1. Types of Views in SQL Server:

1. **Standard View** A standard view is a simple, logical representation of data from one or more tables. It does not store data physically; instead, it retrieves data dynamically when queried.

- **Use Case:**
  Show a simplified view of customer information for a helpdesk.
- **Limitations:**
  Can't be indexed unless it meets specific criteria. Performance depends on base tables.

Example:

CREATE VIEW EmployeeDetails AS SELECT EmployeeID, FirstName, LastName, Department FROM Employees WHERE IsActive = 1; This view retrieves active employees' details from the Employees table.

2. **Indexed View** (Materialized View) An indexed view stores the result set physically on disk, improving performance for frequently accessed queries. It requires certain conditions, such as using WITH SCHEMABINDING.

- **Use Case:**
  Useful in banking for summaries like daily transaction totals.
- **Limitations:**
  Limited to deterministic functions, cannot use COUNT(*), TOP, etc. Requires SCHEMABINDING.

Example:

CREATE VIEW SalesSummary WITH SCHEMABINDING AS SELECT ProductID, SUM(SalesAmount) AS TotalSales FROM dbo.Sales GROUP BY ProductID;

CREATE UNIQUE CLUSTERED INDEX IX_SalesSummary ON SalesSummary(ProductID); This indexed view calculates total sales per product and stores the result for faster access.

3. **Partitioned View** A partitioned view combines data from multiple tables, often distributed across different servers or databases, to appear as a single table. It can be local or distributed.

- **Use Case:**
  Combine branch-wise transaction logs into one virtual table.
- **Limitations:**
  Complexity increases with number of partitions. Updateable only under strict rules.

Example (Local Partitioned View):

CREATE VIEW AllRegionsSales AS SELECT * FROM Sales_North UNION ALL SELECT * FROM Sales_South UNION ALL SELECT * FROM Sales_East UNION ALL SELECT * FROM Sales_West;

2. Can We Use DML on Views?

- **Yes**, DML (INSERT, UPDATE, DELETE) is allowed on views *if* the view:
  - References only one base table.
  - Doesn't use GROUP BY, DISTINCT, aggregates, JOINs (in most cases).
  - Doesn't include computed columns or non-updatable expressions.

Restrictions:

- Updates that would affect multiple base tables are not allowed.
- For indexed views, all DML operations must maintain index integrity.

Real-life Example:
In an HR system, a view showing active employees can be updated directly to mark resignations or promotions.

3. How Views Simplify Complex Queries

In SQL Server, Views are powerful tools that simplify complex queries by encapsulating frequently used logic—especially those involving multiple joins—into a single reusable object. This is particularly valuable in large database systems where teams need fast, secure, and consistent access to data without constantly rewriting lengthy JOIN operations or WHERE clauses.

By using a view, you abstract away the complexity of the underlying tables and relationships. This makes queries easier to read, reduces code duplication, and minimizes the chances of introducing errors. It also improves maintainability: when the business logic changes, only the view needs to be updated instead of every query that depends on it.

Example Scenario: Account Summary for Call Center Agents

A bank's customer support team frequently needs to access account summaries, including customer names, phone numbers, account IDs, balances, and account types. Writing a long JOIN query each time is inefficient.

☑ Without a View (Raw Query):

```
SELECT
    c.FullName,
    c.Phone,
    a.AccountID,
    a.AccountType,
    a.Balance
FROM
    Customer c
JOIN
    Account a ON c.CustomerID = a.CustomerID
WHERE
    a.Status = 'Active';
```

☑ With a View:

```
-- Create the View once
CREATE VIEW CustomerAccountSummary AS
SELECT
    c.FullName,
    c.Phone,
    a.AccountID,
    a.AccountType,
    a.Balance,
    a.Status
FROM
    Customer c
JOIN
    Account a ON c.CustomerID = a.CustomerID;


Now, the support team can simply use:

SELECT * FROM CustomerAccountSummary WHERE Status = 'Active';
```

Key Benefits:

- **Simplicity**: Reduces repetitive code in applications and reports.
- **Consistency**: Ensures all departments use the same logic and data representation.
- **Security**: Sensitive fields like SSN can be excluded from the view, protecting private data.

- **Performance**: If needed, views can be indexed for better performance in read-heavy operations.

Part 2 & 3

1. Customer Service View • Show only customer name, phone, and account status (hide sensitive info like SSN or balance).

```
CREATE VIEW CustomerServiceView AS

SELECT

  c.FullName, c.Phone,a.AccountID, a.Status

FROM Customer c

JOIN

   Account a ON c.CustomerID = a.CustomerID;
```

2. Finance Department View • Show account ID, balance, and account type.

```
CREATE VIEW FinanceView AS

SELECT

   AccountID,Balance,AccountType

FROM

   Account;
```

3. Loan Officer View • Show loan details but hide full customer information. Only include CustomerID.

```
CREATE VIEW LoanOfficerView AS

SELECT

   LoanID,CustomerID,LoanAmount,LoanTyp,Status

FROM

   Loan;
```

4. Transaction Summary View • Show only recent transactions (last 30 days) with account ID and amount

```
CREATE VIEW TransactionSummaryView AS

SELECT

  AccountID,Amount,Type,TransactionDate

FROM

  [Transaction]

WHERE

  TransactionDate >= DATEADD(DAY, -30, GETDATE());
```