1. Transaction in SQL

What is a Transaction?

A transaction in SQL is a logical unit of work that groups one or more database operations together so they are treated as a single, complete action. The main idea of a transaction is that either all operations succeed and are saved permanently in the database, or if any operation fails, all changes are undone and the database returns to its previous consistent state. This is especially important in systems that handle money, fines, bookings, or any critical data, such as the projects you usually work on.

ACID Properties of Transactions

1. **Atomicity** – All operations succeed as a single unit, or none are applied.
   Example: In a money transfer, if the amount is deducted from one account but not added to the other, the transaction should roll back.
2. **Consistency** – The database moves from one valid state to another.
   Example: Any primary key or unique constraints must still be valid after the transaction.
3. **Isolation** – Transactions do not interfere with each other.
   Example: If one user transfers money while another checks the balance, the second user should not see partial changes until the transaction completes.
4. **Durability** – Once committed, changes are permanent, even if the system crashes.

The basic commands used in a transaction are BEGIN TRANSACTION (or BEGIN TRAN), COMMIT, and ROLLBACK. BEGIN TRANSACTION starts the transaction and tells the database to temporarily hold the changes. COMMIT saves all the changes made during the transaction permanently. ROLLBACK cancels all the changes made since the transaction started and restores the previous state.

```sql
BEGIN TRANSACTION;  -- Start a transaction

-- Example of two operations
UPDATE Account
SET Balance = Balance - 500
WHERE AccountID = 101;

UPDATE Account
SET Balance = Balance + 500
WHERE AccountID = 102;

COMMIT;  -- Save changes if all operations succeed

ROLLBACK;  -- Undo all changes if any operation fails
```

In this example, money is transferred from one account to another. If both UPDATE statements run successfully, the COMMIT statement saves the changes. If an error occurs in any statement, you should use ROLLBACK to undo both updates so that no money is lost or created by mistake.

transactions are often combined with error handling using TRY…CATCH blocks. This allows you to automatically roll back the transaction if something goes wrong. For example:

```sql
BEGIN TRY
    BEGIN TRANSACTION;

    UPDATE Account
    SET Balance = Balance - 100
    WHERE AccountID = 1;

    UPDATE Account
    SET Balance = Balance + 100
    WHERE AccountID = 2;

    COMMIT;
END TRY
BEGIN CATCH
    ROLLBACK;
    PRINT 'Transaction failed and was rolled back';
END CATCH;
```

General Form of a Transaction in SQL Server

```sql
BEGIN TRANSACTION;

BEGIN TRY
    -- SQL operations that must succeed together
    UPDATE ...;
    INSERT ...;
    DELETE ...;
    -- etc.

    COMMIT TRANSACTION;  -- make changes permanent if all succeed
    PRINT 'Transaction successful!';
END TRY

BEGIN CATCH
    ROLLBACK TRANSACTION;  -- undo all changes if any error occurs
    PRINT 'Transaction failed: ' + ERROR_MESSAGE();
END CATCH;
```

Basic Example – Bank Transfer

```
BEGIN TRANSACTION;

DECLARE @FromBalance DECIMAL(10,2);
SELECT @FromBalance = Balance FROM Accounts WHERE AccountID = 101;

IF @FromBalance >= 500
BEGIN
    -- Deduct from source account
    UPDATE Accounts
    SET Balance = Balance - 500
    WHERE AccountID = 101;

    -- Add to destination account
    UPDATE Accounts
    SET Balance = Balance + 500
    WHERE AccountID = 102;

    COMMIT TRANSACTION;
    PRINT 'Transaction completed successfully!';
END
ELSE
BEGIN
    ROLLBACK TRANSACTION;
    PRINT 'Transaction failed: Not enough balance.';
END
```

Practical Use Cases of Transactions

- **Banking:** Transferring money between accounts.
- **E-commerce:** Adding products to cart and updating inventory during checkout.
- **University systems:** Registering a student for multiple courses at once; all registrations must succeed or fail together.

Why Transactions are Important

- Protects data from human or technical errors.
- Prevents inconsistencies across multiple tables.
- Ensures complex operations complete safely and reliably.

2. Stored Procedure in SQL

What is a Stored Procedure?

A Stored Procedure (SP) is a named set of SQL statements that is stored in a database and can be executed repeatedly. Unlike ad-hoc queries, stored procedures are precompiled, which improves performance and ensures consistency.
It can include:

- SELECT, INSERT, UPDATE, DELETE
- Conditional logic (IF / ELSE)
- Loops (FOR, WHILE)
- Input parameters for flexibility

Advantages of Stored Procedures

1. **Reusability:** Write once, execute multiple times.
2. **Performance:** Precompiled, so execution is faster.
3. **Security:** Users can execute procedures without direct access to tables.
4. **Error Reduction:** Centralized logic reduces coding mistakes.

Basic form of Stored Procedure

```
CREATE PROCEDURE ProcedureName
    @Parameter1 DataType,
    @Parameter2 DataType = DefaultValue
AS
BEGIN
    -- SQL statements
END;
```

Example :

```
-- Retrieve customer data by ID
CREATE PROCEDURE GetCustomerByID
    @CustomerID INT
AS
BEGIN
    SELECT FullName, Email, Phone
    FROM Customer
    WHERE CustomerID = @CustomerID;
END;
```

Executing the Stored Procedure:

EXEC GetCustomerByID @CustomerID = 1

Real-Life Uses:

- **Banking:** Safe money transfers.
- **Inventory systems:** Updating stock and order details together.
- **University systems:** Registering a student for multiple courses while checking prerequisites.