Lab: Transaction Log Growth and Management in SQL Server

# Objective:

In this lab, you will:  
1. Create a database and a table designed for high transaction log growth.  
2. Use a script to insert large data into the table and observe the transaction log growth.  
3. Monitor the log file growth in real-time, and learn how SQL Server manages log file space.  
4. Understand the impact of uncommitted transactions on log file growth.  
5. Practice managing log file growth by setting reasonable limits.

# Pre-Lab Setup:

Ensure that you have the necessary permissions to create databases and tables.

# Step 1: Create a Playground Database

In the first part of the lab, you'll create a database that will serve as the "playground" for testing. This will be the environment where you'll trigger log file growth.

Instructions:  
1. Open a new query window in SQL Server Management Studio (SSMS).  
2. Run the following script to create the database.

-- Create a new database for the lab  
CREATE DATABASE LogGrowthDemo  
ON  
PRIMARY (  
 NAME = 'LogGrowthDemo\_Data',  
 FILENAME = 'C:\SQLData\LogGrowthDemo\_Data.mdf',  
 SIZE = 50MB,  
 MAXSIZE = UNLIMITED,  
 FILEGROWTH = 10MB  
)  
LOG ON (  
 NAME = 'LogGrowthDemo\_Log',  
 FILENAME = 'C:\SQLLogs\LogGrowthDemo\_Log.ldf',  
 SIZE = 10MB,  
 MAXSIZE = 5GB,  
 FILEGROWTH = 512MB  
);

# Step 2: Create a Table with a LOB Column

Next, you will create a table that includes a `NVARCHAR(MAX)` column, which will allow for large object (LOB) data. This will be the table you use to generate transaction log growth.

Instructions:  
1. Run the following script to create the table.

-- Setup: Create a table with a LOB column  
IF OBJECT\_ID('dbo.LobMonster') IS NOT NULL DROP TABLE dbo.LobMonster;  
CREATE TABLE dbo.LobMonster (  
 ID INT IDENTITY(1,1),  
 Payload NVARCHAR(MAX)  
);  
GO

# Step 3: Simulate Transaction Log Growth

Now, you’ll simulate a scenario where massive transaction logs are generated. This will involve running a script that inserts large chunks of data into the `LobMonster` table, without committing the transaction.

Instructions:  
1. Open \*\*Tab 1\*\* in SSMS.  
2. Run the following script to start a transaction and insert data in a loop.

USE LogGrowthDemo;  
GO  
  
-- Start one big, dirty transaction  
BEGIN TRAN EatTheLog;  
  
DECLARE @i INT = 0;  
DECLARE @Payload NVARCHAR(MAX);  
SET @Payload = REPLICATE(N'A', 8000); -- 8KB chunk  
  
-- Loop: Insert until your disk screams  
WHILE @i < 100000  
BEGIN  
 INSERT INTO dbo.LobMonster (Payload)  
 VALUES (@Payload + @Payload + @Payload + @Payload + @Payload + @Payload + @Payload + @Payload);   
 -- That's ~64KB per row. Do the math.  
  
 SET @i += 1;  
END  
  
GO 10  
-- Don't commit. That’s the whole point.  
--ROLLBACK -- Only if you're done committing crimes

# Step 4: Monitor Log Growth

Now, in \*\*Tab 2\*\*, you will monitor how the transaction log grows in real-time while the data is being inserted.

Instructions:  
1. Open \*\*Tab 2\*\* in SSMS.  
2. Run the following command to monitor log file usage:

-- Monitor log file usage  
DBCC SQLPERF(LOGSPACE);

Alternatively, for more detailed log file growth per database:

-- Monitor log file growth for a specific database  
SELECT   
 database\_id,   
 name,   
 size/128 AS SizeMB,   
 physical\_name   
FROM sys.master\_files   
WHERE type\_desc = 'LOG'   
 AND database\_id = DB\_ID('LogGrowthDemo'); -- Replace 'LogGrowthDemo' with your database name

# Step 5: Observe and Analyze

While the transaction is running in \*\*Tab 1\*\*, keep an eye on the log growth in \*\*Tab 2\*\*.

Key things to look for:  
- The \*\*log file size\*\* will continue to grow as the large transaction is being processed.  
- The \*\*log file growth\*\* may trigger \*\*autogrowth events\*\* depending on how much space is available.  
- SQL Server will \*\*not truncate the log\*\* until a checkpoint occurs, so the log will continue to grow as long as the transaction remains open.

# Step 6: Cleanup and Rollback

After you’ve observed the log growth, \*\*rollback the transaction\*\* to prevent unnecessary growth in the transaction log. This will return the database to its original state.

Instructions:  
1. Go back to \*\*Tab 1\*\* and run the following command to rollback the transaction:

-- Rollback the transaction  
ROLLBACK TRAN EatTheLog;

# Optional Step: Shrink the Log File (Optional)

If you wish, you can shrink the log file to release unused space back to the operating system.

Instructions:  
1. Run the following script to shrink the log file:

-- Shrink the log file  
DBCC SHRINKFILE('LogGrowthDemo\_Log', 1); -- Shrink to 1MB

# Conclusion:

In this lab, you:  
- Created a playground database and a table with a LOB column.  
- Simulated massive transaction log growth by inserting large data without committing.  
- Monitored the log file growth in real-time and observed the behavior of SQL Server during large transactions.  
- Cleaned up the log by rolling back the transaction and, optionally, shrinking the log file.

# Discussion Points:

- \*\*Why does the log grow?\*\* Discuss how transaction logs in SQL Server are used to maintain data integrity and provide recovery.  
- \*\*Impact of large transactions:\*\* What can happen if long-running transactions are left open, and how can this impact log growth?  
- \*\*Best practices for managing log file growth:\*\* What strategies can you use to prevent runaway log file growth in a production environment?