Part 1: Research & Documentation

Each team must research and write a short report that includes the following:
1. Types of Views in SQL Server:

o Standard View

A **Standard View** is a virtual table created by a SQL query.
It does not store data physically; instead, it shows data dynamically from one or more tables when queried.

Key points:

- Stores only the SQL query, not the data
- Used for simplifying complex queries
- Improves security by restricting access to specific columns or rows

Example use:
Showing student names and course titles without exposing sensitive data.

o Indexed View

An **Indexed View** is a view that stores data physically by creating a **unique clustered index** on it.
This improves performance for complex queries that use aggregations.

Key points:

- Data is stored physically
- Improves performance for large datasets
- Requires `SCHEMABINDING`
- Uses more storage space

Example use:
Frequently queried reports such as total sales or average ratings.

o Partitioned View (Union View)

A **Partitioned View** combines data from multiple tables using `UNION ALL`.
Each table usually represents a partition of data (e.g., by year or region).

Key points:

- Uses `UNION ALL`
- Tables must have the same structure
- Useful for working with very large datasets
- Improves manageability and scalability

Example use:
Combining yearly enrollment tables into one logical view.

2. For each type, answer:

o What is it?
o Key differences from other view types.
o Real-life use cases (example from banking, e-commerce, university system…)
o Limitations and performance considerations.

1. Standard View

What is it?

A **Standard View** is a virtual table created using a SQL query.
It does not store data physically; the data is retrieved from the base tables each time the view is queried.

Key differences from other view types

- Does not store data physically
- Does not have indexes
- Simplest type of view

- Slower than Indexed Views for large datasets

Real-life use cases

- **University system:**
  Display student names and enrolled courses without exposing sensitive information
- **Banking:**
  Show transaction history without showing full account balances
- **E-commerce:**
  Display product names and prices to customers

Limitations & performance considerations

- Performance may be slow with large data volumes
- Query is executed every time the view is accessed
- Not suitable for heavy reporting

2. Indexed View

What is it?

An **Indexed View** is a view that stores data physically by creating a clustered index on it.
It is mainly used to improve performance for complex queries with aggregations.

Key differences from other view types

- Stores data physically

- Has indexes
- Faster than Standard Views
- Requires `SCHEMABINDING`

Real-life use cases

- **Banking:**
  Calculating daily transaction totals per branch
- **E-commerce:**
  Daily or monthly sales reports
- **University system:**
  Calculating average grades or total student counts

Limitations & performance considerations

- Uses additional storage space
- Slower insert, update, and delete operations
- Has strict creation requirements

3. Partitioned View (Union View)

What is it?

A **Partitioned View** combines data from multiple tables using `UNION ALL`.
Each table usually represents a partition of the data, such as by year or region.

Key differences from other view types

- Combines multiple tables

- Uses `UNION ALL`
- Does not store data physically
- Suitable for very large datasets

Real-life use cases

- **University system:**
  Combining yearly enrollment tables into one logical view
- **Banking:**
  Combining transaction data from multiple branches
- **E-commerce:**
  Combining orders from different regions

Limitations & performance considerations

- Performance depends on proper table partitioning
- Can be complex to design and maintain
- Requires consistent table structures

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

• Do some research and explain:

o **Which types of views allow DML operations?**

In general, **DML operations cannot be applied directly to views** because views do not store data physically.

- **Standard Views**
  DML operations are **usually not allowed**.
  Only **simple views based on a single table** may allow `INSERT`, `UPDATE`, or `DELETE`, and any change affects the **underlying base table**, not the view itself.

- **Indexed Views**
  DML operations **cannot be performed directly** on indexed views.
  Any modification must be done on the base tables.
- **Partitioned Views (Union Views)**
  DML operations are **not allowed** because they combine multiple tables using `UNION ALL`.

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

DML operations are **not allowed** on a view if it contains:

- `JOIN`
- `GROUP BY`
- Aggregate functions (`AVG`, `SUM`, `COUNT`, etc.)
- `DISTINCT`
- Computed columns
- `UNION` or `UNION ALL`

Additional limitations:

- All NOT NULL columns must be included in the view for `INSERT`
- The view must reference only one base table
- The view must not use subqueries in the SELECT list

o Give at least one real-life example where updating a view is useful (e.g., HR system, e-commerce orders, etc.)
Tip: Try to test this in SQL Server if possible using a simple view.

HR System Example

In an HR system, a view can be used to display employee contact information without exposing sensitive data such as salaries.

```
CREATE VIEW EmployeeContactView AS

SELECT EmployeeID, FullName, Email
```

FROM Employees;

2. How Can Views Simplify Complex Queries?

Explain how a View can help simplify JOIN-heavy queries.

Views help simplify complex SQL queries by storing frequently used JOIN logic in one place.
Instead of writing long and complicated queries every time, users can query the view as if it were a table.
This reduces repetition, minimizes errors, and makes queries easier to understand and maintain.

Views are especially useful when multiple teams need the same data, such as call center agents who frequently access customer account summaries without needing to understand complex SQL JOINs.

Instead of writing the same long SQL query every time, users can simply select data from the view.

- **Create an example view that joins at least two of your banking tables, such as:**
  - **Customer + Account**

SQLQuery1.sql - m...MALAK\malak (64))*

```sql
--Example View: Customer + Account--
CREATE VIEW CustomerAccountView AS
SELECT
    c.customer_id,
    c.name AS CustomerName,
    c.address AS CustomerAddress,
    a.account_no,
    a.account_type,
    a.balance,
    a.date_created
FROM Customer c
JOIN Account a
    ON c.customer_id = a.customer_id;
SELECT *
FROM CustomerAccountView;
```

140 %

Results | Messages

	customer_id	CustomerName	CustomerAddress	account_no	account_type	balance	date_created
1	101	Ahmed Al-Balushi	Muscat	3001	Savings	5000.00	2022-01-10
2	101	Ahmed Al-Balushi	Muscat	3002	Checking	2625.00	2023-03-15
3	102	Sara Al-Hinai	Salalah	3003	Savings	8000.00	2021-07-20
4	103	Malak Al-Sinani	Muscat	3004	Savings	12000.00	2025-12-16

o Account + Transaction

SQLQuery1.sql - m...MALAK\malak (64))*

```sql
--Example View: Account + Transaction_Table--
CREATE VIEW AccountTransactionView AS
SELECT
    a.account_no,
    a.account_type,
    a.balance,
    t.transaction_id,
    t.transaction_date,
    t.amount,
    t.transaction_type
FROM Account a
JOIN Transaction_Table t
    ON a.account_no = t.account_no;
SELECT *
FROM AccountTransactionView;
```

140 %

Results | Messages

	account_no	account_type	balance	transaction_id	transaction_date	amount	transaction_type
1	3001	Savings	5000.00	4001	2024-01-05	500.00	Deposit
2	3001	Savings	5000.00	4002	2024-02-01	200.00	Withdrawal
3	3003	Savings	8000.00	4003	2024-03-10	1000.00	Transfer

• **Show how using the view reduces the need to repeat long queries.**

SQLQuery1.sql - m...MALAK\malak (64))*

```sql
CREATE VIEW CustomerAccountView AS
SELECT
    c.customer_id,
    c.name AS CustomerName,
    c.address AS CustomerAddress,
    a.account_no,
    a.account_type,
    a.balance,
    a.date_created
FROM Customer c
JOIN Account a
    ON c.customer_id = a.customer_id;
SELECT *
FROM CustomerAccountView;
```

140 %

Results | Messages

	customer_id	CustomerName	CustomerAddress	account_no	account_type	balance	date_created
1	101	Ahmed Al-Balushi	Muscat	3001	Savings	5000.00	2022-01-10
2	101	Ahmed Al-Balushi	Muscat	3002	Checking	2625.00	2023-03-15
3	102	Sara Al-Hinai	Salalah	3003	Savings	8000.00	2021-07-20
4	103	Malak Al-Sinani	Muscat	3004	Savings	12000.00	2025-12-16

SQLQuery1.sql - m...MALAK\malak (64))*

```sql
CREATE VIEW AccountTransactionView AS
SELECT
    a.account_no,
    a.account_type,
    a.balance,
    t.transaction_id,
    t.transaction_date,
    t.amount,
    t.transaction_type
FROM Account a
JOIN Transaction_Table t
    ON a.account_no = t.account_no;

SELECT *
FROM AccountTransactionView;
```

140 %

Results | Messages

	account_no	account_type	balance	transaction_id	transaction_date	amount	transaction_type
1	3001	Savings	5000.00	4001	2024-01-05	500.00	Deposit
2	3001	Savings	5000.00	4002	2024-02-01	200.00	Withdrawal
3	3003	Savings	8000.00	4003	2024-03-10	1000.00	Transfer