SQL Server Interview Questions and Answers

1. What are the differences between Log Shipping and Database Mirroring?

Feature	Log Shipping	Database Mirroring
Architecture	One primary database with one or more secondary databases	One principal database and one mirror database
Synchronization	Asynchronous	Synchronous or Asynchronous (in high-performance mode)
Automatic Failover	No	Yes (Only in High-Safety Mode with Witness)
Use Case	DR solution for read-only secondary servers	High availability for automatic failover
Latency	Higher (Depends on copy and restore frequency)	Lower latency (sync mode)

---

2. What are the best practices for SQL Server patching?

- Always test patches in a non-production environment before applying them to production.
- Take full database and transaction log backups before patching.
- Schedule patching during non-peak hours.
- Use a rolling update approach in an Always On setup to minimize downtime.
- Monitor system performance post-patching.
- Keep documentation for rollback in case of failure.

---

3. What are the system databases in SQL Server and their functions?

- **master** – Stores system-level information such as logins, linked servers, and configurations.
- **model** – Template for new databases.
- **msdb** – Used for SQL Server Agent jobs, backups, and alerts.
- **tempdb** – Stores temporary objects, tables, and query processing data.
- **resource** – A hidden database storing system objects (readonly).

---

4. What are the different types of backups in SQL Server?

- **Full Backup** – Backs up the entire database.
- **Differential Backup** – Captures changes since the last full backup.

- **Transaction Log Backup** – Backs up the transaction log to enable point-in-time recovery.
- **Copy-Only Backup** – A full backup that does not impact the differential base.
- **File/Filegroup Backup** – Backs up specific database files or filegroups.
- **Mirror Backup** – Creates an exact copy of the backup file.

---

5. Explain the different isolation levels in SQL Server.

Isolation Level	Dirty Reads	Non-Repeatable Reads	Phantom Reads
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializable	No	No	No
Snapshot	No	No	No

---

6. How does Always On Availability Groups work in SQL Server?

- Always On Availability Groups provide high availability and disaster recovery by allowing multiple replicas.
- It consists of a **primary replica** and up to eight **secondary replicas**.
- The **synchronous-commit mode** ensures zero data loss, while **asynchronous-commit mode** improves performance at the cost of potential data loss.
- It supports automatic failover, read-only secondaries, and backup offloading.

---

7. What are the new features introduced in SQL Server 2022?

- **Azure Synapse Link for SQL Server**: Integrates on-premise data with Azure Synapse.
- **Contained Availability Groups**: Simplifies failover with database-level configuration.
- **Intelligent Query Processing Enhancements**: Optimized performance for parameter-sensitive plans.
- **SQL Ledger**: Blockchain-like ledger table for tamper-proof data integrity.
- **Data Virtualization**: Access external data without replication.

---

8. What are the different types of locks in SQL Server?

- **Shared (S)** – Allows multiple read operations, but no write.
- **Exclusive (X)** – Blocks all other access.
- **Update (U)** – Prevents deadlocks between read/update transactions.

- **Intent (IS, IX, IU)** – Indicates a lock intention at a higher level.
- **Schema Modification (Sch-M)** – Used for DDL changes.
- **Bulk Update (BU)** – Used during bulk inserts.

---

9. What is CTP and MAXDOP in SQL Server?

Answer:

- **CTP (Cumulative Trace Flag)**: A set of trace flags used to enable or disable specific SQL Server features for testing or debugging purposes.
- **MAXDOP (Maximum Degree of Parallelism)**: A setting that controls the number of CPU cores SQL Server uses for parallel query execution, helping optimize performance by limiting or allowing parallelism based on available processors.

---

10. How does SQL Server allocate RAM?

- SQL Server dynamically manages memory using **Min Server Memory** and **Max Server Memory** settings.
- The **Buffer Pool** stores cached data pages.
- The **Plan Cache** holds execution plans to optimize queries.
- The **Worker Threads** use memory for query processing.
- **Resource Governor** helps allocate memory between workloads.

---

11. What are the different database states in SQL Server?

- **ONLINE** – Database is available.
- **OFFLINE** – Database is manually taken offline.
- **RESTORING** – Restore operation is in progress.
- **RECOVERING** – Automatic recovery is in progress.
- **RECOVERY_PENDING** – Recovery is required but unable to start.
- **SUSPECT** – Database is damaged or missing files.
- **EMERGENCY** – Read-only access for troubleshooting.

---

12. How to perform an Always On Availability Group health check?

- Check the **failover readiness** using sys.dm_hadr_availability_replica_cluster_states.
- Monitor **synchronization lag** using sys.dm_hadr_database_replica_states.
- Review the **Cluster Logs** for errors.
- Use DBCC CHECKDB to ensure no data corruption.

- Monitor **CPU, Memory, and Disk I/O** using SQL Profiler.

---

13. How do you manage SLAs and ticket handling as a DBA?

- **Prioritize Tickets**: Critical DB outages → Performance Issues → Maintenance.
- **Response Time Adherence**: Follow SLA guidelines for response and resolution.
- **Documentation**: Maintain logs of RCA (Root Cause Analysis) for future prevention.
- **Automation**: Use SQL Server Agent Jobs for recurring maintenance tasks.
- **Monitoring**: Implement alerts for CPU spikes, blocking sessions, and job failures.

---

14. What are the different types of High Availability (HA) and Disaster Recovery (DR) solutions in SQL Server?
SQL Server provides several HA and DR solutions:

Solution	Type	Description
Failover Clustering (FCI)	HA	Uses shared storage and automatic failover at the instance level.
Always On Availability Groups	HA/DR	Provides automatic failover and read-scale capabilities for multiple replicas.
Log Shipping	DR	Ships transaction logs to a secondary server at scheduled intervals.
Database Mirroring (Deprecated)	HA/DR	Provides real-time synchronization between a principal and mirror database.
Replication	DR	Copies and distributes data across multiple servers for scalability.
Backup and Restore	DR	Standard method to recover data from full, differential, and log backups.

---

15. What is a stored procedure, and what are its benefits?
A stored procedure is a precompiled collection of SQL statements that can be executed as a single unit.

Benefits:

- **Performance Improvement**: Reduces query parsing and compilation time.
- **Security**: Limits direct table access by users.
- **Code Reusability**: Can be used multiple times with different parameters.
- **Transaction Management**: Ensures ACID compliance by grouping multiple statements.

Example:

```
CREATE PROCEDURE GetEmployeeDetails
@EmpID INT
AS
BEGIN
   SELECT * FROM Employees WHERE EmployeeID = @EmpID
END
```

Usage:

```
EXEC GetEmployeeDetails @EmpID = 101
```

---

16. What is Row Versioning in SQL Server?

Row Versioning is a concurrency control mechanism that maintains multiple versions of a row to minimize locking conflicts. It is used in **Read Committed Snapshot Isolation (RCSI)** and **Snapshot Isolation**.

How it works:

- When a transaction modifies a row, the old version is stored in **tempdb**.
- Other transactions read the previous version instead of being blocked.

Benefits:

- Reduces deadlocks.
- Improves read performance by reducing blocking.

Enable RCSI:

```
ALTER DATABASE MyDB SET READ_COMMITTED_SNAPSHOT ON
```

---

17. What are indexes, and what is index fragmentation?

- **Indexes** improve query performance by reducing scan time.
- Types of indexes:
    - **Clustered Index**: Sorts and stores data physically (only one per table).
    - **Non-Clustered Index**: Stores pointers to data (multiple per table).
    - **Filtered Index**: Indexes specific rows.
    - **Columnstore Index**: Used in large data warehouses.

Index Fragmentation:
Occurs when pages are not stored sequentially, affecting performance.

- **Internal Fragmentation**: Free space within pages.
- **External Fragmentation**: Pages stored non-contiguously.

Fixing Fragmentation:

```
-- Check fragmentation
SELECT * FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL,
'DETAILED')

-- Reorganize (if fragmentation < 30%)
ALTER INDEX IDX_Name ON TableName REORGANIZE

-- Rebuild (if fragmentation > 30%)
ALTER INDEX IDX_Name ON TableName REBUILD
```

---

18. What is a Linked Server in SQL Server?

A linked server allows SQL Server to access data from remote databases or external data sources like Oracle, MySQL, or another SQL Server instance.

Create a linked server:

```
EXEC sp_addlinkedserver

   @server = 'RemoteSQLServer',
   @srvproduct = '',
   @provider = 'SQLNCLI',
   @datasrc = 'RemoteServerName'
```

Query a linked server:

```
SELECT * FROM RemoteSQLServer.DatabaseName.dbo.TableName
```

Benefits:

- Enables cross-server querying.
- Reduces data movement between servers.
- Supports heterogeneous databases.

---

19. How do you schedule a SQL Server maintenance job?

Scheduled maintenance tasks such as backups, index rebuilds, and integrity checks can be automated using **SQL Server Agent**.

Steps:

1. Open **SQL Server Management Studio (SSMS)** → Go to **SQL Server Agent**.
2. Create a **New Job** under SQL Server Agent.
3. Define **Steps** (e.g., Full Backup, Index Maintenance).
4. Set a **Schedule** (e.g., Daily at 2 AM).
5. Configure **Alerts** and **Notifications**.

Example Job to Rebuild Indexes:
EXEC sp_MSforeachtable 'ALTER INDEX ALL ON REBUILD'

---

20. How do you troubleshoot a failed SQL Server job?

Steps to Fix a Failed SQL Server Job:

1. **Check Job History Logs:**
   - Navigate to **SQL Server Agent → Job Activity Monitor**.
   - View the failed job logs for error messages.
2. **Check SQL Server Logs & Error Messages:**

   EXEC xp_readerrorlog

3. **Verify Job Owner and Permissions:**
   - Ensure the job owner has the necessary permissions.
   - Run the job manually to test.
4. **Check Disk Space & Performance Issues:**
   - Low disk space can cause backups and index maintenance to fail.
   - Use sp_who2 to check blocking issues.
5. **Fix Corrupt SQL Agent Service:**
   - Restart the SQL Server Agent:

     NET STOP SQLSERVERAGENT
     NET START SQLSERVERAGENT

6. **Retry the Job After Fixing the Issue:**

   EXEC msdb.dbo.sp_start_job @job_name = 'Job_Name' .

---

21. What is Blocking, Locking, and Deadlock in SQL Server?

Blocking:

- Occurs when one process is holding a lock on a resource, preventing other processes from accessing it until the lock is released.
- **Example:**
  - **Transaction 1:** Updates a record and holds an exclusive lock.
  - **Transaction 2:** Tries to read the same record but gets blocked until Transaction 1 commits or rolls back.
- **How to identify blocking?**

  ```
  EXEC sp_who2 -- Shows blocking SPIDs
  SELECT * FROM sys.dm_exec_requests WHERE blocking_session_id <> 0
  ```

- **How to resolve blocking?**
  - Identify the blocking session and kill it if necessary:

    ```
    KILL <SPID>
    ```

Locking:

- Locking is a mechanism that controls data concurrency and consistency by restricting access to resources.
- **Types of Locks in SQL Server:**
  - **Shared (S):** Allows read access; other transactions can also read.
  - **Exclusive (X):** Prevents other transactions from accessing the resource.
  - **Update (U):** Converts into an Exclusive lock if needed.
  - **Intent Locks (IX, IS, IU):** Signals that a more restrictive lock will be taken.
  - **Schema Locks (Sch-M, Sch-S):** Controls access to schema changes.

How to check active locks?

```
SELECT * FROM sys.dm_tran_locks
```

Deadlock:

- A deadlock occurs when two or more processes hold locks on resources that the other processes need, causing a cyclic dependency.

Example:

- **Transaction 1:** Locks Table A, waits for Table B.
- **Transaction 2:** Locks Table B, waits for Table A → Deadlock.

How to detect deadlocks?

```
SELECT * FROM sys.dm_tran_locks WHERE request_status = 'WAIT'
```

How to enable automatic deadlock detection?

```
ALTER DATABASE MyDB SET DEADLOCK_PRIORITY HIGH
```

Fixing Deadlocks:

- Use **Try-Catch with Retry Logic**.
- Use **NOLOCK (with caution) to avoid excessive blocking**:

  ```
  SELECT * FROM Employees WITH (NOLOCK)
  ```

- **Optimize queries to reduce contention**.

---

22. What are the Types of Encryption in SQL Server?

SQL Server provides **encryption** for both **data at rest** and **data in transit**.

Encryption for Data at Rest:

1. **Transparent Data Encryption (TDE)**
   - Encrypts database files at the file level.
   - Protects backups.
   - Uses a **Database Encryption Key (DEK)**.

   Enable TDE:

   ```
   CREATE DATABASE ENCRYPTION KEY
   WITH ALGORITHM = AES_256
   ENCRYPTION BY SERVER CERTIFICATE MyServerCert
   GO
   ALTER DATABASE MyDB SET ENCRYPTION ON
   ```

2. **Cell-Level Encryption (CLE)**
   - Encrypts specific columns.
   - Uses built-in functions like EncryptByKey() and DecryptByKey().

   Example:

```
OPEN SYMMETRIC KEY MyKey DECRYPTION BY CERTIFICATE MyCert
SELECT DecryptByKey(EncryptedColumn) FROM Employees
```

3. **Always Encrypted**
   - Encrypts data at the application level.
   - Supports **Deterministic** (same plaintext → same ciphertext) and **Randomized** encryption.

Encryption for Data in Transit:

1. **TLS (Transport Layer Security)**
   - Encrypts data transmitted between SQL Server and clients.
   - TLS 1.2 is recommended for security.
2. **SSL (Secure Sockets Layer)** (Deprecated)
   - Older encryption protocol.
   - Replaced by TLS.

Enable TLS/SSL Encryption in SQL Server:

- Install an **SSL certificate** on the server.
- Configure the connection string in applications:

```
Encrypt=True; TrustServerCertificate=False;
```

---

23. What are INIT, NOINIT, LOAD, NOLOAD, CHECKSUM, and HEADER_VERIFY in SQL Server?

INIT and NOINIT in Backups

- **INIT (Initialize Backup File):** Overwrites existing backup files.

```
BACKUP DATABASE MyDB TO DISK = 'C:\Backup\MyDB.bak' WITH INIT
```

- **NOINIT:** Appends the new backup to the existing backup set.

```
BACKUP DATABASE MyDB TO DISK = 'C:\Backup\MyDB.bak' WITH NOINIT
```

LOAD and NOLOAD in Restores

- **LOAD:** Used in older versions to force loading of a backup file (not required in SQL Server 2012+).
- **NOLOAD:** Prevents overwriting an existing database (default behavior).

CHECKSUM in Backups

- Ensures backup file integrity by adding a checksum.

  BACKUP DATABASE MyDB TO DISK = 'C:\Backup\MyDB.bak' WITH CHECKSUM

- To verify the checksum:

  RESTORE VERIFYONLY FROM DISK = 'C:\Backup\MyDB.bak' WITH CHECKSUM

HEADER_VERIFY

- Verifies the backup header without restoring the database.

  RESTORE HEADERONLY FROM DISK = 'C:\Backup\MyDB.bak'

---

24. Commonly Used DMV and DBCC Commands in SQL Server

Category	DMV / DBCC	Description
Session Monitoring	sys.dm_exec_sessions	Active sessions
	sys.dm_exec_requests	Running queries
	sys.dm_exec_connections	Active connections
Locking & Blocking	sys.dm_tran_locks	Lock details
	sys.dm_os_waiting_tasks	Blocking queries
Index & Performance	sys.dm_db_index_usage_stats	Index usage stats
	sys.dm_db_index_physical_stats	Index fragmentation
TempDB	sys.dm_db_file_space_usage	TempDB usage
Execution Plans	sys.dm_exec_query_plan(plan_handle)	Query execution plan
	sys.dm_exec_cached_plans	Cached plans
DBCC Maintenance	DBCC CHECKDB	Database consistency check
	DBCC CHECKTABLE	Table consistency check
DBCC Performance	DBCC SHOW_STATISTICS	Index statistics
	DBCC DBREINDEX	Rebuild index
	DBCC INDEXDEFRAG	Defragment index
DBCC Monitoring	DBCC SQLPERF(logspace)	Log space usage
	DBCC FREEPROCCACHE	Clear cache
	DBCC TRACEON(1222, -1)	Deadlock tracking

25. Scenario: Troubleshooting Blocking Issues in SQL Server

Scenario:

Users are complaining about slow performance in the database. You suspect blocking is causing the issue. How will you identify and resolve it?

Answer:

Blocking occurs when one transaction holds a lock on a resource and another transaction waits for it to be released.

Steps to Identify and Fix Blocking:

1. **Identify Blocking Sessions:**

   ```
   SELECT blocking_session_id, session_id, wait_type, wait_time, wait_resource
   FROM sys.dm_exec_requests
   WHERE blocking_session_id <> 0
   ```

2. **Check the Queries Causing the Block:**

   ```
   SELECT blocking_session_id, session_id, text
   FROM sys.dm_exec_requests
   CROSS APPLY sys.dm_exec_sql_text(sql_handle)
   WHERE blocking_session_id <> 0
   ```

3. **Check Locking Details:**

   ```
   SELECT request_session_id, resource_type, resource_description, request_mode
   FROM sys.dm_tran_locks
   WHERE request_mode <> 'S'
   ```

4. **Resolve the Block:**
   - If the blocking process is idle, **kill the session**:

     ```
     KILL <blocking_session_id>
     ```

- Optimize queries using **proper indexing** and **NOLOCK hints** (only if dirty reads are acceptable).
- **Implement deadlock handling** in stored procedures using **TRY…CATCH and SET LOCK_TIMEOUT**.

---

26. Scenario: Troubleshooting Deadlocks in SQL Server

Scenario:

You notice frequent deadlocks in the database, and users report transaction failures. How do you detect and resolve deadlocks?

Answer:

Deadlocks occur when two or more transactions hold locks on resources the other needs, causing a circular dependency.

Steps to Identify and Resolve Deadlocks:

1. **Enable Deadlock Trace Flags:**

   ```
   DBCC TRACEON (1222, -1) -- Logs deadlock details in SQL Server error log
   ```

2. **Use Extended Events to Capture Deadlocks:**

   ```
   CREATE EVENT SESSION DeadlockMonitor
   ON SERVER
   ADD EVENT sqlserver.lock_deadlock_graph
   ADD TARGET package0.event_file (SET filename = 'C:\Temp\Deadlocks.xel')
   WITH (STARTUP_STATE = ON);
   ALTER EVENT SESSION DeadlockMonitor ON SERVER STATE = START;
   ```

3. **Check Deadlock Reports from System Views:**

   ```
   SELECT * FROM sys.dm_exec_requests WHERE blocking_session_id <> 0;
   ```

4. **Resolve the Deadlock:**
   - Use **NOLOCK** or **READ COMMITTED SNAPSHOT ISOLATION** if appropriate.
   - Optimize transactions by **keeping them short** and **accessing resources in the same order**.
   - Implement **deadlock retry logic** in applications.

31. Scenario: Performance Tuning Using Query Store in SQL Server

Scenario:

A user is complaining about slow performance for a specific query. How will you use Query Store to analyze and optimize query performance?

Answer:

To troubleshoot slow query performance using **Query Store**, follow these steps:

Step 1: Check if Query Store is Enabled

Query Store must be enabled for tracking past executions.

```
SELECT name, is_query_store_on
FROM sys.databases
WHERE name = 'MyDatabase';
```

☑ If Query Store is **not enabled**, turn it on:

```
ALTER DATABASE MyDatabase
SET QUERY_STORE = ON;
```

Step 2: Retrieve Query Store Execution Data

Query Store helps analyze query regressions, execution plans, and performance history.

☑ **Find the Slow Query Using Query Store Views**

```
SELECT q.query_id, q.query_hash, qt.query_sql_text, rs.avg_duration, rs.last_execution_time
FROM sys.query_store_query q
JOIN sys.query_store_query_text qt ON q.query_text_id = qt.query_text_id
JOIN sys.query_store_runtime_stats rs ON q.query_id = rs.query_id
ORDER BY rs.avg_duration DESC;
```

☑ This will show the **queries sorted by execution time**.

Step 3: Analyze Past Execution Plans

Identify if the query execution plan has changed recently.

```
SELECT plan_id, query_id, last_execution_time, avg_duration, execution_type_desc
FROM sys.query_store_plan
WHERE query_id = <query_id_from_previous_query>
ORDER BY last_execution_time DESC;
```

☑ If a **new execution plan** is performing worse than the old one, **force the previous plan**:

```
EXEC sp_query_store_force_plan @query_id = <query_id>, @plan_id = <previous_plan_id>;
```

Step 4: Check for Stale Statistics

Outdated statistics can lead to inefficient query plans.

```
SELECT name AS TableName, stats_date(object_id, stats_id) AS LastUpdated
FROM sys.stats
WHERE object_id = OBJECT_ID('MyTable');
```

☑ If statistics are outdated, update them:

```
UPDATE STATISTICS MyTable WITH FULLSCAN;
```

Step 5: Identify Unused Indexes

Unused indexes can slow down insert/update operations.

```
SELECT OBJECT_NAME(i.object_id) AS TableName, i.name AS IndexName,
    user_seeks, user_scans, user_lookups, user_updates
FROM sys.dm_db_index_usage_stats u
JOIN sys.indexes i ON u.object_id = i.object_id AND u.index_id = i.index_id
WHERE OBJECT_NAME(i.object_id) = 'MyTable'
ORDER BY user_updates DESC;
```

☑ If an index has **high updates but low seeks**, it may be unnecessary.

Step 6: Check for Recent Deployments That Could Impact Performance

Check when the last schema change was made.

```
SELECT name, create_date, modify_date
FROM sys.objects
WHERE type = 'P' AND name = 'MyStoredProc';
```

☑ If there was a **recent deployment**, analyze the impact.

Step 7: Check If the Table is Partitioned

Large tables benefit from partitioning for better query performance.

```
SELECT name AS TableName, p.partition_number, p.rows
FROM sys.partitions p
JOIN sys.tables t ON p.object_id = t.object_id
WHERE t.name = 'MyTable';
```

☑ If partitioning is missing and the table is large, **consider partitioning** based on common query filters.

Step 8: Monitor Wait Statistics for Query Delays

Identify if queries are waiting on CPU, IO, or locks.

```
SELECT wait_type, wait_time_ms / 1000 AS WaitSeconds, waiting_tasks_count
FROM sys.dm_exec_requests r
JOIN sys.dm_os_wait_stats w ON r.wait_type = w.wait_type
ORDER BY wait_time_ms DESC;
```

☑ **If there are high waits on IO or CPU**, consider indexing or query optimization.

Step 9: Capture Live Query Execution Plans

Use **Live Query Statistics** to monitor real-time execution plans.

```
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
EXEC sp_executesql N'SELECT * FROM MyTable WHERE Column1 = @Param1',
N'@Param1 INT', 123;
```

☑ Check execution plans for **missing indexes** or **poorly optimized operations**.

Step 10: Use Query Hints if Necessary

If optimization is difficult, use hints like:

- **OPTION (RECOMPILE)** → Forces recompilation
- **FORCESEEK** → Forces index seek
- **MAXDOP** → Limits CPU parallelism

```
SELECT * FROM MyTable
WHERE Column1 = 100
OPTION (RECOMPILE, MAXDOP 1);
```

Final Step: Implement Fixes & Monitor Performance Again

After making changes, **retest the query performance** and monitor Query Store for improvements.

---

36. Scenario: Transaction Log File Growing – Troubleshooting and Solutions

Problem Statement:

The SQL Server transaction log file is growing uncontrollably, and available disk space is decreasing. The log file cannot be shrunk, causing database performance issues.

Troubleshooting Steps & Solutions:

1️⃣ **Check Active Transactions Blocking Log Truncation**

```
DBCC OPENTRAN
```

☑ If there are active transactions, **commit or rollback** them.

2️⃣ **Check Log File Usage and Space Allocation**

```
DBCC SQLPERF(LOGSPACE)
```

☑ Identifies how much space is used in the log file.

3️⃣ **Ensure No Blocking or Deadlocks Are Preventing Log Clearance**

```
SELECT blocking_session_id, wait_type, wait_time, wait_resource
```

FROM sys.dm_exec_requests WHERE blocking_session_id <> 0;

☑ If blocking is found, resolve the blocking session.

4 **Verify if Transaction Log Backups Are Running (If in Full Recovery Mode)**

SELECT name, log_reuse_wait_desc FROM sys.databases WHERE name = 'MyDatabase'

☑ If log backups are not happening, **schedule regular log backups**:

BACKUP LOG MyDatabase TO DISK = 'D:\Backups\MyDatabase_log.trn'

5 **Check Recovery Model:**

- **Simple Mode:** The log should truncate automatically.
- **Full Mode:** If logs are not needed, **switch to Simple mode temporarily**:

  ALTER DATABASE MyDatabase SET RECOVERY SIMPLE;
  DBCC SHRINKFILE (MyDatabase_log, 500);
  ALTER DATABASE MyDatabase SET RECOVERY FULL;

☑ Only use this if point-in-time recovery is not required.

6 **Take Regular Log Backups and Try Shrinking Again**

BACKUP LOG MyDatabase TO DISK = 'D:\Backups\MyDatabase_log.trn'
DBCC SHRINKFILE (MyDatabase_log, 500);

7 **If Disk Has Free Space, Add an Additional Log File**

ALTER DATABASE MyDatabase
ADD LOG FILE (NAME = MyDatabase_log2, FILENAME =
'D:\SQLLogs\MyDatabase_log2.ldf', SIZE = 1GB, FILEGROWTH = 512MB);

8 **If Disk Has No Free Space and No Other User Database Exists, Add Log File to Another Database**

ALTER DATABASE MyDatabase
ADD LOG FILE (NAME = MyDatabase_log2, FILENAME =
'E:\SQLLogs\MyDatabase_log2.ldf', SIZE = 1GB, FILEGROWTH = 512MB);

☑ Only a temporary fix; **not recommended for long-term use**.

9 **If Disk Has No Free Space, Try Shrinking Another Database**

DBCC SHRINKFILE (OtherDB_log, 500);

☑ Helps reclaim space from other databases if possible.

🔟 **If All Possibilities Fail:**

- **Expand the disk** if storage can be increased.
- **Contact the application team** to check if processes generating excessive logs can be stopped.

---

37. Scenario: TempDB is Full – Troubleshooting and Solutions

Problem Statement:

TempDB has reached maximum capacity, causing query failures and performance degradation.

Troubleshooting Steps & Solutions:

1️⃣ **Check Active Transactions in TempDB**

DBCC OPENTRAN('tempdb')

☑ If any transactions are running, commit or rollback them.

2️⃣ **Check for Blocking or Deadlocks in TempDB**

SELECT blocking_session_id, wait_type, wait_time, wait_resource
FROM sys.dm_exec_requests WHERE blocking_session_id <> 0;

☑ Identify and resolve blocking sessions.

3️⃣ **Check Version Store Tables (For Read Committed Snapshot Isolation - RCSI)**

SELECT * FROM sys.dm_tran_version_store;

☑ Large version store usage can cause tempdb growth. Consider reducing long-running transactions.

4️⃣ **Check If Any Heavy DML Operations Are Running**

SELECT session_id, start_time, command, status, blocking_session_id
FROM sys.dm_exec_requests WHERE database_id = DB_ID('tempdb');

☑ If large **INSERT, UPDATE, DELETE, or SELECT INTO** queries are running, optimize or stop them.

5⃣ **Add Additional TempDB Files to Distribute Load**

```
ALTER DATABASE tempdb
ADD FILE (NAME = tempdb2, FILENAME = 'D:\TempDB2.ndf', SIZE = 500MB,
FILEGROWTH = 250MB);
```

☑ Helps avoid contention and improves performance.

6⃣ **Clear Unused Cache to Free Space in TempDB**

```
DBCC FREEPROCCACHE
DBCC DROPCLEANBUFFERS
DBCC FREESYSTEMCACHE ('ALL')
DBCC FREESESSIONCACHE
```

☑ Helps reclaim space occupied by unused execution plans.

7⃣ **Restart SQL Server (Only If Absolutely Necessary)**

- Restarting clears tempdb but disrupts all active connections.
- Ensure application downtime is planned before restarting.

8⃣ **Contact Application Team**

- If excessive tempdb usage is due to an **application issue**, request process optimization.
- Consider **index tuning** and **query optimization** to reduce tempdb dependency.

These structured solutions will help efficiently troubleshoot and resolve **log file growth** and **tempdb full issues**.

---

38. **Scenario: Database Crash on Thursday – Recovery Testing with Full, Differential, and Log Backups**

Problem Statement:

A user performs regular transactions on a database, but it crashed on Thursday. We have the following backups:

- ☑ **Full backup from Sunday**
- ☑ **Differential backups from Monday, Tuesday, and Wednesday**
- ☑ **Transaction log backups from Sunday to Thursday**

The user wants to test the recovery process in the following manner:

1. **Restore the Sunday full backup**.
2. **Apply Monday's differential backup**.
3. If the required data is found after **Monday's differential backup**, **stop** and bring the database online.
4. If the required data is **not found** after Monday's differential, continue applying **Tuesday, Wednesday, and log backups** without starting from the full backup again.

Step 1: Restore the Sunday Full Backup in Standby Mode

Start by restoring the **Sunday full backup**. This is done using **Standby Mode**, which keeps the database in a **read-only** state, allowing the user to check if the data after Sunday is sufficient.

```
RESTORE DATABASE MyDatabase
FROM DISK = 'D:\Backups\MyDatabase_FULL_SUNDAY.bak'
WITH STANDBY = 'D:\Backups\MyDatabase_Standby.undo', REPLACE;
```

- The **STANDBY** option ensures that the database is restored in **read-only mode**. The user can access the data and perform queries.
- The **undo file** (MyDatabase_Standby.undo) is created so that if needed, uncommitted transactions can be rolled back if you continue restoring other backups.

Step 2: Restore Monday's Differential Backup in Standby Mode

Next, restore **Monday's differential backup**. The database will continue to be in **standby mode**, allowing the user to check if the data from Monday's differential backup is sufficient.

```
RESTORE DATABASE MyDatabase
FROM DISK = 'D:\Backups\MyDatabase_DIFF_MONDAY.bak'
```

```
WITH STANDBY = 'D:\Backups\MyDatabase_Standby.undo';
```

- The **STANDBY** mode ensures that the database remains in read-only mode, so the user can check if the required data is there after applying the Monday differential backup.

Step 3: User Review After Monday's Differential Backup

- **If the user finds the required data** after applying the Monday differential backup, they can **stop** the restore process and **keep the database in standby mode** for further read-only access.
- **If the user does not find the required data** after Monday's differential, proceed to restore the subsequent backups (Tuesday, Wednesday, and the log backups).

Step 4: Restore Tuesday's Differential Backup in Standby Mode

If the user requires more recent data, restore **Tuesday's differential backup** in **standby mode**.

```
RESTORE DATABASE MyDatabase
FROM DISK = 'D:\Backups\MyDatabase_DIFF_TUESDAY.bak'
WITH STANDBY = 'D:\Backups\MyDatabase_Standby.undo';
```

- The database remains in **read-only mode**, and the user can check if the required data is present after applying Tuesday's differential.

Step 5: Restore Wednesday's Differential Backup in Standby Mode

If more recent data is still needed, restore **Wednesday's differential backup** in **standby mode**.

```
RESTORE DATABASE MyDatabase
FROM DISK = 'D:\Backups\MyDatabase_DIFF_WEDNESDAY.bak'
WITH STANDBY = 'D:\Backups\MyDatabase_Standby.undo';
```

- The database remains in **standby mode** for the user to check the required data after Wednesday's differential.

Step 6: Restore Transaction Log Backups in Standby Mode

If needed, restore the transaction log backups from Sunday to Thursday. Transaction log backups will allow you to bring the database closer to the desired point in time.

```
RESTORE LOG MyDatabase
FROM DISK = 'D:\Backups\MyDatabase_LOG_SUNDAY.trn'
WITH STANDBY = 'D:\Backups\MyDatabase_Standby.undo';

RESTORE LOG MyDatabase
FROM DISK = 'D:\Backups\MyDatabase_LOG_MONDAY.trn'
WITH STANDBY = 'D:\Backups\MyDatabase_Standby.undo';

RESTORE LOG MyDatabase
FROM DISK = 'D:\Backups\MyDatabase_LOG_TUESDAY.trn'
WITH STANDBY = 'D:\Backups\MyDatabase_Standby.undo';

RESTORE LOG MyDatabase
FROM DISK = 'D:\Backups\MyDatabase_LOG_WEDNESDAY.trn'
WITH STANDBY = 'D:\Backups\MyDatabase_Standby.undo';

RESTORE LOG MyDatabase
FROM DISK = 'D:\Backups\MyDatabase_LOG_THURSDAY.trn'
WITH STANDBY = 'D:\Backups\MyDatabase_Standby.undo';
```

- Each **log backup** brings the database closer to the state it was at after the last log backup (Thursday in this case).
- The database remains in **read-only** state until you are ready to finalize the recovery.

Step 7: User Decision After Applying Log Backups

- **If the user finds the required data** after applying the log backups up to Thursday, they can **stop** the restore process and keep the database in **standby mode**.
- **If the user wants to continue testing further**, continue applying additional log backups or any other necessary backup until the desired point in time is reached.

Step 8: Finalizing the Recovery Process

Once the user is satisfied with the data after the recovery point, you can **bring the database online** for regular use.

```
RESTORE DATABASE MyDatabase WITH RECOVERY;
```

- The **RECOVERY** option finalizes the restore process and allows the database to be brought back online in **read-write mode**.

Summary of Steps:

1. **Restore Full Backup (Sunday)** in **STANDBY mode**.
2. **Restore Monday Differential** in **STANDBY mode**.
3. **User Decision**: If the data is sufficient, **stop** and keep the database in **standby mode**.
4. If more data is required, continue restoring **Tuesday, Wednesday, and log backups** in **STANDBY mode**.
5. **Final Decision**: After applying the necessary backups, stop and bring the database **online** if needed, or continue testing further.
6. **Final Recovery**: If data is satisfactory, **bring the database online** with the RECOVERY option.

This approach using **Standby Mode** allows the user to test recovery at each intermediate point, without bringing the database fully online until they are satisfied with the data recovered.