# Types of Views in SQL Server:

## Standard View:

## What is it?

* A Standard View is a virtual table defined by a SELECT query. It does not store data itself but displays data stored in base tables.

## Key differences from other view types.

* Not materialized; always reads from the base tables.
* Cannot have indexes directly (except with SCHEMABINDING and converting to an Indexed View).
* Simple and easy to use for abstraction

## Real-life use cases

* A bank may use a view to show customer account details by joining Customers and Accounts tables for reporting.

### Limitations and performance considerations

* Always executes underlying query, which can affect performance for complex joins.
* Not suitable for high-performance aggregation queries.

## Indexed View:

## What is it?

* A view that has a clustered index built on it. Data is physically stored and maintained by SQL Server, which improves query performance.

## Key differences from other view types.

* Materialized (stores data).
* Can significantly improve performance.
* Requires **SCHEMABINDING**, **deterministic functions**, and other strict rules.

Real-life use cases

* An e-commerce system might use an indexed view to store aggregated sales data for daily reporting, reducing processing time.  
  Limitations and performance considerations

## Partitioned View (Union View):

## What is it?

* A view that unites multiple tables (usually with the same schema) via UNION ALL. Often used for data partitioning across tables.

## Key differences from other view types.

* Can span across multiple servers (Distributed Partitioned Views).
* Used for horizontal partitioning.

Real-life use cases

* A university storing student records in yearly tables (e.g., Students\_2023, Students\_2024) can create a partitioned view to query all years seamlessly.

### Limitations and performance considerations

* Indexed partitioned views have strict rules.
* Can be slow if not indexed or properly filtered using CHECK constraints.

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

* **Yes**, but with limitations.

## Which types of views allow DML operations?

* **Standard Views:** Yes, if based on a single table and meets certain criteria.
* **Indexed Views:** Yes, but must meet specific rules.
* **Partitioned Views:** Yes, if all participating tables follow partitioning rules (e.g., CHECK constraints and updatable columns).

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

| **Restriction** | **Explanation** |
| --- | --- |
| Multi-table views | Cannot be directly updated. |
| Aggregated/Grouped views | Not updatable. |
| Views with joins | DML allowed only under specific conditions (INSTEAD OF triggers can help). |
| Computed columns or functions | Non-deterministic expressions block DML. |

## real-life example where updating a view is useful

* An HR system might use a view vw\_EmployeeDetails that pulls data from Employees and Departments. If only Employees is updated via the view (e.g., salary changes), it simplifies app development while maintaining data encapsulation.

# How Can Views Simplify Complex Queries?

Views Abstract JOIN-heavy Logic for example:

* **Before View:** Developers must repeatedly write long queries with joins and filters.
* **After View:** A view encapsulates that logic. Users just select from the view.

## Example view that joins at least two banking tables

CREATE VIEW vw\_CustomerAccountSummary AS

SELECT

c.CustomerID, c.Name, c.Email,

a.AccountID, a.AccountType, a.Balance

FROM

Customers c

JOIN

Accounts a ON c.CustomerID = a.CustomerID;