Question Answering Assignment

1. What's the difference between batch, script, transaction, BackUp ?

Batch

A batch is a group of one or more SQL statements sent to SQL Server for execution as a single unit.

Batches are separated by the GO statement in SQL Server Management Studio (SSMS) or other query tools. GO is not a SQL statement; it's a client-side command that signals the end of a batch.

Each batch is parsed, compiled, and executed independently.

Variables declared in one batch are not accessible in another batch.

Errors in one batch do not necessarily affect other batches unless they cause a connection failure.

So the use case is to group related SQL statements for sequential execution, often for performance or logical separation.

Example:

```
DECLARE @MyVar INT = 10;
SELECT @MyVar;
GO

-- This will fail because @MyVar is not accessible in this new batch
SELECT @MyVar;
```

Script

A script is a text file or a collection of SQL statements saved with a .sql extension (or executed directly in a query tool) that can include one or more batches.

A script is essentially a saved set of SQL commands that can be run multiple times.

It may contain multiple batches (separated by GO) and can include comments, variables, and flow control statements.

Scripts are typically used to automate tasks, deploy database changes, or execute repetitive queries. It can also be used to backup the database structure.

SO Used for database setup, maintenance, or running complex operations across multiple batches.

Example:

```
-- MyScript.sql

CREATE TABLE MyTable (ID INT);

GO

INSERT INTO MyTable VALUES (1);
```

Transaction

A transaction is a logical unit of work that contains one or more SQL statements, ensuring data integrity by following the ACID properties (Atomicity, Consistency, Isolation, Durability).

Transactions are defined using BEGIN TRANSACTION, COMMIT, and ROLLBACK statements.

All statements within a transaction are either fully completed (COMMIT) or undone (ROLLBACK) if an error occurs.

Transactions can span multiple batches, but a single batch can contain multiple transactions.

Transactions are critical for ensuring data consistency, especially in operations that modify data (e.g., INSERT, UPDATE, DELETE).

So Used to ensure data integrity during operations that involve multiple related changes (e.g., transferring money between accounts).

Example:

```
BEGIN TRANSACTION;
UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;
UPDATE Accounts SET Balance = Balance + 100 WHERE AccountID = 2;
IF @@ERROR <> 0
   ROLLBACK;
ELSE
   COMMIT;
```

Transaction

A transaction is a logical unit of work that contains one or more SQL statements, ensuring data integrity by following the ACID properties (Atomicity, Consistency, Isolation, Durability).

Transactions are defined using BEGIN TRANSACTION, COMMIT, and ROLLBACK statements.

All statements within a transaction are either fully completed (COMMIT) or undone (ROLLBACK) if an error occurs.

Transactions can span multiple batches, but a single batch can contain multiple transactions.

Transactions are critical for ensuring data consistency, especially in operations that modify data (e.g., INSERT, UPDATE, DELETE).

So Used to ensure data integrity during operations that involve multiple related changes (e.g., transferring money between accounts).

Example:

```
BEGIN TRANSACTION;
UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;
UPDATE Accounts SET Balance = Balance + 100 WHERE AccountID = 2;
IF @@ERROR <> 0
   ROLLBACK;
ELSE
   COMMIT;
```

Backup

A backup is a copy of a database (or its components, like transaction logs) created to protect against data loss and enable recovery in case of failure.

Backups are created using the BACKUP DATABASE or BACKUP LOG commands.

Types of backups include:

Full Backup: Copies the entire database.

Differential Backup: Copies only the data changed since the last full backup.

Transaction Log Backup: Copies the transaction log to allow point-in-time recovery.

Backups are stored in external files (e.g., .bak or .trn) and can be restored using the RESTORE command.

SO Used for disaster recovery, database migration, or creating a snapshot of the database for testing.

Example:

```
BACKUP DATABASE MyDatabase
TO DISK = 'C:\Backups\MyDatabase.bak'
WITH FORMAT;
```

2. What is meant by logging transactions and why does this happen ?

What is Transaction Logging?

In SQL Server, logging transactions refers to the process of recording all changes made to a database in the transaction log, a dedicated file (or set of files) that tracks every operation that modifies data (e.g., INSERT, UPDATE, DELETE) or the database structure (e.g., CREATE, ALTER, DROP). This process is integral to the database's ability to ensure data integrity, recoverability, and consistency.

Why Does Transaction Logging Happen?

Transaction logging occurs for several critical reasons, all tied to maintaining the reliability and integrity of the database:

1. **Ensuring Durability**: Durability property of ACID (Atomicity, Consistency, Isolation, Durability) ensures that once a transaction is committed, its changes are permanently saved, even in the event of a system failure (e.g., power outage or crash).
2. **Enabling Rollback (Atomicity and Consistency)**: Transaction logging allows SQL Server to undo (rollback) a transaction if it fails or is explicitly rolled back (e.g., using ROLLBACK).
3. **Crash Recovery**: In case of a system crash, SQL Server uses the transaction log to perform crash recovery when the database restarts.
4. **Point-in-Time Recovery**: In the Full or Bulk-Logged recovery models, transaction log backups allow restoring a database to a specific point in time.
5. **Replication and High Availability**: Transaction logs are used in features like transactional replication, log shipping, and Always On Availability Groups to propagate changes from one database (or server) to another.

3. What's the difference between soft delete and hard delete ?

Soft Delete

A soft delete marks a record as "deleted" without physically removing it from the database. This is typically done by adding a flag or status column (e.g., IsDeleted, DeletedAt) to indicate that the record is no longer active.

The record remains in the database but is excluded from queries by filtering on the flag (e.g., WHERE IsDeleted = 0).

Data can be easily restored by updating the flag (e.g., setting IsDeleted = 0).

Soft deletes preserve historical data, which is useful for auditing, tracking, or accidental deletion recovery.

Requires additional logic in queries to filter out "deleted" records.

SO Soft deletes are common in applications where data recovery is important, such as user accounts, order histories, or systems requiring audit trails (e.g., CRM systems, e-commerce platforms).

Example:

```
-- Table with soft delete column
CREATE TABLE Employees (
   Id INT PRIMARY KEY,
   Name NVARCHAR(100),
   IsDeleted BIT DEFAULT 0
);

-- Soft delete an employee
UPDATE Employees
SET IsDeleted = 1

WHERE Id = 1;

-- Query active employees
SELECT * FROM Employees WHERE IsDeleted = 0;
```

Hard Delete

A hard delete permanently removes a record from the database using the DELETE statement, making it unrecoverable without restoring from a backup.

The record is physically removed from the table and is no longer accessible.

Space occupied by the record is freed (or marked for reuse) in the database.

Recovery is only possible through backups or transaction log recovery (if available).

No additional columns or query logic are needed to manage deleted records.

SO Hard deletes are suitable for temporary data, non-critical records, or systems where storage optimization is a priority and data recovery is not a concern (e.g., cache tables, temporary logs).

Example

```sql
-- Table without soft delete
CREATE TABLE Employees (
   Id INT PRIMARY KEY,
   Name NVARCHAR(100)
);

-- Hard delete an employee
DELETE FROM Employees WHERE Id = 1;

-- Record is gone
SELECT * FROM Employees WHERE Id = 1; -- Returns no rows
```