Abubakr Mamajonov 52493

1. **How much availability is enough?**

   The level of availability needed in SQL Server depends on various factors such as the specific requirements of the application or system, the criticality of the data, the expected downtime tolerance, and the budget allocated for implementing high availability solutions. There is no one-size-fits-all answer to this question, as different organizations have different needs and priorities.

   SQL Server provides several options for achieving high availability, each with different levels of downtime and data loss potential. Here are some common availability options:

   Backup and Restore: This is the most basic form of data protection. Regular backups are taken, and in the event of a failure, the database can be restored to a previous point in time. However, this method can result in some data loss and downtime.

   Database Mirroring: This feature allows for real-time data replication from one server to another. In case of a failure, the standby server can take over quickly. Database mirroring provides relatively low downtime and can be suitable for certain applications.

   Always On Failover Cluster Instances (FCI): This method involves a cluster of servers where one server acts as the primary and another as a secondary. If the primary server fails, the secondary takes over. Failovers are typically quick, resulting in minimal downtime.

   Always On Availability Groups (AG): This feature builds upon database mirroring and provides a more advanced and flexible solution. It allows for multiple secondary replicas that can be used for read-only operations and automatic failover. Availability Groups can offer higher availability and scalability compared to FCI.

   SQL Server Always on Availability Groups with Failover Cluster Instances: This combines the benefits of Always on AG and FCI, providing both high availability and disaster recovery capabilities.

   The appropriate level of availability depends on the organization's requirements. For some applications, minimal downtime and data loss are critical, and they might opt for synchronous replication and automatic failover. Others might tolerate a higher downtime window and choose a less complex solution.

   It's important to conduct a thorough analysis of the business needs, infrastructure, and budgetary constraints to determine the level of availability that meets the specific requirements. Additionally, it is recommended to consult with database administrators or experts who can assess the unique needs of the organization and provide guidance on the best approach for achieving the desired availability.

2. **Describe the Installing & Upgrading SQL Server.**

1. Introduction SQL Server supports two types of installation: Standalone and Cluster-based. This guide will walk you through the steps to install SQL Server and provide the necessary information and prerequisites for a successful installation.
2. Pre-installation Checks Before starting the installation process, perform the following checks: Ensure you have RDP (Remote Desktop Protocol) access to the server. Check the operating system (OS) bit, IP address, and domain of the server. Verify that your account is in the admin group to run the setup.exe file. Determine the software location where the SQL Server installation files are stored.
3. Requirements Gather the following information and requirements before starting the installation: Determine the version, edition, service pack (SP), and any hotfixes required for SQL Server. Identify the service accounts for the database engine, agent, SSAS (SQL Server Analysis Services), SSIS (SQL Server Integration Services), and SSRS (SQL Server Reporting Services), if applicable. Specify the named instance name, if any. Decide on the location for the SQL Server binaries, system databases, and user databases. Choose the authentication mode (Windows Authentication or Mixed Mode). Define the collation setting for the SQL Server instance. Prepare a list of features you want to install with SQL Server.
4. Pre-requisites Based on the version of SQL Server you are installing, ensure the following pre-requisites are met: For SQL Server 2005: Setup support files. .NET Framework 2.0. SQL Server Native Client. For SQL Server 2008 & 2008 R2: Setup support files. .NET Framework 3.5 SP1. SQL Server Native Client. Windows Installer 4.5 or a later version. For SQL Server 2012 & 2014: Setup support files. .NET Framework 4.0. SQL Server Native Client. Windows Installer 4.5 or a later version. Windows PowerShell 2.0.
5. Installation Process Follow these steps to install SQL Server: Locate the SQL Server installation files and run the setup.exe file. Choose the installation type (Standalone or Cluster-based) and proceed. Accept the license terms and choose the SQL Server edition, version, and features you want to install. Specify the instance name, if applicable. Configure the service accounts for the database engine, agent, and other components. Define the installation location for the SQL Server binaries, system databases, and user databases. Choose the authentication mode and set the collation. Complete the installation process by following the prompts.
6. Configuring TCP/IP Port Number If you want to change the default TCP port for SQL Server, follow these steps: Open SQL Server Configuration Manager. Expand SQL Server Network Configuration and click on Protocols for MSSQLSERVER (or the instance name). Right-click on TCP/IP and select Properties. Configure the specific IP address or change the port for all IP addresses under the IP Addresses tab. Restart the SQL Server service for the changes to take effect.
7. Opening the SQL Server Instance Port Using Windows Firewall To allow connections to the SQL Server instance through Windows Firewall, follow these steps: Open Windows Firewall settings. Go to Advanced Settings. Select Inbound Rules and click on New Rule. Choose the Port rule type and specify the TCP port (default is 1433). Allow the connection and provide a descriptive name for the rule. Finish the configuration.

3. Describe the Monitoring and Tuning the Data Cache.

Monitoring and tuning the data cache in SQL Server is an important aspect of optimizing database performance. The data cache, also known as the buffer pool, is a memory area used to

store frequently accessed data pages from the database. By monitoring and tuning the data cache, you can improve query response times and overall system performance. Here are the key steps involved in monitoring and tuning the data cache:

Monitoring Data Cache Usage:

Utilization: Monitor the usage of the data cache to determine how much of the allocated memory is being used.
Page Reads/Writes: Keep an eye on the number of pages read from or written to the disk. Frequent disk I/O can indicate a need for tuning the data cache.
Page Life Expectancy (PLE): PLE represents the average time a data page remains in the cache before being removed. A low PLE value might indicate memory pressure and the need for tuning.
Analyzing Performance Metrics:

Identify frequently accessed tables and indexes: Monitor query performance and identify tables and indexes that are heavily used. These should be prioritized for caching.
Identify poorly performing queries: Analyze query execution plans, identify slow queries, and optimize them to reduce data cache pressure.
Adjusting Memory Allocation:

Set appropriate maximum memory: Configure the maximum amount of memory that SQL Server can utilize, considering the available physical memory and other applications running on the server.
Adjust the data cache size: Based on your analysis, adjust the "max server memory" setting to allocate an appropriate amount of memory to the data cache.
Index Optimization:

Ensure proper indexing: Review table indexes and ensure they are properly designed and maintained. Well-designed indexes can reduce the amount of data read from disk and improve cache utilization.
Identify missing or unused indexes: Analyze query plans and identify missing or redundant indexes that can impact data cache efficiency. Create necessary indexes and remove unused ones.
Query and Database Optimization:

Optimize queries: Review and optimize poorly performing queries by rewriting or reorganizing them to reduce data access and improve cache utilization.
Normalize or denormalize database schema: Evaluate the database schema design and make appropriate changes to normalize or denormalize tables, depending on the workload and query patterns.
Regular Maintenance:

Regularly update statistics: Outdated statistics can result in poor query plans. Maintain up-to-date statistics to ensure accurate query optimization.

Regularly defragment indexes: Fragmented indexes can impact cache utilization. Regularly defragment indexes to improve performance.
Monitor and Iterate:

Continuously monitor and analyze the performance metrics related to the data cache.
Iterate on the tuning process by implementing changes, monitoring their impact, and adjusting accordingly.
By following these steps, you can effectively monitor and tune the data cache in SQL Server to optimize performance and enhance the overall efficiency of your database system.

4. Define Performance.

In the context of SQL (Structured Query Language), performance refers to the efficiency and speed at which database operations and queries are executed. It involves optimizing the execution of SQL statements to minimize response time, reduce resource consumption, and improve overall database system performance.

Performance in SQL can be evaluated based on various factors, including:

Query Execution Time: The time taken to execute SQL queries is a crucial performance metric. Faster query execution time means that results are obtained more quickly, improving the responsiveness of the application or system.

Indexing: Proper indexing of database tables is essential for efficient query execution. Well-designed and properly utilized indexes can significantly enhance performance by allowing the database engine to locate and retrieve data more efficiently.

Query Optimization: Optimizing SQL queries involves analyzing query execution plans, identifying inefficiencies, and making appropriate modifications to improve performance. This may include rewriting queries, rearranging join orders, or utilizing appropriate join algorithms.

Data Volume and Access Patterns: The size of the database and the frequency of data access can impact performance. Large datasets may require specific optimizations such as partitioning or data archiving strategies to maintain optimal query performance.

Database Design: The design of the database schema, including table structure, relationships, and normalization, can affect performance. A well-designed database can minimize unnecessary joins, data duplication, and improve data retrieval efficiency.

Resource Utilization: Efficient utilization of system resources like CPU, memory, and disk I/O is crucial for SQL performance. Monitoring and optimizing resource usage can prevent bottlenecks and ensure optimal database performance.

Query Caching: SQL query caching can improve performance by storing and reusing the results of previously executed queries. Caching eliminates the need to recompute or reprocess the same query repeatedly.

Concurrency and Locking: Managing concurrent access to the database is vital for performance. Effective concurrency control mechanisms, such as appropriate locking strategies, can minimize contention and maximize throughput.

By addressing these factors, monitoring system performance, and continually optimizing SQL queries and database operations, it is possible to achieve optimal performance in SQL-based systems, leading to improved response times, efficient resource utilization, and better overall user experience.

5. **Grant and Remove permissions.**

Granting Permissions: Granting permissions allows users to perform specific actions or operations on database objects. In SQL Server, you can use the GRANT statement to grant permissions to users or roles. For example, to grant read-only access to the surname field in the "test" table, you can use the following command:
GRANT SELECT ON dbo.test(surname) TO [52493];

Removing Permissions: If you need to revoke previously granted permissions, you can use the REVOKE statement. For example, to revoke the read-only access to the surname field, you can use the following command:
REVOKE SELECT ON dbo.test(surname) FROM [52493];

```
-- Creating the database
CREATE DATABASE [52493];
GO

-- Using the newly created database
USE [52493];
GO

-- Creating the 'test' table
CREATE TABLE test (
  Id INT IDENTITY PRIMARY KEY,
  name VARCHAR(40),
  surname VARCHAR(40)
);
GO

-- Creating a new user without a password
CREATE USER [52493] WITHOUT LOGIN;
GO
```

```
-- Grant read-only access to the 'surname' field
GRANT SELECT (surname) ON test TO [52493];
GO
```

6. **Write a procedure for deleting tables in the database.**

```
CREATE PROCEDURE DeleteTables
AS
BEGIN
  -- Create a temporary table to store the list of tables
  CREATE TABLE #TableList (
    TableId INT IDENTITY(1, 1),
    TableName VARCHAR(100)
  );

  -- Insert the table names into the temporary table
  INSERT INTO #TableList (TableName)
  SELECT TABLE_NAME
  FROM INFORMATION_SCHEMA.TABLES
  WHERE TABLE_TYPE = 'BASE TABLE';

  -- Display the list of tables to the user
  SELECT TableId, TableName
  FROM #TableList;

  -- Prompt the user to select tables for deletion
  DECLARE @Decision INT;
  SET @Decision = 0;

  WHILE @Decision <> -1
  BEGIN
    -- Prompt the user for the decision
    SET @Decision = 0;
    PRINT 'Enter the TableId of the table to delete (-1 to exit):';
    SET @Decision = CONVERT(INT, INPUT());

    -- Delete the selected table if the decision is valid
    IF @Decision > 0
    BEGIN
      DECLARE @TableName VARCHAR(100);

      -- Get the table name based on the selected TableId
```

```
        SELECT @TableName = TableName
        FROM #TableList
        WHERE TableId = @Decision;

        -- Delete the table if it exists
        IF EXISTS (SELECT 1 FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME
= @TableName)
        BEGIN
           EXEC('DROP TABLE ' + QUOTENAME(@TableName));
           PRINT 'Table ' + QUOTENAME(@TableName) + ' deleted successfully.';
        END
        ELSE
        BEGIN
           PRINT 'Table ' + QUOTENAME(@TableName) + ' does not exist.';
        END
     END
  END

  -- Clean up the temporary table
  DROP TABLE #TableList;
END
```

To use the stored procedure, you can execute the following command:

```
EXEC DeleteTables;
```

The stored procedure will display a list of tables with their corresponding TableId. You can then enter the TableId of the table you want to delete. Enter -1 to exit the deletion process.