Research

1 VIEW

What it is:
A **View** is a *virtual table* created from a SQL query. It does **not store data** itself; it shows data from one or more tables.

Why we use it:

- Simplify complex queries
- Improve security (hide sensitive columns)
- Reuse common queries

Type	Description
Simple View	From one table
Complex View	Uses JOIN, GROUP BY
Updatable View	Can INSERT/UPDATE
Read-Only View	Cannot modify data

Type of view:

View Type	Purpose
Standard View	Simplify queries
Complex View	Reports & analysis
Indexed View	High performance
Partitioned View	Large data sets
Updatable View	Controlled data updates
Secure View	Data security

2 Transaction

What it is:

A **Transaction** is a group of SQL operations executed as **one unit**.

Either *all* changes happen, or *none* happen.

Key properties (ACID):

- **Atomicity** – all or nothing
- **Consistency** – data stays valid
- **Isolation** – transactions don't interfere
- **Durability** – changes are permanent after commit

Commands:

- `BEGIN TRANSACTION`
- `COMMIT`
- `ROLLBACK`

◇ Real-Life Example

💳 Bank transfer:

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

  If one fails → rollback everything

🔹 ACID Properties

Property	Meaning
Atomicity	All or nothing
Consistency	Data remains valid
Isolation	Transactions don't interfere
Durability	Changes are permanent

◇ Advantages

✅ Data integrity

✅ Error recovery

Type of Transaction:

1. Read-Only Transaction

- Only **retrieves data** (no changes).
- Uses `SELECT` statements.
- Example:
- ```sql
  SELECT * FROM Accounts WHERE CustomerID = 101;
  ```

Safe and fast; no risk of data modification.

2. Read-Write Transaction

- **Reads and modifies** data.
- Uses `INSERT`, `UPDATE`, or `DELETE`.
- Example:

```sql
BEGIN TRANSACTION;
UPDATE Accounts SET Balance = Balance - 500 WHERE AccountID = 1;
UPDATE Accounts SET Balance = Balance + 500 WHERE AccountID = 2;
COMMIT;
```

3. Implicit Transaction

- Automatically handled by the DBMS.
- Each statement is treated as a transaction.
- Common in simple operations.

4. Explicit Transaction

- Manually controlled by the user.
- Uses:
  - `BEGIN TRANSACTION`
  - `COMMIT`
  - `ROLLBACK`
```

- Gives full control over success or failure.

5. Single Transaction

- Contains **one SQL statement**.
- Example:

```
DELETE FROM Loans WHERE LoanID = 10;
```

6. Multi-Statement Transaction

- Contains **multiple SQL statements**.
- All succeed or all fail together.
- Common in banking and inventory systems.

7. Distributed Transaction

- Spans **multiple databases or servers**.
- Requires coordination (e.g., two-phase commit).
- Example: online payment involving bank + payment gateway.

8. Nested Transaction

- A transaction inside another transaction.
- Supported logically in SQL Server using savepoints.
- Example:

```
SAVE TRANSACTION SavePoint1;
```

9. Long-Running Transaction

- Takes a long time to complete.

- Example: monthly payroll processing.
- Can impact performance if not managed carefully.

Summary Table:

Type	Description
Read-Only	Data retrieval only
Read-Write	Data modification
Implicit	Auto-managed
Explicit	User-controlled
Single	One statement
Multi-statement	Multiple statements
Distributed	Multiple databases
Nested	Transaction inside another
Long-running	Takes extended time

3 Stored Procedure

What it is:
A **Stored Procedure** is a *prewritten SQL program* saved in the database and executed when needed.

Why we use it:

- Faster execution
- Reusable logic
- Better security
- Reduces SQL code repetition

🔁 COMPARISON TABLE:

Feature	View	Transaction	Stored Procedure
Stores data	❌ No	❌ No	❌ No
Accepts parameters	❌ No	❌ No	✅ Yes
Contains logic	❌ No	⚠️ Control only	✅ Yes
Improves security	✅ Yes	❌ No	✅ Yes
Improves performance	⚠️ Sometimes	❌ No	✅ Yes

Types of Stored Procedures:

1. System Stored Procedures

- Built into the DBMS (especially SQL Server).
- Used to manage and get information about the database.
- Stored in the `master` database.
- Usually start with `sp_`.

```sql

sp_help Employees;
sp_databases;
```

Use case:
Check table structure, database info, users, permissions.

2. User-Defined Stored Procedures

- Created by database developers.
- Used to implement business logic.

3. Stored Procedures with Parameters

- Accept input values.
- Can return different results based on input.

4. Stored Procedures with Output Parameters

- Return values using OUTPUT parameters.

5. Stored Procedures with Return Value

- Return a single integer value using `RETURN`

6. Encrypted Stored Procedures

- Definition is hidden using `WITH ENCRYPTION`.

Example:

```
CREATE PROCEDURE SecureProc
WITH ENCRYPTION
AS
SELECT * FROM Employees;
```

7. Temporary Stored Procedures

- Exist temporarily.
- Local (#) or Global (##).

8. CLR Stored Procedures (Advanced)

- Written in .NET languages (C#, VB).
- Used when SQL logic isn't enough.

Use case:
Complex calculations, file handling.

Summary Table:

Type	Purpose
System	DB management
User-defined	Business logic
With parameters	Dynamic input
Output parameters	Return values
Return value	Status codes
Encrypted	Hide logic
Temporary	Short-term use
CLR	Advanced logic