SQL Views – Research & Implementation Task

Part 1: Research & Documentation:

1. Types of Views in SQL Server

1.1 Standard View

- ***What is it?***

A **Standard View** (also called a regular view) is a **virtual table** created from a SELECT query.
It does **not store data** itself; it only stores the query definition. Data is retrieved from the base tables each time the view is queried.

Standard views are mainly used to:

- Simplify complex queries
- Hide sensitive columns
- Provide logical data abstraction
- Improve security and maintainability

Key Differences

Feature	Standard View
Data storage	No
Performance	Depends on base tables
Index allowed	No
DML support	Yes (with restrictions)

Real-Life Use Case

Banking system:
Create a view to show customer account summaries for call center staff without exposing sensitive data.

```
CREATE VIEW vw_CustomerAccounts AS
SELECT c.CustomerID, c.FullName, a.AccountNumber, a.Balance
FROM Customer c
JOIN Account a ON c.CustomerID = a.CustomerID;
```

Call center agents can use this view instead of joining tables manually.

Limitations & Performance Considerations

- Cannot be indexed
- Performance depends on underlying tables
- Complex views may slow queries
- Cannot use `ORDER BY` without `TOP`

1.2 Indexed View

What is it?

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

Indexed views are mainly used to:

- Improve performance of **aggregation-heavy queries**
- Speed up **frequently used summary reports**
- Reduce CPU cost for complex calculations

Key Differences

Feature	Indexed View
Data storage	Yes
Performance	Faster for aggregation
Index allowed	Yes (required)
DML impact	Slower inserts/updates

Real-Life Use Case

E-commerce system:
Calculate total sales per day for dashboards.

```
CREATE VIEW vw_DailySales

WITH SCHEMABINDING

AS

SELECT OrderDate, COUNT_BIG(*) AS TotalOrders

FROM dbo.Orders

GROUP BY OrderDate;
```

1.3 Partitioned View (Union View)

What is it?

A **Partitioned View** combines data from multiple tables (often across databases or servers) using `UNION ALL`.

Key Differences

Feature	Partitioned View
Combines tables	Yes
Cross-database	Yes
Index support	No
Best for	Large, distributed data

Real-Life Use Case

University system:

Combine student records from different campuses.

```
CREATE VIEW vw_AllStudents AS
SELECT * FROM Campus1.dbo.Student
UNION ALL
SELECT * FROM Campus2.dbo.Student;
```

Limitations & Performance Considerations

- No indexing
- Requires identical table structures
- Can be slow for large datasets
- Complex maintenance

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

Which Views Allow DML?

✓ **Standard Views** – Yes (with restrictions)
✓ **Indexed Views** – Yes (but heavy restrictions)
✕ **Partitioned Views** – Very limited

DML Restrictions on Views

You **cannot** perform DML on a view if it contains:

- `GROUP BY`
- `DISTINCT`
- `JOIN` (in most cases)
- Aggregate functions (`SUM`, `COUNT`)
- `UNION`

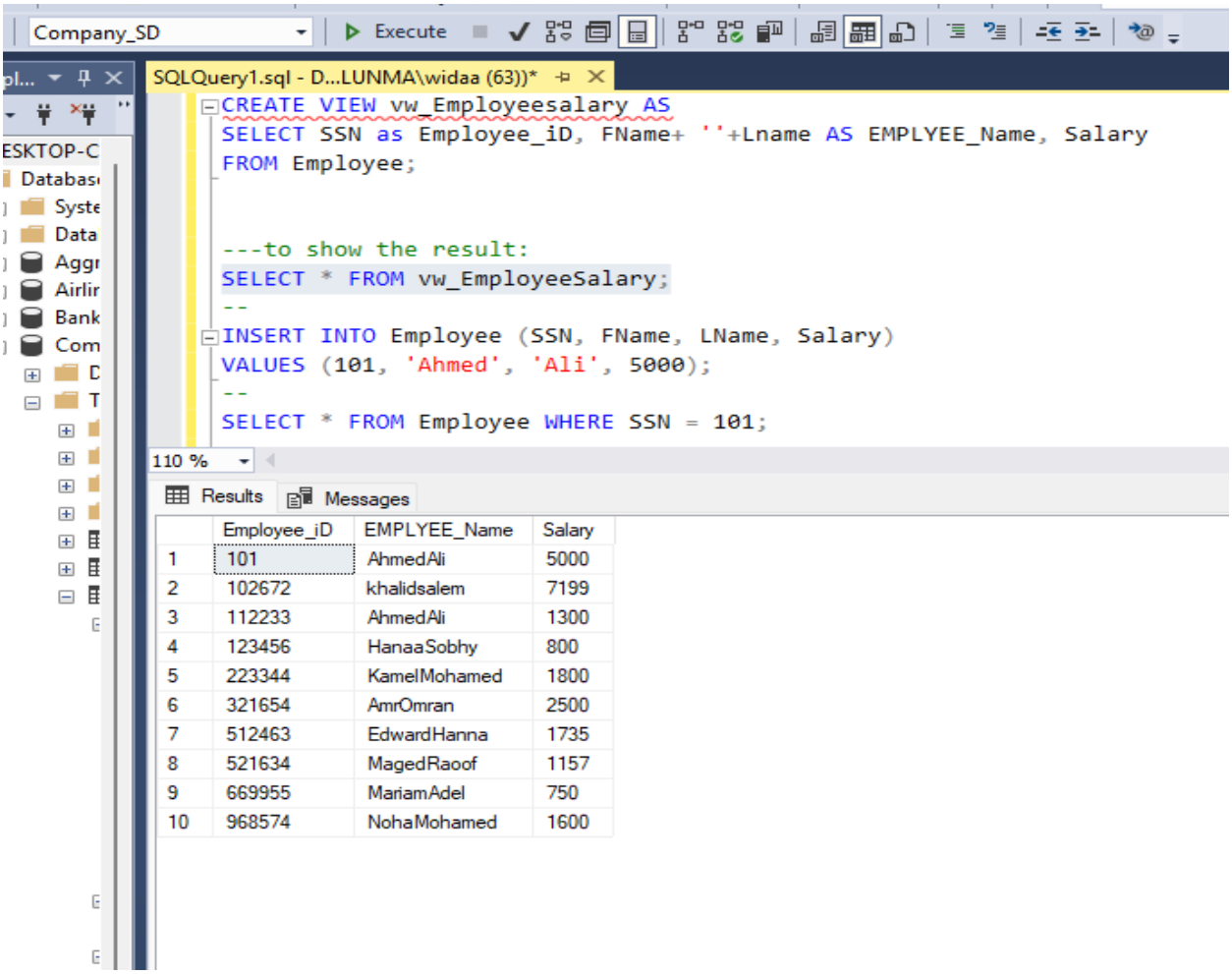
The view must reference **only one base table** for full DML support.

Real-Life Example (HR System)

Update employee salary through a view:

```
CREATE VIEW vw_EmployeeBasic AS

SELECT EmployeeID, Name, Salary

FROM Employee;

UPDATE vw_EmployeeBasic

SET Salary = Salary + 500

WHERE EmployeeID = 101;
```

✓ The update affects the base table directly.



	Employee_iD	EMPLYEE_Name	Salary
1	101	AhmedAli	5000
2	102672	khalidsalem	7199
3	112233	AhmedAli	1300
4	123456	HanaaSobhy	800
5	223344	KamelMohamed	1800
6	321654	AmrOmran	2500
7	512463	EdwardHanna	1735
8	521634	MagedRaoof	1157
9	669955	MariamAdel	750
10	968574	NohaMohamed	1600

```sql
UPDATE vw_Employeesalary
SET Salary = Salary + 500
WHERE Employee_iD = 101;

--to show the result after update:
SELECT * FROM vw_EmployeeSalary;
```

10 %

Results | Messages

	Employee_iD	EMPLYEE_Name	Salary
1	101	AhmedAli	5500
2	102672	khalidsalem	7199
3	112233	AhmedAli	1300
4	123456	HanaaSobhy	800
5	223344	KamelMohamed	1800
6	321654	AmrOmran	2500
7	512463	EdwardHanna	1735
8	521634	MagedRaoof	1157
9	669955	MariamAdel	750
10	968574	NohaMohamed	1600

3. How Views Simplify Complex Queries

Why Views Are Useful

- Hide complex JOIN logic
- Improve readability
- Ensure consistency
- Enhance security
- Reduce query repetition

Banking Example: Customer + Account

Without View (Complex Query)

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

Create View

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

Using the View (Simple Query)

```
SELECT * FROM vw_CustomerAccountSummary;
```

Real-Life Scenario

Call Center Agents

Agents frequently need account summaries.

Using a view:

- Saves time
- Reduces errors
- Improves performance
- Increases data security

Conclusion

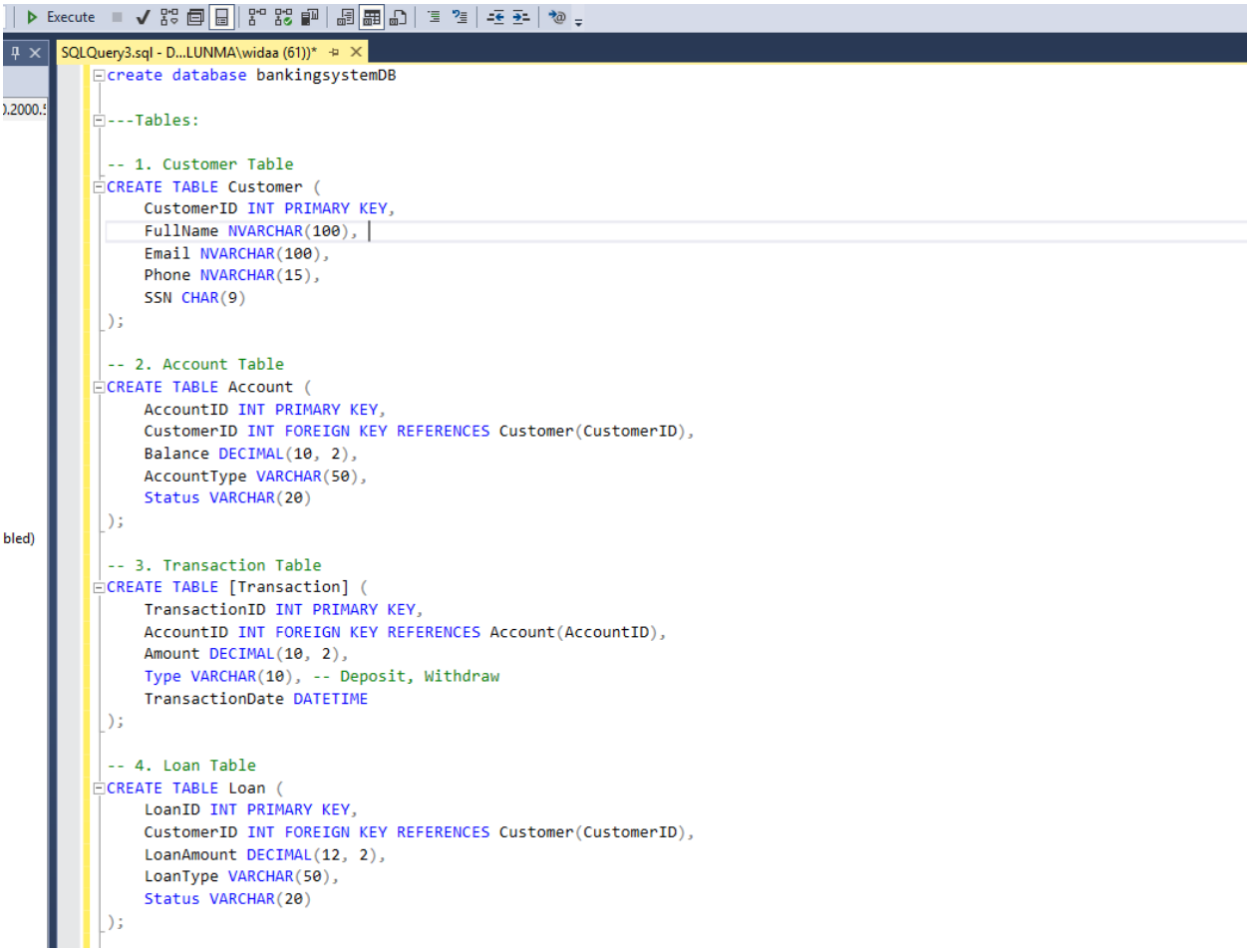
SQL Server Views are powerful tools for:

- Simplifying complex queries
- Improving security
- Enhancing performance (Indexed Views)
- Supporting data abstraction

Choosing the correct type of view depends on:

- Performance needs
- Data size
- Update requirements

Part2:


Execute
SQLQuery3.sql - D...LUNMA\widaa (61))*
create database bankingsystemDB
---Tables:
-- 1. Customer Table
CREATE TABLE Customer (
CustomerID INT PRIMARY KEY,
FullName NVARCHAR(100),
Email NVARCHAR(100),
Phone NVARCHAR(15),
SSN CHAR(9)
);
-- 2. Account Table
CREATE TABLE Account (
AccountID INT PRIMARY KEY,
CustomerID INT FOREIGN KEY REFERENCES Customer(CustomerID),
Balance DECIMAL(10, 2),
AccountType VARCHAR(50),
Status VARCHAR(20)
);
-- 3. Transaction Table
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
);
-- 4. Loan Table
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
-- Customers
INSERT INTO Customer VALUES
(1, 'Ahmed Ali', 'ahmed@gmail.com', '91234567', '123456789'),
(2, 'Mona Hassan', 'mona@gmail.com', '92345678', '987654321');

-- Accounts
INSERT INTO Account VALUES
(101, 1, 5000.00, 'Savings', 'Active'),
(102, 1, 2000.00, 'Checking', 'Active'),
(103, 2, 10000.00, 'Savings', 'Inactive');

-- Transactions
INSERT INTO [Transaction] VALUES
(1001, 101, 1000.00, 'Deposit', GETDATE()-10),
(1002, 101, 500.00, 'Withdraw', GETDATE()-5),
(1003, 102, 200.00, 'Deposit', GETDATE()-40),
(1004, 103, 1500.00, 'Deposit', GETDATE()-20);

-- Loans
INSERT INTO Loan VALUES
(201, 1, 10000.00, 'Home Loan', 'Approved'),
(202, 2, 5000.00, '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).

```sql
---Part 3: View Creation Scenarios:

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

create view Vcustomer_service as
select c.FullName as Customer_name,
       c.Phone,
       a.Status as Account_Status
from Customer c
inner join Account a
on c.CustomerID=a.CustomerID

--show the result of view
select*from Vcustomer_service
```

110 %

Results | Messages

	Customer_name	Phone	Account_Status
1	Ahmed Ali	91234567	Active
2	Ahmed Ali	91234567	Active
3	Mona Hassan	92345678	Inactive

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


create view Vfinance_department as
select AccountID,Balance,AccountType
from Account

----show the result of view
select*from Vfinance_department
```

110 %

Results | Messages

	AccountID	Balance	AccountType
1	101	5000.00	Savings
2	102	2000.00	Checking
3	103	10000.00	Savings

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

create view VLoan_officer as
select LoanId,CustomerID,LoanAmount,LoanType,Status
from Loan;

----show the result of view
select*from VLoan_officer
```

110 %

Results | Messages

	LoanId	CustomerID	LoanAmount	LoanType	Status
1	201	1	10000.00	Home Loan	Approved
2	202	2	5000.00	Car Loan	Pending

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

create view Vtransaction_summary as
select TransactionDate,AccountID,Amount
from [Transaction]
where TransactionDate >= DATEADD(DAY, -30, GETDATE());

----show the result of view
select*from Vtransaction_summary
```

110 %

Results | Messages

	TransactionDate	AccountID	Amount
1	2025-12-18 14:07:17.323	101	1000.00
2	2025-12-23 14:07:17.323	101	500.00
3	2025-12-08 14:07:17.323	103	1500.00