SQL Views – Research & Implementation Task

**Objective**: Deepen your understanding of **Views** in SQL Server by researching their types and applying them in a realistic, security-focused banking scenario.

# Part 1: Research & Documentation

Each team must research and **write a short report** that includes the following:

1. **Types of Views** in SQL Server:
   * Standard View
   * Indexed View
   * Partitioned View (Union View)
2. **For each type**, answer:

**1.1 Standard View**

* + What is it?

A Standard View is a virtual table based on a SELECT query. It does not store data physically but provides a way to simplify access to specific subsets of data.

* + Key differences from other view types.

• It is the simplest form of view.

* No indexing is applied.
* Primarily used for simplifying queries or restricting access to certain columns.

* + Real-life use cases (example from banking, e-commerce, university system…)

**Use Case (Banking):**  
A view that allows customer service agents to see a customer’s name and account status without exposing SSNs or balances.

* + Limitations and performance considerations.

- Cannot improve performance significantly.

-Not suitable for highly complex or performance-critical queries.

**1.2 Indexed View**

* **What is it?**  
  An Indexed View stores the result set of the view physically on disk. It has a unique clustered index created on it, making it behave similarly to a table.
* **Key Differences:**
  + Improves query performance significantly.
  + Requires stricter rules (e.g., deterministic expressions, schema binding).
* **Use Case (E-Commerce):**  
  A view aggregating daily sales data that is queried frequently by reporting tools.
* **Limitations & Performance:**
  + Cannot include TEXT, NTEXT, or IMAGE types.
  + Slower to update due to physical storage and index maintenance.
  + DML operations are possible but must adhere to strict conditions.

**1.3 Partitioned View (Union View)**

* **What is it?**  
  A Partitioned View combines multiple tables with identical structures using UNION ALL. It is often used to implement horizontal partitioning.
* **Key Differences:**
  + Allows data from multiple partitions (tables) to appear as one logical set.
  + Can span across databases or servers in distributed partitioned views.
* **Use Case (University System):**  
  Combining student records from different campuses or years into one virtual table.
* **Limitations & Performance:**
  + Must maintain identical table structures.
  + Cannot easily perform DML unless constraints are strictly followed.

## Can We Use DML (INSERT, UPDATE, DELETE) on Views?

* + Do some research and explain:
    - **Which types of views** allow DML operations?

**Standard Views:** Yes, if based on a single table with no joins, aggregates, or GROUP BY.

**Indexed Views:** Yes, under very strict conditions with schema binding and deterministic functions.

**Partitioned Views:** Yes, if properly configured with CHECK constraints.

* + - What are the **restrictions or limitations** when performing DML on a view?

Cannot include DISTINCT, aggregate functions, or joins if you want to perform DML.

All base columns being modified must be updatable.

Triggers and indexed views must follow schema binding.

* + - Give at least **one real-life example** where updating a view is useful (e.g., HR system, e-commerce orders, etc.)

**Real-life Example (HR System):**  
A view like ActiveEmployees shows only currently working employees. The HR team can use this view to update phone numbers or job titles without accessing the full employee table.

*Tip: Try to test this in SQL Server if possible using a simple view.*

## How Can Views Simplify Complex Queries?

* + Explain how a **View** can help simplify **JOIN-heavy queries**.
  + Create an example view that joins at least **two of your banking tables**, such as:
    - Customer + Account
    - Account + Transaction
  + Show how using the view reduces the need to repeat long queries.

**Benefits:**

* Abstracts complex JOIN conditions.
* Reduces repetition of query logic.
* Improves readability and maintainability.
* Helps grant fine-grained access control.

**Example:**

**Without View:**

**SELECT c.FullName, a.Balance**

**FROM Customer c**

**JOIN Account a ON c.CustomerID = a.CustomerID**

**WHERE a.Balance > 1000;**

**With View:**

**CREATE VIEW CustomerAccountSummary AS**

**SELECT**

**c.FullName,**

**c.Phone,**

**a.AccountID,**

**a.Balance,**

**a.Status**

**FROM Customer c**

**JOIN Account a ON c.CustomerID = a.CustomerID;**

**-- Then reuse it**

**SELECT FullName, Balance FROM CustomerAccountSummary WHERE Balance > 1000;**

**Real-life Use Case (Call Center):**

A call center agent regularly checks account status. Instead of writing a join query every time, they can simply query a predefined view to get the required information quickly and securely.

*Hint: Think of a scenario where a team needs to access information frequently — like account summaries for call center agents.*

**Submission Format**: PDF or Word document

*Best report will be highlighted and used as a template for future batches.*

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

You are working for a **Banking System**, and you need to define views to control access and simplify queries for various departments.

Use the following tables (create with dummy data if not already available):

## Tables:

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 (

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)

);

# Part 3: View Creation Scenarios

Use **Simple Views** to implement the following:

## Customer Service View

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

## Finance Department View

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

## Loan Officer View

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

## Transaction Summary View

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