 ****

**View**

**What is view in MYSQL**?

In simple words, a **view** in a database is like a virtual table. A view is just a query that pretends to be a table. It doesn't store data itself but displays data stored in other tables. You can think of it as a saved query that you can use like a regular table.

It is not stored physically (except for materialized views) but is derived from a query that retrieves data from one or more base tables. Views are used to simplify complex queries, enforce security by restricting access to certain data, or present data in a specific format.

Here’s a basic example to illustrate:

* Imagine you have a table called employees with lots of information about your company's employees.
* You only want to see the names and salaries of the employees without all the other details.

You can create a view called employee\_salaries that shows only the names and salaries. When you query the employee\_salaries view, it fetches and displays just the names and salaries from the employees table, even though the view itself doesn't store that data.

Views are useful because they can simplify complex queries, improve security by limiting access to specific data, and help present data in a more understandable way.

Advantages of Views:

1. **Abstraction:**  Maybe you need to get data from many different tables to create a specific report. Instead of writing a complex query every time, you can create a view that combines all the needed data. Then, you can simply query this view as if the data is already neatly combined in one place.
2. **Security:** You don't want to give any more access to people than they need. By creating a view and only giving access to the view instead of the underlying tables, the user will only be able to query what you have specifically selected for in the view.
3. **Efficiency:** If you know that your users are going to have to query a specific domain of data that will require several different tables all joined together in a specific way, it's more efficient (from a development perspective) and consistent to just have a single view they can all use.

**Types of view:**

1. **Simple View:**
2. Based on multiple tables.
3. Includes JOINs, subqueries, and sometimes aggregate functions.
4. **Complex View**:
   1. Based on multiple tables.
   2. Includes JOINs, subqueries, and sometimes aggregate functions.
5. **Materialized View**:
6. Stores query results physically in the database.
7. Improves performance by avoiding re-executing the query every time the view is accessed.
8. Requires periodic refresh to keep the data up to date.

**1. Updatable Views**

* If a view is **updatable**, changes made in the view (such as INSERT, UPDATE, or DELETE operations) will directly affect the data in the underlying table.
* However, for a view to be updatable, it must satisfy certain conditions:
  + The view should be based on a single table (or simple joins).
  + The view should not contain aggregate functions (SUM, COUNT, etc.), DISTINCT, GROUP BY, or HAVING.
  + It should not have subqueries or complex expressions.

**2. Non-Updatable Views**

* Views involving multiple tables, like joins, or complex queries are typically **non-updatable**. This means any attempt to modify the data in the view will not update the underlying tables, and in many cases, the database will not allow the modification at all.

**Explanation of Why It’s Non-Updatable:**

1. **Aggregate Function (COUNT())**: The view is performing a count of students, and aggregate functions like COUNT(), SUM(), AVG(), etc., make views non-updatable because there’s no straightforward way to map the update back to the original rows in the base table.
2. **GROUP BY Clause**: When a view uses GROUP BY, it groups data into summarized rows, and thus it becomes impossible to determine how to propagate changes back to the original data in the underlying tables.

**Key Factors That Make Views Non-Updatable:**

* Aggregate functions (COUNT(), SUM(), etc.).
* Use of GROUP BY, HAVING, or DISTINCT.
* Views based on complex joins where it's ambiguous how to map the update back to the original tables.
* Subqueries in the SELECT statement.
* Use of UNION or UNION ALL.