1️⃣ What is a Stored Procedure?

A **stored procedure** is a **precompiled collection of SQL statements** stored in the database and executed as a single unit.

Think of it as:

A reusable SQL program saved inside the database.

2️⃣ Why Use Stored Procedures?

✓ Advantages

- **Reusability** – write once, use many times
- **Better performance** – compiled & cached execution plans
- **Security** – restrict direct table access
- **Maintainability** – update logic in one place
- **Reduced network traffic** – send procedure name, not full SQL

3️⃣ Basic Syntax (SQL Server)

Create a Stored Procedure

```
CREATE PROCEDURE GetAllStudents
AS
BEGIN
   SELECT * FROM Student;
END;
```

Execute a Stored Procedure

```
EXEC GetAllStudents;
-- OR
```

```
EXECUTE GetAllStudents;
```

4 Stored Procedure with Parameters

Input Parameters

```
CREATE PROCEDURE GetStudentById
   @StudentId INT
AS
BEGIN
   SELECT *
   FROM Student
   WHERE S_Id = @StudentId;
END;
```

Execution:

```
EXEC GetStudentById 5;
```

Multiple Parameters

```
CREATE PROCEDURE GetStudentsByDept
   @DeptId INT,
   @MinAge INT
AS
BEGIN
   SELECT *
   FROM Student
   WHERE D_Id = @DeptId AND Age >= @MinAge;
END;
```

5 Output Parameters

Used to return values to the caller.

```
CREATE PROCEDURE GetStudentCount
   @DeptId INT,
   @TotalStudents INT OUTPUT
AS
BEGIN
   SELECT @TotalStudents = COUNT(*)
   FROM Student
   WHERE D_Id = @DeptId;
END;
```

Execution:

```
DECLARE @Count INT;

EXEC GetStudentCount 2, @Count OUTPUT;

SELECT @Count AS TotalStudents;
```

6 Stored Procedures with INSERT / UPDATE / DELETE

INSERT

```
CREATE PROCEDURE AddStudent
   @Id INT,
   @FName NVARCHAR(50),
   @LName NVARCHAR(50)
AS
BEGIN
   INSERT INTO Student (S_Id, F_Name, L_Name)
   VALUES (@Id, @FName, @LName);
END;
```

UPDATE

```
CREATE PROCEDURE UpdateStudentPhone
   @Id INT,
   @Phone INT
AS
BEGIN
   UPDATE Student
   SET Phone_No = @Phone
   WHERE S_Id = @Id;
END;
```

DELETE

```
CREATE PROCEDURE DeleteStudent
   @Id INT
AS
BEGIN
   DELETE FROM Student
   WHERE S_Id = @Id;
END;
```

7 Conditional Logic in Stored Procedures

```
CREATE PROCEDURE CheckStudentAge
   @Age INT
AS
BEGIN
   IF @Age >= 18
      SELECT 'Adult' AS Status;
   ELSE
      SELECT 'Minor' AS Status;
END;
```

8 Using TRY...CATCH (Error Handling)

```
CREATE PROCEDURE SafeInsertStudent
   @Id INT,
   @Name NVARCHAR(50)
AS
BEGIN
   BEGIN TRY
      INSERT INTO Student (S_Id, F_Name)
      VALUES (@Id, @Name);
   END TRY
   BEGIN CATCH
      SELECT ERROR_MESSAGE() AS ErrorMessage;
   END CATCH
END;
```

9 Altering a Stored Procedure

```
ALTER PROCEDURE GetAllStudents
AS
BEGIN
   SELECT S_Id, F_Name, L_Name FROM Student;
END;
```

10 Dropping a Stored Procedure

```
DROP PROCEDURE GetAllStudents;
```

1️⃣1️⃣ Types of Stored Procedures

Type	Description
System	Built-in (e.g. sp_help)
User-defined	Created by users
Temporary	#TempProc (session-based)
CLR	Written in .NET languages

1️⃣2️⃣ Stored Procedure vs Function

Stored Procedure	Function
Can modify data	Usually cannot
Can't be used in SELECT	Can be used in SELECT
Supports output params	Returns value
No return mandatory	Must return value

1️⃣3️⃣ Best Practices

✓ Use meaningful names (usp_AddStudent)
✓ Avoid SELECT *
✓ Use parameters
✓ Handle errors (TRY...CATCH)
✓ Use transactions for critical changes

A **transaction** in SQL is a **logical unit of work** that contains one or more SQL statements and is treated as a **single, all-or-nothing operation**.

Think of it like:

Either everything succeeds, or everything is undone.

1️⃣ Why Do We Need Transactions?

Transactions protect the database from:

- Partial updates
- Data inconsistency
- Errors during multiple related operations

Example (Real-life)

💳 **Bank transfer**

- Deduct money from Account A
- Add money to Account B

If one fails, both must fail → **transaction**

2️⃣ ACID Properties (Very Important ⭐)

Every transaction follows **ACID**:

Property	Meaning
A – Atomicity	All or nothing
C – Consistency	Data remains valid
I – Isolation	Transactions don't interfere
D – Durability	Committed data is permanent

3️⃣ Transaction Commands

Start Transaction

```
BEGIN TRANSACTION;
-- or
BEGIN TRAN;
```

Save Changes

```
COMMIT;
```

Undo Changes

```
ROLLBACK;
```

4️⃣ Simple Example

```
BEGIN TRANSACTION;

INSERT INTO Student (S_Id, F_Name) VALUES (10, 'Ali');
INSERT INTO Student (S_Id, F_Name) VALUES (11, 'Sara');

COMMIT;
```

✓ Both inserts succeed → saved
❌ Any error → nothing is saved (if rolled back)

5️⃣ Transaction with ROLLBACK

```
BEGIN TRANSACTION;

INSERT INTO Account VALUES (1, 500);
INSERT INTO Account VALUES (1, 1000); -- error (duplicate key)

ROLLBACK;
```

➡ Result: no rows inserted

6 Transactions + TRY...CATCH (Best Practice)

```
BEGIN TRANSACTION;

BEGIN TRY
    UPDATE Account
    SET Balance = Balance - 100
    WHERE AccId = 1;

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

    COMMIT;
END TRY
BEGIN CATCH
    ROLLBACK;
    SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
```

✓ Ensures safe execution

✓ Common in **stored procedures**

7 Transactions in Stored Procedures

```
CREATE PROCEDURE TransferMoney
    @FromAcc INT,
    @ToAcc INT,
    @Amount DECIMAL(10,2)
AS
BEGIN
    BEGIN TRAN;

    BEGIN TRY
        UPDATE Account
        SET Balance = Balance - @Amount
```

```
        WHERE AccId = @FromAcc;

        UPDATE Account
        SET Balance = Balance + @Amount
        WHERE AccId = @ToAcc;

        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
    END CATCH;
END;
```

8 SAVEPOINT (Partial Rollback)

```
BEGIN TRAN;

INSERT INTO Student VALUES (1, 'Ahmed');
SAVE TRAN sp1;

INSERT INTO Student VALUES (1, 'Ali'); -- error

ROLLBACK TRAN sp1; -- rollback only last insert
COMMIT;
```

9 Implicit vs Explicit Transactions

Type	Description
Implicit	Each statement auto-commits
Explicit	You control BEGIN / COMMIT / ROLLBACK

SQL Server default → **implicit OFF**

🔟 Transactions vs Stored Procedures

Transaction	Stored Procedure
Controls data consistency	Stores SQL logic
Uses BEGIN / COMMIT / ROLLBACK	Can contain transactions
Temporary	Permanent object

1️⃣1️⃣ When Should You Use Transactions?

✓ Money transfers
✓ Order processing
✓ Inventory updates
✓ Student enrollment systems
✓ Any **multi-step data change**

1️⃣2️⃣ Common Mistakes ❌

❌ Forgetting to `COMMIT`
❌ Leaving open transactions
❌ Using transactions for SELECT only
❌ Long transactions (cause locking)