## What are views?

In SQL, a view is a virtual table that provides a way to present data from one or more tables in a specific way without actually storing the data. Views are created by defining a SELECT statement as a named query and then using this query to create the view. Once a view is created, it can be used just like a table in SQL queries, but any changes made to the underlying tables will be reflected in the view.

Here are some important things to note about views in SQL:

1. Views can simplify complex queries by abstracting away the details of how the data is stored.
2. Views can be used to restrict access to certain columns or rows of a table, making it possible to provide a simplified interface for end-users.
3. Views can be used to aggregate data from multiple tables, making it easier to answer complex business questions.
4. Views do not store any data themselves, they simply provide a window into the data stored in the underlying tables.
5. Views can be used to encapsulate complex joins, making it easier to maintain the SQL code over time.

Here is an example of creating a view in SQL:

*CREATE VIEW EmployeeSales AS*

*SELECT E.Name, SUM(S.Amount) as TotalSales*

*FROM Employee E*

*JOIN Sales S ON E.EmployeeID = S.EmployeeID*

*GROUP BY E.Name;*

This creates a view called "EmployeeSales" that shows the total sales for each employee. This view can then be queried like a regular table:

*SELECT \* FROM EmployeeSales WHERE TotalSales > 10000;*

This query would return all employees with total sales greater than 10,000.

***-- VIEW?***

***create view emp\_vu as***

***select \* from employees***

***where employeenumber>1100;***

***select \* from emp\_vu where firstname like 'A%';***

## What is Indexing?

Indexing in SQL is a technique used to speed up database queries by creating a data structure that allows for faster retrieval of data. An index is a data structure that is associated with a table or a view in a database, and it contains a subset of the data in that table or view in a sorted order, along with pointers to the original rows in the table or view.

When a query is executed on a table or a view that has an index, the database engine can use the index to quickly find the relevant rows, rather than scanning the entire table or view. This can greatly improve query performance, especially for large tables with many rows.

Some common uses of indexing in SQL include:

1. Improving query performance: By creating indexes on columns that are frequently queried, the database engine can quickly find the relevant rows, resulting in faster query execution times.
2. Enforcing uniqueness: Indexes can be used to enforce uniqueness constraints on columns in a table, preventing duplicate values from being inserted.
3. Join optimization: Indexes can be used to optimize queries that involve joining multiple tables, by allowing the database engine to quickly find matching rows in each table.
4. Sorting: Indexes can be used to speed up queries that involve sorting large result sets, by pre-sorting the data in the index.
5. Full-text search: Indexes can be used to support full-text search queries, by creating indexes on columns that contain text data.

Overall, indexing is a powerful tool for optimizing database performance and improving query response times. However, creating too many indexes or creating indexes on the wrong columns can also have a negative impact on performance, so it's important to carefully consider the indexing strategy for a database based on the specific use cases and query patterns.

## Example :

Here's an example of creating a simple index on a single column in a table:

CREATE INDEX idx\_last\_name ON employees (last\_name);

In this example, the index is named "idx\_last\_name" and is created on the "last\_name" column in the "employees" table. The default order for the index is ascending, but you can specify "ASC" or "DESC" after the column name to control the order.

You can also create indexes on multiple columns, like this:

CREATE INDEX idx\_full\_name ON employees (last\_name, first\_name);

In this example, the index is named "idx\_full\_name" and is created on the "last\_name" and "first\_name" columns in the "employees" table. When querying the table, the index can be used to quickly find rows that match a particular last name and first name combination.

Note that you can also create unique indexes by adding the "UNIQUE" keyword before the "INDEX" keyword. This ensures that each value in the indexed columns is unique, and prevents duplicate values from being inserted. For example:

CREATE UNIQUE INDEX idx\_employee\_id ON employees (employee\_id);

In this example, the index is named "idx\_employee\_id" and is created on the "employee\_id" column in the "employees" table, with the requirement that each value in the column must be unique.