SQL Views – Research & Implementation Task

Objective

This document explains SQL Server Views through research and practical implementation in a **banking system**, focusing on **security, performance, and query simplification**.

Part 1: Research & Documentation

1. Types of Views in SQL Server

1.1 Standard View (Regular View)

What is it?

A **Standard View** is a virtual table created using a SELECT query.
It does **not store data physically**—it stores only the query definition.

```
CREATE VIEW vw_ActiveAccounts
AS
SELECT AccountID, Balance, AccountType
FROM Account
WHERE Status = 'Active';
```

Key Differences

Feature	Standard View
Data Storage	❌ No
Performance Boost	❌ No
Supports DML	✅ Yes (with rules)
Complexity	Simple

Real-Life Use Case

Banking:
Customer service agents view **customer contact details** without seeing SSN or balances.

Limitations & Performance

- Query runs every time the view is accessed.
- No indexing (except metadata).
- Heavy joins can slow performance.

1.2 Indexed View

What is it?

An **Indexed View** stores the result set **physically on disk** by creating a **unique clustered index**.

```
CREATE VIEW dbo.vw_TotalBalance
WITH SCHEMABINDING
AS
SELECT CustomerID, SUM(Balance) AS TotalBalance
FROM dbo.Account
GROUP BY CustomerID;

CREATE UNIQUE CLUSTERED INDEX IX_TotalBalance
ON dbo.vw_TotalBalance(CustomerID);
```

Key Differences

Feature	Indexed View
Data Storage	✅ Yes
Performance	🚀 Fast for aggregates
Maintenance Cost	High
Requirements	Very strict

Real-Life Use Case

Banking:
Fast dashboards showing **total customer balances** for management reports.

Limitations

- Requires SCHEMABINDING
- No SELECT *, no outer joins
- Extra overhead on INSERT/UPDATE/DELETE

1.3 Partitioned View (Union View)

What is it?

A **Partitioned View** combines data from **multiple tables** using UNION ALL, often to simulate table partitioning.

```
CREATE VIEW vw_AllTransactions
AS
SELECT * FROM Transaction_2024
UNION ALL
SELECT * FROM Transaction_2025;
```

Key Differences

Feature	Partitioned View
Combines Tables	✅
Physical Storage	❌
Used For	Large datasets
DML Support	Limited

Real-Life Use Case

Banking:
Handling **year-wise transaction tables** without querying each one separately.

Limitations

- Complex setup
- Strict CHECK constraints
- Maintenance overhead

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

Yes—but with restrictions

Views that allow DML

- ✅ **Simple Standard Views**
- ❌ Indexed Views (limited)
- ❌ Complex Views (joins, group by, aggregates)

Restrictions

DML is **NOT allowed** if the view contains:

- GROUP BY
- DISTINCT
- JOIN
- Aggregates (SUM, COUNT)
- UNION

Example of Updatable View

```
CREATE VIEW vw_ActiveCustomers
AS
SELECT CustomerID, FullName, Phone
FROM Customer
WHERE Phone IS NOT NULL;

UPDATE vw_ActiveCustomers
SET Phone = '99999999'
```

```
WHERE CustomerID = 1;
```

Real-Life Example

HR System:
Updating employee phone numbers using a filtered view instead of the full employee table.

3. How Views Simplify Complex Queries

Without View

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

With View

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

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

Benefits

- Cleaner queries
- Consistent logic
- Easier maintenance
- Better security

Use case: Call center agents viewing account summaries all day.