Stored procedures can be used to perform complex operations on large datasets, which makes them more scalable than ad-hoc SQL queries.

Consistency: Stored procedures can be used to enforce business rules and data integrity constraints, which ensures consistency in the data.

In summary, stored procedures provide a number of advantages such as reusability, performance, security, maintenance, scalability, and consistency. They are a powerful tool that can be used to simplify the development and maintenance of database applications.

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

Stored functions and stored procedures are both database objects that contain SQL code and can be called by other programs or applications. However, there are some important differences between them:

Return value: The primary difference between stored functions and stored procedures is that a function returns a value, while a procedure does not necessarily return a value. A function always returns a single value of a specific data type, while a procedure may return multiple values or no values at all.

Usage: Stored procedures are typically used to perform a specific task or set of tasks, such as updating a table, inserting data, or retrieving data. Stored functions, on the other hand, are used to perform a calculation or transformation on input values and return a result.

Call syntax: Stored functions and stored procedures are called in different ways. To call a stored function, you use a SELECT statement that includes the function name and any input parameters. To call a stored procedure, you use an EXECUTE statement that includes the procedure name and any input parameters.

Transactions: Stored procedures can be used to start or commit transactions, which can be used to ensure data consistency and integrity. Stored functions, on the other hand, cannot start or commit transactions.

In summary, stored functions and stored procedures have different purposes and are used in different ways. Stored functions are used to perform calculations or transformations on input values and return a result, while stored procedures are used to perform a specific task or set of tasks. Stored functions always return a value, while stored procedures may or may not return a value.

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

In SQL, CHAR, VARCHAR, and TEXT are data types used to store character strings, but they have some differences:

CHAR: The CHAR data type is used to store fixed-length character strings. When a value is stored in a CHAR column, the column will always occupy the same amount of space in the database, regardless of the length of the value. For example, if you define a CHAR(10) column and store the value "Hello" in it, the column will occupy 10 bytes in the database.

VARCHAR: The VARCHAR data type is used to store variable-length character strings. When a value is stored in a VARCHAR column, the column will occupy only the amount of space needed to store the value, plus one or two bytes to store the length of the value. For example, if you define a VARCHAR(10) column and store the value "Hello" in it, the column will occupy only 5 bytes in the database.

TEXT: The TEXT data type is used to store large variable-length character strings. When a value is stored in a TEXT column, the column will occupy only the amount of space needed to store the value, plus two or four bytes to store the length of the value. TEXT columns can store up to 2^32-1 bytes of data.

Regarding the second question, yes, the data type does matter while creating an index. Indexes are used to improve the performance of database queries by allowing the database to find the required data more quickly. When you create an index on a column, the database creates a separate data structure that contains the indexed values and their locations in the table. When a query searches for data using the indexed column, the database can use the index to quickly locate the required data.

However, creating an index on a large data type such as TEXT can be less efficient than creating an index on a smaller data type such as VARCHAR or CHAR. This is because the index data structure will also need to store the large TEXT values, which can take up a lot of space and slow down index creation and querying. So, it's generally recommended to use a smaller data type for indexed columns, if possible.

Explain ACID properties of transaction. What is use of transaction?’

ACID is an acronym for Atomicity, Consistency, Isolation, and Durability. These are the properties that ensure reliability and consistency of database transactions.

Atomicity: This property ensures that a transaction is treated as a single, indivisible unit of work. Either all the operations of the transaction are completed successfully or none of them are. If a transaction is interrupted or fails for any reason, all changes made by the transaction are rolled back to their previous state, so that the database remains in a consistent state.

Consistency: This property ensures that a transaction brings the database from one consistent state to another consistent state. In other words, the database remains in a valid state before and after the transaction. The database constraints and rules are enforced during the transaction to maintain the integrity of the data.

Isolation: This property ensures that each transaction is executed independently of any other transactions that are concurrently executing on the database. Transactions are isolated from each other to prevent interference, so that the changes made by one transaction are not visible to other transactions until the first transaction is completed.

Durability: This property ensures that the changes made by a transaction are permanent and will survive any subsequent failures, such as power failures, system crashes, or other types of errors. Once a transaction has been committed, its changes are stored permanently in the database.

The use of transactions is essential in database systems where multiple users or applications may be accessing the same data simultaneously. Transactions help to ensure that changes to the data are consistent and reliable, even in the presence of concurrent access and failure conditions. Transactions also help to maintain the integrity of the database by enforcing the ACID properties, which prevent data corruption and inconsistencies.

What is table level locking and row level locking? What is optimistic locking and pessimistic  
locking?

In a database system, locking is a mechanism used to control concurrent access to data. Locking can be done at the table level or the row level, depending on the requirements of the application.

Table-level locking: In table-level locking, the entire table is locked when a user or transaction accesses the table. This means that no other user or transaction can access the table until the lock is released. Table-level locking is simple to implement, but it can lead to concurrency issues and decreased performance in systems with high transaction volumes.

Row-level locking: In row-level locking, only the specific row that is being accessed or modified is locked, while other rows in the table remain available for access by other users or transactions. Row-level locking provides greater concurrency and performance, but it is more complex to implement.

Optimistic locking and pessimistic locking are two strategies used for row-level locking:

Optimistic locking: In optimistic locking, the database assumes that conflicts between concurrent transactions are rare, so it does not lock the data when it is being read. Instead, the database checks for conflicts when the data is being written. If a conflict is detected, the transaction is rolled back and the user must retry the transaction. Optimistic locking is useful when conflicts are infrequent, but it can result in a high rate of transaction rollbacks.

Pessimistic locking: In pessimistic locking, the database locks the data when it is being read, so that no other transactions can modify the data until the lock is released. Pessimistic locking is useful when conflicts are expected to be frequent, but it can result in lower concurrency and performance.

In summary, table-level locking locks the entire table, while row-level locking locks only the specific row being accessed. Optimistic locking assumes that conflicts are rare and checks for conflicts when data is being written, while pessimistic locking assumes that conflicts are common and locks data when it is being read.

Write a query to find third highest salary of employee (on EMP table) using sub-queries and  
without using sub-queries.

SELECT DISTINCT salary

FROM EMP

ORDER BY salary DESC

LIMIT 2,1;

SELECT MAX(salary)

FROM EMP

WHERE salary < (

SELECT MAX(salary)

FROM EMP

WHERE salary < (

SELECT MAX(salary)

FROM EMP

)

);

Write a query to print name of employee and his manager. Also print manager’s manager.

To print the name of the employee, his manager, and the manager's manager, we need to perform a self-join on the EMP table. The following SQL query will accomplish this:

SELECT

e.ename as employee\_name,

m.ename as manager\_name,

mm.ename as manager\_manager\_name

FROM

EMP e

JOIN EMP m ON e.mgr = m.empno

JOIN EMP mm ON m.mgr = mm.empno;

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

A view is a virtual table that is based on the result of an SQL SELECT statement. Views are used to simplify the complexity of the underlying schema, to restrict access to sensitive data, or to provide an alternative way of presenting data.

Views can be used to limit DML operations by defining a set of rules or criteria that restrict the type of operations that can be performed on the view. This can be accomplished using the WITH CHECK OPTION clause in the CREATE VIEW statement.

The WITH CHECK OPTION clause ensures that any updates or inserts made to the view meet the specified criteria. For example, consider the following CREATE VIEW statement:

CREATE VIEW SalesView AS

SELECT \* FROM SalesTable

WHERE year = 2022;

WITH CHECK OPTION;

In this example, the view SalesView is created based on the SalesTable, but it only includes data for the year 2022. The WITH CHECK OPTION clause ensures that any updates or inserts made to the view also meet this criterion. If a user attempts to update or insert a record with a year other than 2022, the operation will fail and an error message will be returned.

By using the WITH CHECK OPTION clause, DML operations on the view are limited to the given criteria, and users are prevented from modifying or inserting data that does not meet these criteria. This can help to ensure data integrity and prevent unauthorized access to sensitive data.

窗体顶端

窗体底端

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

A NoSQL database is a type of database that stores and retrieves data without using the traditional relational database model. NoSQL stands for "not only SQL," indicating that it may use non-relational models as well as SQL. NoSQL databases are designed to handle large volumes of unstructured, semi-structured, and structured data.

Advantages of NoSQL databases:

Scalability: NoSQL databases are designed to scale horizontally, meaning that they can handle large amounts of data by adding more nodes to the database cluster. This makes them well-suited for big data applications.

Flexibility: NoSQL databases are schema-less, meaning that they can store data in any format without needing a predefined schema. This makes it easy to add new data types or modify existing ones without the need for schema changes.

Performance: NoSQL databases are optimized for performance and can handle large volumes of data quickly and efficiently. They can perform well in distributed environments and can handle high read and write throughput.

Cost-effective: NoSQL databases can be more cost-effective than traditional relational databases since they do not require expensive hardware and software licenses.

Limitations of NoSQL databases:

Lack of standardization: NoSQL databases do not have a standard language or protocol, making it difficult to move data between different NoSQL databases or integrate with other systems.

Limited query capabilities: NoSQL databases are optimized for specific types of queries, and they may not support complex queries as well as traditional relational databases.

Limited transactional support: NoSQL databases may not provide full support for ACID transactions, which can be a limitation for applications that require strong consistency guarantees.

Limited tool support: NoSQL databases may have limited tool support for monitoring, management, and administration compared to traditional relational databases.

In summary, NoSQL databases offer scalability, flexibility, and performance advantages over traditional relational databases, but they may also have limitations around standardization, query capabilities, transactional support, and tool support.

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

There are four main types of NoSQL databases:

Document-based databases: Document-based databases store data in semi-structured documents, such as JSON or XML, rather than in structured tables. Each document contains key-value pairs and can have a flexible schema, which allows for easy scalability and data modeling. Examples of document-based databases include MongoDB and Couchbase. They can be used for content management, e-commerce, and mobile applications.

Key-value databases: Key-value databases store data as a collection of key-value pairs, where the key is used to uniquely identify the value. These databases are optimized for high throughput and low latency and are commonly used for caching and session management. Examples of key-value databases include Redis and Riak.

Column-family databases: Column-family databases store data in column families, which are groups of related columns. They are designed for handling large volumes of data and are commonly used for data warehousing and analytics. Examples of column-family databases include Apache Cassandra and Apache HBase.

Graph databases: Graph databases store data in nodes and edges, which represent entities and their relationships. They are optimized for handling complex relationships and are commonly used for social networks, recommendation engines, and fraud detection. Examples of graph databases include Neo4j and OrientDB.

NoSQL databases can be used in a variety of applications and industries, including:

Web applications: NoSQL databases can handle high volumes of data and provide scalability, making them well-suited for web applications.

Big data: NoSQL databases can handle large volumes of unstructured and semi-structured data, making them ideal for big data applications.

IoT: NoSQL databases can handle the large volumes of data generated by IoT devices and can provide real-time analytics and insights.

E-commerce: NoSQL databases can handle large volumes of customer data and can provide real-time recommendations and personalization.

In summary, the type of NoSQL database to use will depend on the application requirements and the nature of the data being stored. Different types of NoSQL databases offer unique advantages and can be used in a variety of applications and industries.