Views

Creating views

What is a view?

SQL queries that return tables/result sets can be assigned to a **virtual table**. This is what's referred to as a view. These tables have the following characteristics.

Dynamic	Views, unlike conventional tables, **do not store data**. They **display information from other tables**. The data in a view are always fresh and up to date since the view refreshes when the tables used to generate the view change.
Data abstraction	Views can **hide the complex data**. For example, you could join multiple tables and present only a selection of columns or rows where a column has a specific value. Views can also **restrict access to data** for different users (for example, customers, employees, suppliers, etc.).
Read-only or updatable	Some views are **read-only**, meaning you **can't modify the data** using the view. Others can be updatable, allowing `INSERT`, `UPDATE`, and `DELETE` operations on the view.

Why do we need views?

Most organizations handle sensitive data such as personal, client, and employee information. Different users can be **authorized** to use or view different parts of the data and system. Here are a few examples:

Regulatory compliance:
Ensures only **relevant** data are accessible, meeting industry **compliance** regulations.

```
CREATE VIEW CompliantView AS
SELECT CustomerID, Name, SalesDate
FROM Sales
WHERE DataCompFlag = 'Compliant';
```

CustomerID	Name	SalesDate
101	Jane Doe	2023-01-15
102	Jane Smith	2023-01-20
103	Alice Lee	2023-01-22

Cross-departmental reporting:
Caters to each department's data requirements, **tailoring their reports**.

```
CREATE VIEW FinReportView AS
SELECT EmployeeID, Salary, Tax
FROM EmployeeData
WHERE Department = 'Finance';
```

EmployeeID	Salary	Tax
E001	$5,000	$500
E002	$5,500	$480
E003	$4,800	$520

Third-party integration:
Provides data interfaces for **external stakeholders** without revealing internal data.

```
CREATE VIEW VendorView AS
SELECT ProductID, ProductName, Stock
FROM Inventory
WHERE VendorAccess = 'Allowed';
```

ProductID	ProductName	Stock
P001	Laptop	50
P002	Mobile Phone	100
P003	Mouse	200

Types of views

While views can be broadly categorized as simple or complex, we will focus on **three specific types derived** from their **DQL query characteristics**.

Types of views

Lookup view

- Primarily used to fetch **static or reference data.**
- For example, providing a list of country names and their estimated population or GDP.

Join view

- Combines data from multiple tables into one dataset.
- Useful for **representing relationships** between tables without requiring complex joins.

Aggregating view

- Used to **display summarized or aggregated data.**
- Often involves `GROUP BY` clauses and aggregated functions like `SUM`, `AVG`, `COUNT`, etc.

Lookup view example

> Let's create a lookup view that provides a **list** of **country names** and their **estimated population** from the "Access_to_Basic_Services" table.

```
CREATE VIEW Country_Lookup AS
SELECT
    Country_name,
    Est_population_in_millions,
    Time_period
FROM
    united_nations.Access_to_Basic_Services
```

This view, **Country_Lookup**, retrieves data from the Access_to_Basic_Services table.

It serves as a reference for **quick access** to country names and their estimated populations.

Country_name	Est_population_in_millions	Time_ period
Kazakhstan	18.037	2018
Kazakhstan	18.513	2019
Kazakhstan	18.756	2020
Tajikistan	12.037	2018
Tajikistan	12.001	2019

Join view example: Source tables

Consider these two tables within the `United Nations` database: **`Access_to_Basic_Services`** and **`Regions`**, which will help us to **add geographical data** to our analysis.

Table: **`Access_to_Basic_Services`**

Country_name	Sub_region	…	…
Sri Lanka	Central Asia	…	…
China	Eastern Asia	…	…
Canada	Northern America	…	…

Table: **`Regions`**

Region	Sub_ region	…	…
Central and Southern Asia	Central Asia	…	…
Eastern and South-Eastern Asia	Eastern Asia	…	…
Europe and Northern America	Northern America	…	…

Join view example

Let's create a join view that helps us to determine the **regions for each country** in our dataset. This can be done by **joining tables** `Access_to_Basic_Services` and `Regions`.

```
CREATE VIEW Region_Country_Join AS
SELECT
    A.Country_name,
    A.Sub_region,
    B.Region
FROM
    united_nations.Access_to_Basic_Services
AS A
INNER JOIN
    united_nations.Regions
AS B
ON A.Sub_region = B.Sub_region
```

This view, `Region_Country_Join`, retrieves data from the `Access_to_Basic_Services` and `Regions` tables. It simplifies **querying relationships** between countries and regions.

Country_name	Region	Sub_region
Sri Lanka	Central and Southern Asia	Central Asia
China	Eastern and South-Eastern Asia	Eastern Asia
Canada	Europe and Northern America	Northern America

Aggregating view example

Let's create an aggregating view that **displays summarized data** about regions, including the total population and average GDP.

```
CREATE VIEW Region_Aggregation AS
SELECT
    Region,
    SUM(Est_population_in_millions)
        AS Total_Population_in_millions,
    AVG(Est_gdp_in_billions)
        AS Avg_GDP_in_billions
FROM
    united_nations.Access_to_Basic_Services
```

This view, `Region_Aggregation`, retrieves data from the `Access_to_Basic_Services` and Regions tables. It **provides insights** into the total population and average GDP for each region.

Region	Total_Population_in_millions	Avg_GDP_in_billions
Central and Southern Asia	11371.04644	302.7461
Eastern and South-Eastern Asia	12477.7617	1703.9799
Europe and Northern America	221.7836	550.3311

Advantages of using views

Data security

Views can **restrict access** to specific columns and rows in a table, ensuring users **only see authorized data**.

Consistency

Views **offer a consistent snapshot** of data, which can be very useful for reporting and analysis.

Simplify querying

Views **combine complex SQL queries**, allowing users to retrieve data without understanding or dealing with the underlying complexity.

Reduced network traffic

Instead of sending multiple complex queries over a network, **applications can query a view**, potentially reducing the amount of data transferred.

Data abstraction

Data is presented in a more understandable format, **abstracting the underlying table's complexity** and offering a simplified perspective of the data.

Disadvantages of using views

Performance overhead

Views built from multiple tables and complex conditions introduce **performance overhead** because they are run every time the view is queried.

Maintenance complexity

As the database schema evolves, views might need updates and if they're not managed properly, views can become **challenging to maintain**.

Potential for misuse

Inexperienced users might treat views as physical tables, and over-reliance on views can lead to a **cluttered database with redundant views**.

Debugging

Debugging and finding the source of the issue with views, especially nested views, can be more **challenging** than with regular tables.

Updatability

Depending on the database system and the complexity of the view, we **might not be able to perform** `INSERT`, `UPDATE`, or `DELETE` queries directly on the view.

Best practices when creating views

01. Be specific with columns

Instead of using **SELECT *, specify the exact columns** you need in the view. This might also improve performance.

02. Document views

Always **provide comments or documentation** for views, explaining their purpose, the data they represent, and any other relevant details.

03. Avoid nested views

While it's possible to create a view based on another view, **excessive nesting can lead to performance issues** and increased complexity.

04. Use schema binding

If it's supported, use **schema binding to bind the view to the schema** of the tables used to generate it. This ensures stability and integrity.

05. Test views thoroughly

Before deploying a view in a production environment, **test it** to ensure it returns the expected results and performs well.

06. Stay updated with changes

If there are changes to the schema of the tables used to create the views, ensure that the **views are updated accordingly**.