**1. What is index; types of indices; pros and cons** A database index is a data structure that provides a quick lookup of data in a column or columns of a table. It enhances the speed of operations accessing data from a database table at the cost of additional writes and memory to maintain the index data structure.

There are different types of indexes that can be created for different purposes:

* **Unique and Non-Unique Index:**

Unique indexes are indexes that help maintain data integrity by ensuring that no two rows of data in a table have identical key values. Once a unique index has been defined for a table, uniqueness is enforced whenever keys are added or changed within the index.

Non-unique indexes, on the other hand, are not used to enforce constraints on the tables with which they are associated. Instead, non-unique indexes are used solely to improve query performance by maintaining a sorted order of data values that are used frequently.

* **Clustered and Non-Clustered Index:**

Clustered indexes are indexes whose order of the rows in the database corresponds to the order of the rows in the index. This is why only one clustered index can exist in a given table, whereas, multiple non-clustered indexes can exist in the table.

The only difference between clustered and non-clustered indexes is that the database manager attempts to keep the data in the database in the same order as the corresponding keys appear in the clustered index.

Clustering indexes can improve the performance of most query operations because they provide a linear-access path to data stored in the database.

**2. What's the difference between Primary key and Unique constraint?**

Primary key differs from the unique constraint in that; you can create multiple unique constraints in a table, with the ability to define only one SQL primary key each table. Another difference is that the unique constraint allows for one NULL value, but the primary key does not allow NULL values.

**3. Tell me about check constraint**

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

**4. Difference between temp table and table variable**

* Temporary Tables are physically created in the tempdb database. These tables act as the normal table and also can have constraints, index like normal tables.
* Table Variable acts like a variable and exists for a particular batch of query execution. It gets dropped once it comes out of batch. It is created in the memory database but may be pushed out to tempdb.

**5. Difference between WHERE and HAVING**

1) Both are used as filters, but having apply only to groups as a whole, and only filters on aggregation functions; where applys to individual rows;

2)WHERE goes before aggregations, but HAVING filters after the aggregations:

FROM/JOIN -> WHERE -> GROUP BY -> HAVING -> SELECT -> DISTINCT -> ORDER BY

3)WHERE can used with SELECT UPDATE OR DELETE, but HAVING will only be used in SELECT

**6. Difference between RANK() and DenseRank() — value gap**

RANK() gives you the ranking within your ordered partition. Ties are assigned the same rank, with the next ranking(s) skipped. So, if you have 3 items at rank 2, the next rank listed would be ranked 5.

DENSE\_RANK() again gives you the ranking within your ordered partition, but the ranks are consecutive. No ranks are skipped if there are ranks with multiple items.

**7. COUNT(\*) vs. COUNT(colName)**

* COUNT(\*) counts all rows
* COUNT(colName) counts non-NULLs only

**8. What's the difference between left join and inner join? JOIN and Subquery, which one has a better performance, why?**

INNER JOIN: returns rows when there is a match in both tables.

LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.

JOIN. The advantage of a join includes that it executes faster. The retrieval time of the query using joins almost always will be faster than that of a subquery. By using joins, you can maximize the calculation burden on the database i.e., instead of multiple queries using one join query.

**9. What is correlated subquery?**

In a SQL query, a correlated subquery is a subquery (a query nested inside another query) that uses values from the outer query. Because the subquery may be evaluated once for each row processed by the outer query, it can be slow.

**10. What is a CTE, why do we need CTE?**

CTE stands for common table expression. In SQL, we will use sub-queries to join the records or filter the records from a sub-query. Whenever we refer the same data or join the same set of records using a sub-query, the code maintainability will be difficult. A CTE makes improved readability and maintenance easier.

 Advantages of CTE:

* CTE improves the code readability.
* CTE provides recursive programming.
* CTE makes code maintainability easier.
* Though it provides similar functionality as a view, it will not store the definition in metadata.

**11. What does SQL Profiler do?**

SQL Server Profiler is an interface to create and manage traces and analyze and replay trace results. Events are saved in a trace file that can later be analyzed or used to replay a specific series of steps when diagnosing a problem.

**12. \*\*\*What is SQL injection, how to avoid SQL injection?**

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

Most instances of SQL injection can be prevented by using parameterized queries (also known as prepared statements) instead of string concatenation within the query.

**13. Difference between SP and user defined function? When to use SP when to use function?**

* Procedure can return zero or n values whereas function can return one value which is mandatory.
* Procedures can have input/output parameters for it whereas functions can have only input parameters.
* Procedure allows select as well as DML statement in it whereas function allows only select statement in it.
* Functions can be called from procedure whereas procedures cannot be called from function.
* Exception can be handled by try-catch block in a procedure whereas try-catch block cannot be used in a function.
* We can go for transaction management in procedure whereas we can't go in function.
* Procedures can not be utilized in a select statement whereas function can be embedded in a select statement.
* UDF can be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section where as Stored procedures cannot be.
* UDFs that return tables can be treated as another rowset. This can be used in JOINs with other tables.
* Inline UDF's can be though of as views that take parameters and can be used in JOINs and other Rowset operations.

**14. Criteria of Union and Union all? Difference between UNION and UNION ALL**

* UNION: only keeps *unique* records
* UNION ALL: keeps all records, including *duplicates*

**15. Steps you take to improve SQL Queries**

* Indexing: Ensure proper indexing for quick access to the database.
* Select query: Specify the columns in SELECT query instead of SELECT\* to avoid extra fetching load on the database.
* Running queries: Loops in query structure slows the sequence. Thus, avoid them.
* Matching records: Use EXITS() for matching if the record exists.
* Subqueries: Avoid correlated sub queries as it searches row by row, impacting the speed of SQL query processing.
* Wildcards: Use wildcards (e.g. %xx%) wisely as they search the entire database for matching results.
* Operators: Avoid using function at RHS of the operator.
* Fetching data: Always fetch limited data.
* Loading: Use a temporary table to handle bulk data.
* Selecting Rows: Use the clause WHERE instead of HAVING for primary filters.

**16. concurrency problem in transaction**

Concurrency is a situation where two users are trying to access the same information and while they are accessing the same information we do not want any kind of inconsistency result or abnormal behavior.

The following four concurrency problems may appear:

* Lost update
* Dirty reads
* Nonrepeatable reads
* Phantoms

The *lost update* concurrency problem occurs when no isolation is provided to a transaction from other transactions. This means that several transactions can read the same data and modify it. The changes to the data by all transactions, except those by the last transaction, are lost.

The dirty reads refer to reading data that is uncommitted. This can become problematic if the uncommitted transaction fails or for some other reason is rolled back.

The *nonrepeatable read* concurrency problem occurs when one process reads data several times, and another process changes the same data between two read operations of the first process. Therefore, the values read by both read operations of the first process are different.

The *phantom* concurrency problem is similar to the nonrepeatable read concurrency problem, because two subsequent read operations can display different values, but in this case, the reason for this behavior lies in the different number of rows being read the first time and the second time. (Additional rows, called phantoms, are inserted by other transactions.)

**17. what is deadlock, how to prevent**

To follow the ACID properties, SQL Server uses locking mechanisms, constraints and write-ahead logging. Various lock types include: exclusive lock(X), shared lock(S), update lock (U), intent lock (I), schema lock (SCH) and bulk update lock (BU). These locks can be acquired on the key, table, row, page and database level. Deadlocks occur when two processes want to access resources that are mutually being locked by each other. This locked situation can continue forever if nobody stops it.

Ways to avoid and minimize SQL Server deadlocks:

* Try to keep transactions short; this will avoid holding locks in a transaction for a long period of time.
* Access objects in a similar logical manner in multiple transactions.
* Create a covering index to reduce the possibility of a deadlock.
* Create Index to match the foreign key columns. This way, you can eliminate deadlocks due to cascading referential integrity.
* Set deadlock priorities using the SET DEADLOCK\_PRIORITY session variable. If you set the deadlock priority, SQL Server kills the session with the lowest deadlock priority.
* Utilize the error handling using the try-catch blocks. You can trap the deadlock error and rerun the transaction in the event of a deadlock victim.
* Change the isolation level to the READ COMMITTED SNAPSHOT ISOLATION or SNAPSHOT ISOLATION. This changes the SQL Server locking mechanism. Although, you should be careful in changing the isolation level, as it might impact other queries negatively.

Benefits of Data Normalization:

As data becomes more and more valuable to any type of business, data normalization is more than just reorganizing the data in a database. Here are some of its major benefits:

* Reduces redundant data
* Provides data consistency within the database
* More flexible database design
* Higher database security
* Better and quicker execution
* Greater overall database organization

**18. what is normalization, 1NF - BCNF, benefits using normalization**

Database normalization is the process of restructuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity.

First Normal Form (1NF):

The first normal form requires that a table satisfies the following conditions:

1. Rows are not ordered
2. Columns are not ordered
3. There is duplicated data
4. Row-and-column intersections always have a unique value
5. All columns are “regular” with no hidden values

Second Normal Form (2NF):

An entity is in a second normal form if all of its attributes depend on the whole primary key. So this means that the values in the different columns have a dependency on the other columns.

1. The table must be already in 1 NF and all non-key columns of the tables must depend on the PRIMARY KEY
2. The partial dependencies are removed and placed in a separate table

Third Normal Form (3NF):

The third normal form states that you should eliminate fields in a table that do not depend on the key.

1. A Table is already in 2 NF
2. Non-Primary key columns shouldn’t depend on the other non-Primary key columns
3. There is no transitive functional dependency

Boyce-Codd Normal Form (BCNF):

Boyce-Codd Normal Form (BCNF) is one of the forms of database normalization. A database table is in BCNF if and only if there are no non-trivial functional dependencies of attributes on anything other than a superset of a candidate key.

**19. what are the system defined databases?**

Databases deployed on a SQL Server instance can either be System Databases or User Databases. System Databases come installed with the instance.

**20. composite key**

A set of columns used to uniquely identify the rows in a table.

**21. candidate key**

Candidate key is a single key or a group of multiple keys that uniquely identify rows in a table. A Candidate key is a subset of Super keys and is devoid of any unnecessary attributes that are not important for uniquely identifying tuples.

**22. DDL vs. DML**

* **DDL:** DDL is that part of SQL that defines the data structure of the database in the initial stage when the database is about to be created. It is mainly used to create and restructure database objects. Commands in DDL are:
  + Create table
  + Alter table
  + Drop table
* **DML:** DML is used to manipulate already existing data in a database, i.e., it helps users to retrieve and manipulate data. It is used to perform operations such as inserting data into the database through the insert command, updating data with the update command, and deleting data from the database through the delete command.

**23. ACID property**

The full form of ACID is atomicity, consistency, isolation, and durability. ACID properties are used to check the reliability of transactions.

* Atomicity refers to completed or failed transactions, where a transaction refers to a single logical operation on data. This implies that if any aspect of a transaction fails, the whole transaction fails and the database state remains unchanged.
* Consistency means that the data meets all validity guidelines. The transaction never leaves the database without finishing its state.
* Concurrency management is the primary objective of isolation.
* Durability ensures that once a transaction is committed, it will occur regardless of what happens in between such as a power outage, fire, or some other kind of disturbance.

**24. table scan vs. index scan**

Table scan means iterate over all table rows.

Index scan means iterate over all index items, when item index meets search condition, table row is retrived through index.

Ussualy index scan is less expensive than a table scan because index is more flat than a table.

**25. Difference between Union and JOIN**

JOIN in SQL is used to combine data from many tables based on a matched condition between them. The data combined using JOIN statement results into new columns.

UNION in SQL is used to combine the result-set of two or more SELECT statements. The data combined using UNION statement is into results into new distinct rows.