SQL Views - Research & Implementation Task:

1. Types of Views in SQL Server:

Type of View	What is it?	Key Differences	Real-Life Use Case	Limitations & Performance
Standard View	A virtual table based on a SELECT query from base tables.	Doesn't store data; simpler; cannot have indexes unless schema-bound.	**University System:** Combine `Students`, `Courses`, and `Enrollments` for reporting.	No physical storage; slower on large data; can't use DML if contains joins or aggregates.
Indexed View	A view with a clustered index; stores data physically.	Improves performance; strict rules; uses `SCHEMABINDING`.	**Banking System:** Fast access to aggregated balances across millions of records.	Slower DML on base tables due to maintenance; strict creation rules.
Partitioned View	Combines tables using `UNION ALL`; used for horizontal partitioning.	Spans multiple tables; supports large datasets; can be distributed.	**E-Commerce:** Combine monthly sales tables for yearly reports.	DML possible only if strict rules are followed (e.g., check constraints); can be complex and slower.

Can We Use DML on Views?
Yes, but only under certain conditions. Views can support `INSERT`, `UPDATE`, and `DELETE` if they meet specific requirements.

Types of Views That Allow DML Operations:
- standard view , Allowed **only** if:
  • View is based on a **single base table**
  • No aggregates, DISTINCT, GROUP BY, UNION, etc.
- indexed view: Requires `SCHEMABINDING`; expression must be **deterministic**; strict limitations.
- partitioned view: allows DML with conditions: Allowed only if each base table has a **CHECK constraint** to define partition ranges and **no overlapping data**.

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

Restrictions on DML in Views

Not Allowed If View Contains:

- Aggregate functions (e.g., SUM, COUNT)
- `DISTINCT` keyword
- `GROUP BY`, `HAVING` clauses
- `UNION` or `UNION ALL`
- Joins (in most update cases)
- Subqueries or computed columns
- Read-only columns or functions

one real life example where updating a view is useful:

A real-life example of updating a view is in an HR system. Suppose you create a view that shows only active employees:

```
CREATE VIEW ActiveEmployees AS
SELECT EmployeeID, Name, JobTitle
FROM Employees
WHERE Status = 'Active';
```

In this case, the HR staff can use this view to update job titles or employee names directly, without needing access to the full `Employees` table. This improves security by hiding unnecessary fields (like salary or inactive employees), while still allowing safe and limited updates.

2. How Can Views Simplify Complex Queries?
Views in SQL Server are incredibly useful for simplifying JOIN-heavy or repetitive queries. Instead of writing long SQL statements with multiple JOINs every time you need the same data, you can encapsulate the logic in a view and reference it like a regular table. This reduces code duplication, improves readability, and makes maintenance easier.

Create an example view that joins at least two of your banking tables, such as:
assume you have two tables:

Customer + Account view :

1. `Customer(CustomerID, Name, Phone)`
2. `Account(AccountID, CustomerID, Balance, AccountType)`

Create a View:

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

Using the View:

```
SELECT Name, Phone, AccountID, Balance
FROM CustomerAccountView
WHERE Balance > 5000;
```

Account + Transaction View:

Create View:

```
CREATE VIEW AccountTransactionView AS
SELECT a.AccountID, a.Balance, t.TransactionID, t.Amount, t.Date
FROM Account a
JOIN Transaction t ON a.AccountID = t.AccountID;
```

Query the View:

```
SELECT AccountID, TransactionID, Amount, Date
FROM AccountTransactionView
WHERE Amount > 1000;
```