Part 1: Research & Documentation

- ✓ **Types of Views in SQL Server**

1. Standard View

What is it?

A **Standard View** is a virtual table created using a SELECT statement. It does not store data physically; instead, it dynamically retrieves data from underlying base tables each time it is queried.

Key Differences

- Does **not store data**
- Query is executed every time the view is accessed
- Simplest and most commonly used view type
- Can often allow DML operations (with restrictions)

Real-Life Use Case (Banking)

A bank creates a view to expose **customer and account summary** data to call center agents without giving access to sensitive columns like PINs or passwords.

Limitations & Performance

- Performance depends on the underlying query
- Complex joins may slow execution
- No built-in indexing (except on base tables)

2. Indexed View

What is it?

An **Indexed View** physically stores the result set of the view and creates a **clustered index** on it. SQL Server maintains the data automatically.

Key Differences

- Stores data physically
- Improves performance for aggregation-heavy queries
- Requires `SCHEMABINDING`
- Stricter rules than standard views

Real-Life Use Case (Banking)

A bank uses an indexed view to store **daily total transaction amounts per account** for fast reporting and fraud detection.

Limitations & Performance

- Higher storage usage
- Slower INSERT/UPDATE/DELETE on base tables
- Many restrictions (no `OUTER JOIN`, no `COUNT(*)`, etc.)
- Suitable mainly for read-heavy systems

3. Partitioned View (Union View)

What is it?

A **Partitioned View** combines multiple tables (usually partitioned by year, region, or branch) using `UNION ALL`, making them appear as a single table.

Key Differences

- Used to split data across multiple tables
- Helps manage large datasets
- Can support horizontal scaling

Real-Life Use Case (Banking)

Transactions are stored in separate tables per year (`Transactions_2023`, `Transactions_2024`), and a partitioned view allows querying all transactions seamlessly.

Limitations & Performance

- Requires strict constraints (CHECK)
- DML operations must affect only one partition
- Complex to manage

✓ DML Operations on Views

Can We Use INSERT, UPDATE, DELETE on Views?

Yes, **but with limitations**.

Views That Allow DML

- **Standard Views** (with restrictions)
- **Partitioned Views** (limited)
- **Indexed Views** → × DML not allowed directly

Restrictions on DML

DML is allowed only if:

- The view references **one base table**
- No `DISTINCT`, `GROUP BY`, `HAVING`
- No aggregate functions
- No joins (in most cases)
- No computed columns

Real-Life Example (HR System)

An HR manager updates employee contact details through a view instead of the base table to ensure security and validation.

```
CREATE VIEW vw_EmployeeContact
AS
SELECT EmployeeID, Phone, Email
FROM Employees;
```

```
UPDATE vw_EmployeeContact
SET Phone = '99999999'
WHERE EmployeeID = 101;
```

✓ How Views Simplify Complex Queries

Why Views Are Useful?

- Hide complex JOIN logic
- Improve readability and reusability
- Centralize business logic
- Improve security by limiting column access

Banking Scenario: Account Summary for Call Center Agents

Tables Used:

- `Customer(CustomerID, FullName, Phone)`
- `Account(AccountID, CustomerID, Balance, AccountType)`

Without a View (Complex Query)

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

This query must be rewritten every time.

Creating a View

```
CREATE VIEW vw_CustomerAccountSummary
AS
SELECT c.CustomerID, c.FullName, c.Phone,
       a.AccountID, a.Balance, a.AccountType
FROM Customer c
JOIN Account a ON c.CustomerID = a.CustomerID;
```

Using the View (Simplified Query)

```
SELECT *
FROM vw_CustomerAccountSummary
WHERE Balance > 5000;
```

Benefits

- No repeated joins
- Easy for non-technical users
- Consistent business logic
- Enhanced security

Part 2: Real-Life Implementation Task (Banking System)

```sql
CREATE DATABASE banking;
USE banking;

CREATE TABLE Customer (
CustomerID INT PRIMARY KEY,
FullName NVARCHAR(100),
Email NVARCHAR(100),
Phone NVARCHAR(15),
SSN CHAR(9)
);
CREATE TABLE Account (
AccountID INT PRIMARY KEY,
CustomerID INT FOREIGN KEY REFERENCES Customer(CustomerID),
Balance DECIMAL(10, 2),
AccountType VARCHAR(50),
Status VARCHAR(20)
);
CREATE TABLE Transaction_b (
TransactionID INT PRIMARY KEY,
AccountID INT FOREIGN KEY REFERENCES Account(AccountID),
Amount DECIMAL(10, 2),
Type VARCHAR(10), -- Deposit, Withdraw
TransactionDate DATETIME
);
CREATE TABLE Loan (
LoanID INT PRIMARY KEY,
CustomerID INT FOREIGN KEY REFERENCES Customer(CustomerID),
LoanAmount DECIMAL(12, 2),
LoanType VARCHAR(50),
Status VARCHAR(20)
);
```

```sql
INSERT INTO Customer VALUES
(1, 'Ali Khan', 'ali@email.com', '91234567', '123456789'),
(2, 'Sara Ahmed', 'sara@email.com', '92345678', '987654321');

INSERT INTO Account VALUES
(101, 1, 8500.00, 'Savings', 'Active'),
(102, 2, 1200.00, 'Checking', 'Inactive');

INSERT INTO [Transaction_b] VALUES
(1001, 101, 500.00, 'Deposit', GETDATE() - 5),
(1002, 101, 200.00, 'Withdraw', GETDATE() - 20),
(1003, 102, 300.00, 'Deposit', GETDATE() - 40);

INSERT INTO Loan VALUES
(201, 1, 50000.00, 'Home Loan', 'Approved'),
(202, 2, 15000.00, 'Car Loan', 'Pending');
--------------------------------
SELECT * FROM Customer;
SELECT * FROM Account;
SELECT * FROM [Transaction_b];
SELECT * FROM Loan;
```

Results Messages

	CustomerID	FullName	Email	Phone	SSN
1	1	Ali Khan	ali@email.com	91234567	123456789
2	2	Sara Ahmed	sara@email.com	92345678	987654321

	AccountID	CustomerID	Balance	AccountType	Status
1	101	1	8500.00	Savings	Active
2	102	2	1200.00	Checking	Inactive

	TransactionID	AccountID	Amount	Type	TransactionDate
1	1001	101	500.00	Deposit	2025-12-24 09:23:24.943
2	1002	101	200.00	Withdraw	2025-12-09 09:23:24.943
3	1003	102	300.00	Deposit	2025-11-19 09:23:24.943

	LoanID	CustomerID	LoanAmount	LoanType	Status
1	201	1	50000.00	Home Loan	Approved
2	202	2	15000.00	Car Loan	Pending

Part 3: View Creation Scenarios

Use Simple Views to implement the following:

1. Customer Service View

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

```
---1. Customer Service View
CREATE VIEW vw_CustomerService
AS
SELECT c.FullName,
       c.Phone,
       a.Status
FROM Customer c
JOIN Account a ON c.CustomerID = a.CustomerID;

SELECT * FROM vw_CustomerService;
```

150 %

Results | Messages

	FullName	Phone	Status
1	Ali Khan	91234567	Active
2	Sara A...	92345678	Inac...

2. Finance Department View

• Show **account ID**, **balance**, and **account type**.

```
---2. Finance Department View
CREATE VIEW vw_FinanceAccounts
AS
SELECT AccountID,
       Balance,
       AccountType
FROM Account;

SELECT * FROM vw_FinanceAccounts;
```

150 %

Results | Messages

	AccountID	Balance	AccountType
1	101	8500.00	Savings
2	102	1200.00	Checking

3. Loan Officer View

• Show **loan details** but hide full customer information. Only include **CustomerID**.

```
---3. Loan Officer View
CREATE VIEW vw_LoanOfficer
AS
SELECT LoanID,
       CustomerID,
       LoanAmount,
       LoanType,
       Status
FROM Loan;

SELECT * FROM vw_LoanOfficer;
```

150 %

Results | Messages

	LoanID	CustomerID	LoanAmount	LoanType	Status
1	201	1	50000.00	Home Loan	Approved
2	202	2	15000.00	Car Loan	Pending

4. Transaction Summary View

• Show only **recent transactions** (last 30 days) with **account ID** and **amount**.

```sql
---4. Transaction Summary View
CREATE VIEW vw_RecentTransactions
AS
SELECT AccountID,
       Amount,
       TransactionDate
FROM [Transaction_b]
WHERE TransactionDate >= DATEADD(DAY, -30, GETDATE());

SELECT * FROM vw_RecentTransactions;
```

150 %

Results | Messages

	AccountID	Amount	TransactionDate
1	101	500.00	2025-12-24 09:23:24.943
2	101	200.00	2025-12-09 09:23:24.943