**Difference Between CTE,View And Temporary Table**

Temporary Table:-

A temporary table in **SQL** is an important tool for maintaining intermediate results during **query**execution. They help store temporary data without affecting the underlying permanent tables.

In this article, we’ll explore temporary tables in **SQL**, their types (local vs. global), and how to use them effectively in your database operations.

A temporary table in SQL is a **special type of table** that is created and stored in the system’s temporary database (such as TempDB in SQL Server). This table is primarily used to store and generate important mediation results when **executing**a query, stored procedure, or session.

Temporary tables are **automatically deleted** when the session or transaction that created them ends, making them perfect for temporary or intermediate data storage. They are particularly useful in situations where you need to perform **calculations**or data transformations without changing the permanent database structure.

View:-

Views in SQL are a type of **virtual table** that simplifies how users interact with data across one or more tables. Unlike**traditional tables**, a view in **SQL** does not store data on disk; instead, it dynamically retrieves data based on a pre-defined query each time it’s accessed.

SQL views are particularly useful for managing complex queries, enhancing security, and presenting data in a simplified format. In this guide, we will cover the **SQL** create view statement, updating and deleting views, and using the WITH CHECK OPTION clause.

A view in SQL is a saved SQL query that acts as a virtual table. It can fetch data from one or more tables and present it in a customized format, allowing developers to:

* **Simplify Complex Queries:** Encapsulate complex joins and conditions into a single object.
* **Enhance Security:**Restrict access to specific columns or rows.
* **Present Data Flexibly:** Provide tailored data views for different users.

Types of View:-

* **Simple View:** A view based on only a single table, which doesn't contain GROUP BY clause and any functions.
* **Complex View:** A view based on multiple tables, which contain GROUP BY clause and functions.
* **Inline View:** A view based on a subquery in FROM Clause, that subquery creates a temporary table and simplifies the complex query.
* **Materialized View:** A view that stores the definition as well as data. It creates replicas of data by storing it physically.

Difference Between Simple View And Materialized View :-

| **Feature** | **View** | **Materialized View** |
| --- | --- | --- |
| **Definition** | A **view** is a virtual table created from a query, and it doesn’t store data physically. | A **materialized view** stores the results of a query physically in the database for faster retrieval. |
| **Data Storage** | Only the query expression is stored; the result set is generated dynamically when the view is accessed. | Query results are stored physically in the database, consuming additional storage space. |
| **Performance** | Slower for complex queries since the result set is computed dynamically on each access. | Faster as results are precomputed and stored, reducing computation time. |
| **Update Behavior** | Automatically reflects changes in the underlying tables since data is retrieved dynamically. | Needs manual or automatic refresh to update the stored data when underlying tables change. |
| **Storage Cost** | No additional storage cost since data is not physically stored. | Requires extra storage as it saves query results. |
| **Maintenance Cost** | No maintenance cost, as views are dynamically updated with no stored data. | Involves maintenance cost due to periodic refreshes to keep data synchronized with base tables. |
| **SQL Standards** | Fully standardized and supported by all major database systems. | Not fully standardized; support and implementation vary across database systems. |
| **Use Cases** | Best for scenarios where data is accessed infrequently and requires up-to-date values. | Ideal for frequently accessed data where performance is critical, such as reporting and analytics. |

CTE In SQL :-

In SQL, a **Common Table Expression (CTE)** is an essential tool for simplifying **complex queries** and making them more readable. By defining temporary result sets that can be referenced multiple times, a **CTE in SQL** allows developers to break down **complicated logic** into manageable parts. CTEs help with **hierarchical data representation**, improve code reusability, and simplify maintenance.

In this article, we will explain **SQL CTEs**, their purpose, **syntax**, and **practical applications** with examples. We will also explore **recursive CTEs**, their structure, and when to use them. By the end, we will know to use the **CTEs in SQL queries** for better **performance**and **readability**.

A **Common Table Expression (CTE)** in [SQL](https://www.geeksforgeeks.org/what-is-sql/)is a temporary result set that is defined and used within the execution scope of a **SELECT**, **INSERT**, **UPDATE**, or **DELETE** statement. CTEs are designed to make queries easier to read and maintain by allowing complex queries to be broken into **modular components**.

CTEs are defined using the[WITH clause](https://www.geeksforgeeks.org/sql-with-clause/)and can be referenced multiple times within the main SQL query. This makes CTEs a great alternative to **subqueries**, especially in cases where we need to perform the same operation multiple times or create **recursive queries**.