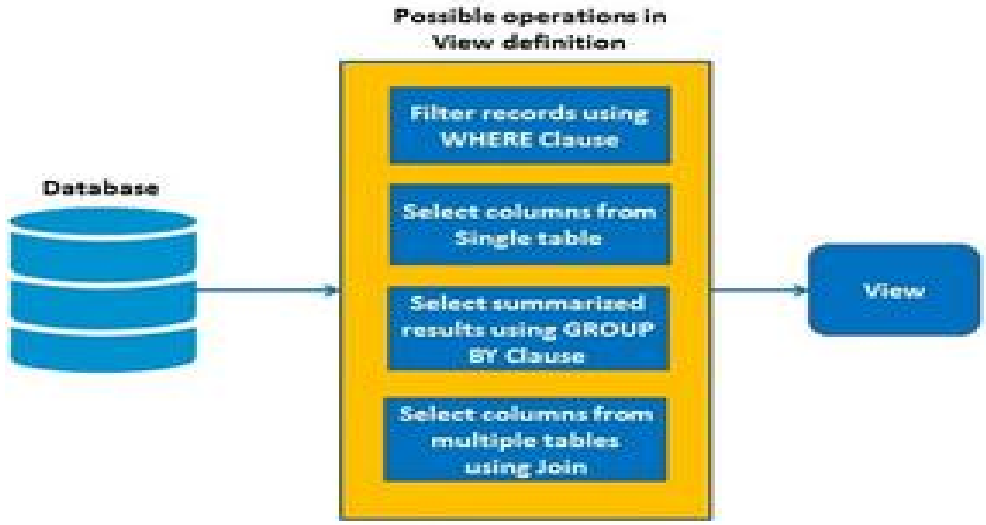
MySQL View

- The View is a **virtual table** created by a query by joining one or more tables. It is operated similarly to the base table but does not contain any data of its own.
- The View and table have one main difference that the views are definitions built on top of other tables (or views).
- If any changes occur in the underlying table, the same changes reflected in the View also.
- The view is a query stored in the data dictionary, on which the user can query just like they do on tables. It does not use the physical memory, only the query is stored in the data dictionary.
- It is computed dynamically, whenever the user performs any query on it. Changes made at any point in view are reflected in the actual base table.

The view has primarily two purposes:

- Simplify the complex SQL queries.
- Provide restriction to users from accessing sensitive data.



- We can create a new view by using the **CREATE VIEW** and **SELECT** statement. SELECT statements are used to take data from the source table to make a VIEW.

Syntax

Following is the syntax to create a view in MySQL:

```
1. CREATE [OR REPLACE] VIEW view_name AS
2. SELECT columns
3. FROM tables
4. [WHERE conditions];
```

Parameters:

The view syntax contains the following parameters:

OR REPLACE: It is optional. It is used when a VIEW already exists. If you do not specify this clause and the VIEW already exists, the CREATE VIEW statement will return an error.

view_name: It specifies the name of the VIEW that you want to create in MySQL.

WHERE conditions: It is also optional. It specifies the conditions that must be met for the records to be included in the VIEW.

Example :

```
create or replace view new_employee as
select empid ,empname,age,phoneno
from employee;
```

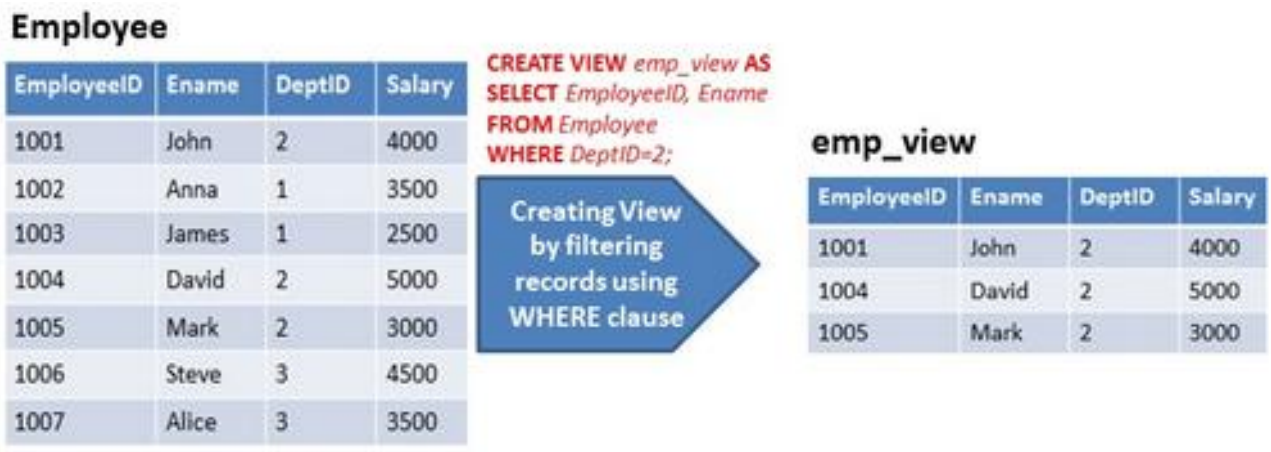
or

```
create or replace view new_employee as
select empname,age,projectname
from employee,project;
```

Types of Views

- **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.

Simple View



Complex View

Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

```
CREATE VIEW emp_view AS
SELECT DeptID, AVG(Salary)
FROM Employee
GROUP BY DeptID;
```

Create View of grouped records on Employee table

emp_view

DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00

Features	Simple Views	Complex Views
No. of tables	One	One or More
Containing Functions	No	Yes
Contain Group of data	No	Yes
DML through view	Yes	Not Allowed
Features	Simple Views	Complex Views

Inline View

An inline view is a SELECT statement in the FROM-clause of another SELECT statement to create a **temporary table** that could be referenced by the SELECT statement. Inline views are utilized for writing complex SQL queries

```
SELECT column_names, ...
FROM (subquery)
WHERE ROWNUM<= N;
```

Materialized View

Materialized view replicates the retrieved data physically. This replicated data can be reused without executing the view again.

Comparison Between View and Materialized View

View	Materialized View
View is a logical structure of the table which will be used to retrieve data from one or more table.	Materialized views are also logical structure but data is physically stored in database.
Data access is slower compared to materialized views	Data access is faster compared to simpler view because data is directly accessed from physical location
Views are generally used to restrict data from database	Materialized Views are used in Data Warehousing.

MySQL Drop VIEW

We can drop the existing VIEW by using the **DROP VIEW** statement.

Syntax:

The following is the syntax used to delete the view:

DROP VIEW [IF EXISTS] view_name;