**View is just a representative of Underlying SQL Query.**

**View does not store any data**

**View is like a Virtual table**

**We used view: 1. Security, 2. Simplify the Complex SQL Query**

**“Views in MySQL”**, which act as **virtual tables,** and understand its advantages, as well as the syntax of Views for creation, updation, and deletion, with the help of awesome examples.

**What are Views in MySQL?**

**Views** in MySQL are indeed **“virtual tables”** that are used to view data from one or more tables. Views do not have their data but rather store data virtually, consisting of rows and columns. Views are very helpful in restricting access to your application’s critical data to third-party users. Views in MySQL can be created by selecting some/all columns and some/all rows of a table by filtering out the rows based on some condition(s).

**What is the Benefit of Using Views in MySQL?**

Views help particularly in the following ways:

1. **Simplicity:** Instead of writing complex joins & queries, views provide a way of writing simple SELECT statements.
2. **Enhanced Security:** Views expose only the data to the third-party apps and hide the internal details like table structure, attributes, etc, thus adding extra security.
3. **Consistency:** By writing views instead of common queries, we can write a view that avoids multiple declarations & definitions of the same queries and eventually provides a centralized way.

Just like the normal tables, you can perform operations like create, update, drop, etc., on the views. We will look at their syntax along with examples now.

-- Create and use the database

CREATE DATABASE UniversityDB;

USE UniversityDB;

-- Create the StudentDetails table

CREATE TABLE StudentDetails (

sid INT PRIMARY KEY,

sname VARCHAR(50),

age INT,

university VARCHAR(100)

);

-- Insert data into StudentDetails table

INSERT INTO StudentDetails (sid, sname, age, university) VALUES

(1, 'Girish', 24, 'IIT Hyderabad'),

(2, 'Aaditya', 24, 'SRM University'),

(3, 'Aashish', 23, 'IIT Hyderabad'),

(4, 'John', 25, 'Mumbai University'),

(5, 'Shruti', 24, 'IIT Hyderabad'),

(6, 'Leena', 25, 'Mumbai University');

-- Create the CourseDetails table

CREATE TABLE CourseDetails (

cid INT PRIMARY KEY,

cname VARCHAR(100),

ratings DECIMAL(2, 1),

price DECIMAL(10, 2)

);

-- Insert data into CourseDetails table

INSERT INTO CourseDetails (cid, cname, ratings, price) VALUES

(1, 'Python Fundamentals', 4.6, 2999),

(2, 'Machine Learning', 4.3, 1999),

(3, 'DSA A-Z', 4.9, 5999),

(4, 'Competitive Programming', 4.7, 4999);

-- Create the EnrolledIn table

CREATE TABLE EnrolledIn (

sid INT,

cid INT,

PRIMARY KEY (sid, cid),

FOREIGN KEY (sid) REFERENCES StudentDetails(sid),

FOREIGN KEY (cid) REFERENCES CourseDetails(cid)

);

-- Insert data into EnrolledIn table

INSERT INTO EnrolledIn (sid, cid) VALUES

(1, 3),

(1, 4),

(2, 1),

(2, 3),

(3, 3),

(4, 1);

**Create View in MySQL**

A view in MySQL can be created based on a single table or multiple tables. The **CREATE VIEW** statement is used to create a view in MYSQL.

**1. Create a View Based On Single Table**

**Syntax:**

*CREATE VIEW <view\_name> AS*

*SELECT <column1>, <column2>……., <columnN>*

*FROM <table-name>*

*WHERE [conditions];*

**Example:**

Let’s us create a view named “**IITHyderabadStudentsView**” from the StudentDetails table. This view selects the students from the StudentDetails table who study in “IIT Hyderabad” university and outputs their details like student id, name, and age.

CREATE VIEW IITHyderabadStudentsView AS  
SELECT sid, sname, age   
FROM StudentDetails  
WHERE university = "IIT Hyderabad";

Now to view the tuples in the IITHyderabadStudentsView, we will query the view the just like how we query a normal table.

SELECT \* FROM IITHyderabadStudentsView;

-- Show all views in the current database

SHOW FULL TABLES IN universitydb WHERE TABLE\_TYPE = 'VIEW';

-- Get detailed information about views

SELECT TABLE\_NAME

FROM INFORMATION\_SCHEMA.VIEWS

WHERE TABLE\_SCHEMA = 'universitydb';

DROP VIEW:

**DROP VIEW IITHyderabadStudentsView;**

**2. Create a View Based On Multiple Tables With JOIN Clause**

We can create a view by joining multiple tables using the JOIN clause and additionally using WHERE keyword or GROUP BY, HAVING clauses.

**Syntax:**

*CREATE VIEW <view\_name> AS*

*SELECT <column1>, <column2>……., <columnN>*

*FROM <table1>*

*[ INNER | LEFT | RIGHT | FULL ] JOIN <table2> ON <joining-column>*

*WHERE [condition1 | condition2 | ……];*

**Example:**

Let’s us create a view named “**PythonEnrolledView**” using the StudentDetails, CourseDetails, and EnrolledIn table. This view outputs the students who are enrolled in “Python Fundamentals” course the details as student id, name, and age.

CREATE VIEW PythonEnrolledView AS  
SELECT S.sid, S.sname, S.age   
FROM StudentDetails S JOIN EnrolledIn E ON S.sid = E.sid  
JOIN CourseDetails C ON C.cid = E.cid  
WHERE C.cname = "Python Fundamentals";

Now to view the tuples in the PythonEnrolledView, we will again query the view the just like how we query a normal table.

SELECT \* FROM PythonEnrolledView;

**Update View in MySQL**

There are certain conditions that need to be satisfied to update a view. If any one of these conditions is not met, then we are not allowed to update the view.

1. The view can be created from one or more tables. However, views created from multiple tables may have restrictions on updates.
2. The view can be created using nested or complex queries, but this may limit the view's updatability.
3. The view can include columns with NULL values, depending on the data in the underlying tables.
4. The SELECT statement used to create the view should avoid using the DISTINCT keyword if the view needs to be updatable.
5. The SELECT statement used to create the view can include GROUP BY or ORDER BY clauses, but this may restrict the view's updatability.

**1. Update View Definition/Structure**

To update the view for adding or remove columns and rows by changing WHERE clause condition, we can use **CREATE OR REPLACE VIEW** statement.

**Syntax:**

*CREATE OR REPLACE VIEW <view\_name> AS*

*SELECT <column1>, <column2>, ………, <columnN>*

*FROM <table\_name>*

*WHERE [condition];*

**Example:**

Suppose we want to update the view IITHyderabadStudentsView we created above and delete the column sid from this view from StudentDetails table, we can do this as follows:

**CREATE OR REPLACE VIEW IITHyderabadStudentsView AS**

**SELECT sname, age, university**

**FROM StudentDetails**

**WHERE university = 'IIT Hyderabad';**

**2. Insert Into View**

To insert the new row into the view, we can do it in a similar way just like how we do it for normal tables.

**Syntax:**

INSERT INTO <view\_name>(<column1>, <column2>, <column3>,.........)   
VALUES(<value1>, <value2>, <value3>,...........);

**Example:**

Let us insert a new row in the view IITHyderabadStudentsView which we have created above in the example of “Create View-based On Single Table”.

INSERT INTO StudentDetails (sid, sname, age, university)

VALUES (11, 'Tenali Rama', 26, 'IIT Hyderabad');  
  
SELECT \* FROM IITHyderabadStudentsView;  
SELECT \* FROM StudentDetails;

**Drop View:**

**DROP VIEW IITHyderabadStudentsView;**

**3. Delete From View**

To insert the new row into the view, we can do it in a similar way just like how we do it for normal tables. The syntax is:

DELETE FROM <view\_name> WHERE [condition];

**Example:**

Let us delete a row in the view IITHyderabadStudentsView which we have created above in the example of “Create View-based On Single Table”. Let us delete the details of student whose name is “Tenali Rama”.

DELETE FROM IITHyderabadStudentsView WHERE sname = "Tenali Rama";

DELIMITER $$

CREATE TRIGGER prevent\_delete\_through\_view

BEFORE DELETE ON StudentDetails

FOR EACH ROW

BEGIN

-- Check if the delete operation is coming from the view

IF EXISTS (

SELECT \* FROM IITHyderabadStudentsView

WHERE sname = OLD.sname

AND age = OLD.age

AND university = OLD.university

) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Deletion through the view is not allowed.';

END IF;

END $$

DELIMITER ;

1. From the following table, create a view for those salespeople who belong to the city of New York.

**Sample table: salesman**

-- Create the salesman table

CREATE TABLE salesman (

salesman\_id INT PRIMARY KEY,

name VARCHAR(50),

city VARCHAR(50),

commission DECIMAL(4, 2)

);

-- Insert data into the salesman table

INSERT INTO salesman (salesman\_id, name, city, commission) VALUES

(5001, 'James Hoog', 'New York', 0.15),

(5002, 'Nail Knite', 'Paris', 0.13),

(5005, 'Pit Alex', 'London', 0.11),

(5006, 'Mc Lyon', 'Paris', 0.14),

(5007, 'Paul Adam', 'Rome', 0.13),

(5003, 'Lauson Hen', 'San Jose', 0.12);

**-- Creating a VIEW named 'newyorkstaff'**

**CREATE VIEW newyorkstaff**

**-- Selecting all columns from the 'salesman' table**

**AS SELECT \***

**-- Filtering rows in the 'salesman' table where the 'city' column is 'New York'**

**FROM salesman**

**WHERE city = 'New York';**

1. From the following table, create a view for all salespersons. Return salesperson ID, name, and city.

**Sample table: salesman**

**-- Creating a VIEW named 'salesown'**

**CREATE VIEW salesown**

**-- Selecting specific columns (salesman\_id, name, city) from the 'salesman' table**

**AS SELECT salesman\_id, name, city**

**-- Retrieving data from the 'salesman' table and storing it in the VIEW**

**FROM salesman;**

1. From the following table, create a view to locate the salespeople in the city 'New York'.

**Sample table: salesman**

**-- Creating a VIEW named 'newyorkstaff'**

**CREATE VIEW newyorkstaff**

**-- Selecting all columns from the 'salesman' table**

**AS SELECT \***

**-- Filtering rows in the 'salesman' table where the 'city' column is 'New York'**

**FROM salesman**

**WHERE city = 'New York';**

SELECT \*

FROM newyorkstaff

WHERE commission > 0.13;

1. From the following table, create a view that counts the number of customers in each grade.

**Sample table: customer**

**-- Create the customer table**

**CREATE TABLE customer (**

**customer\_id INT PRIMARY KEY,**

**cust\_name VARCHAR(50),**

**city VARCHAR(50),**

**grade INT NULL,**

**salesman\_id INT**

**);**

**-- Insert data into the customer table**

**INSERT INTO customer (customer\_id, cust\_name, city, grade, salesman\_id) VALUES**

**(3002, 'Nick Rimando', 'New York', 100, 5001),**

**(3007, 'Brad Davis', 'New York', 200, 5001),**

**(3005, 'Graham Zusi', 'California', 200, 5002),**

**(3008, 'Julian Green', 'London', 300, 5002),**

**(3004, 'Fabian Johnson', 'Paris', 300, 5006),**

**(3009, 'Geoff Cameron', 'Berlin', 100, 5003),**

**(3003, 'Jozy Altidor', 'Moscow', 200, 5007),**

**(3001, 'Brad Guzan', 'London', NULL, 5005);**

-- Creating a VIEW named 'gradecount' with columns 'grade' and 'number'

CREATE VIEW gradecount (grade, number)

-- Selecting columns 'grade' and the count of rows as 'number' from the 'customer' table

-- Grouping the result by the 'grade' column

AS SELECT grade, COUNT(\*)

FROM customer

GROUP BY grade;

**update gradecount**

**set grade = 100**

**where grade = 150;**

**ERROR 1288 (HY000): The target table gradecount of the UPDATE is not updatable**

1. From the following table, create a view to count the number of unique customers, compute the average and the total purchase amount of customer orders by each date.

**Sample table : orders**

-- Create the orders table

CREATE TABLE orders (

ord\_no INT PRIMARY KEY,

purch\_amt DECIMAL(10, 2),

ord\_date DATE,

customer\_id INT,

salesman\_id INT

);

**-- Insert data into the orders table**

**INSERT INTO orders (ord\_no, purch\_amt, ord\_date, customer\_id, salesman\_id) VALUES**

**(70001, 150.5, '2012-10-05', 3005, 5002),**

**(70009, 270.65, '2012-09-10', 3001, 5005),**

**(70002, 65.26, '2012-10-05', 3002, 5001),**

**(70004, 110.5, '2012-08-17', 3009, 5003),**

**(70007, 948.5, '2012-09-10', 3005, 5002),**

**(70005, 2400.6, '2012-07-27', 3007, 5001),**

**(70008, 5760, '2012-09-10', 3002, 5001),**

**(70010, 1983.43, '2012-10-10', 3004, 5006),**

**(70003, 2480.4, '2012-10-10', 3009, 5003),**

**(70012, 250.45, '2012-06-27', 3008, 5002),**

**(70011, 75.29, '2012-08-17', 3003, 5007),**

**(70013, 3045.6, '2012-04-25', 3002, 5001);**

-- Creating a VIEW named 'totalforday'

CREATE VIEW totalforday

-- Selecting columns 'ord\_date', count of distinct 'customer\_id', average 'purch\_amt', and total 'purch\_amt' from the 'orders' table

-- Grouping the result by the 'ord\_date' column

AS SELECT ord\_date, COUNT(DISTINCT customer\_id), AVG(purch\_amt), SUM(purch\_amt)

FROM orders

GROUP BY ord\_date;

1. From the following tables, create a view to get the salesperson and customer by name. Return order name, purchase amount, salesperson ID, name, customer name.

**Sample table: salesman**

**Sample table: customer**

**Sample table: orders**

**-- Create the salesman table**

**CREATE TABLE salesman (**

**salesman\_id INT PRIMARY KEY,**

**name VARCHAR(50),**

**city VARCHAR(50),**

**commission DECIMAL(4, 2)**

**);**

**-- Insert data into the salesman table**

**INSERT INTO salesman (salesman\_id, name, city, commission) VALUES**

**(5001, 'James Hoog', 'New York', 0.15),**

**(5002, 'Nail Knite', 'Paris', 0.13),**

**(5005, 'Pit Alex', 'London', 0.11),**

**(5006, 'Mc Lyon', 'Paris', 0.14),**

**(5007, 'Paul Adam', 'Rome', 0.13),**

**(5003, 'Lauson Hen', 'San Jose', 0.12);**

**-- Creating a VIEW named 'nameorders'**

**CREATE VIEW nameorders**

-- Selecting columns 'ord\_no', 'purch\_amt', 'salesman\_id', 'name', and 'cust\_name' from the 'orders', 'customer', and 'salesman' tables

-- Using aliases 'a', 'b', and 'c' for the respective tables

-- Joining tables based on matching 'customer\_id' and 'salesman\_id'

AS SELECT ord\_no, purch\_amt, a.salesman\_id, name, cust\_name

FROM orders a, customer b, salesman c

WHERE a.customer\_id = b.customer\_id

AND a.salesman\_id = c.salesman\_id;

1. From the following table, create a view to find the salesperson who handles a customer who makes the highest order of the day. Return order date, salesperson ID, name.

**Sample table: customer**

**Sample table: salesman**

**Sample table: orders**

CREATE VIEW elitsalesman AS

SELECT b.ord\_date, a.salesman\_id, a.name

FROM salesman a, orders b

WHERE a.salesman\_id = b.salesman\_id

AND b.purch\_amt = (

SELECT MAX(purch\_amt)

FROM orders c

WHERE c.ord\_date = b.ord\_date

);

1. From the following table, create a view to find the salesperson who deals with the customer with the highest order at least three times per day. Return salesperson ID and name.

**Sample table: customer**

**Sample table: elitsalesman**

CREATE VIEW incentive AS

SELECT DISTINCT a.salesman\_id, a.name

FROM elitsalesman a

WHERE (

SELECT COUNT(\*)

FROM elitsalesman b

WHERE a.salesman\_id = b.salesman\_id

) >= 3;

1. From the following table, create a view to find all the customers who have the highest grade. Return all the fields of customer.

**Sample table: customer**

CREATE VIEW highgrade AS

SELECT \*

FROM customer

WHERE grade = (

SELECT MAX(grade)

FROM customer

);

1. From the following table, create a view to count the number of salespeople in each city. Return city, number of salespersons.

**Sample table: salesman**

CREATE VIEW citynum AS

SELECT city, COUNT(DISTINCT salesman\_id) AS num\_salesmen

FROM salesman

GROUP BY city;

1. From the following table, create a view to compute the average purchase amount and total purchase amount for each salesperson. Return name, average purchase and total purchase amount. (Assume all names are unique.).

**Sample table: salesman**

**Sample table: orders**

-- Creating a VIEW named 'norders'

CREATE VIEW norders

-- Selecting 'name', average of 'purch\_amt', and sum of 'purch\_amt' for each salesman

-- Joining the 'salesman' and 'orders' tables on 'salesman\_id'

-- Grouping the results by 'name'

AS SELECT name, AVG(purch\_amt), SUM(purch\_amt)

FROM salesman, orders

WHERE salesman.salesman\_id = orders.salesman\_id

GROUP BY name;

1. From the following table, create a view to identify salespeople who work with multiple clients. Return all the fields of salesperson.

**Sample table: salesman**

**Sample table: customer**

-- Creating a VIEW named 'mcustomer'

CREATE VIEW mcustomer

-- Selecting all columns from the 'salesman' table as 'a'

-- Filtering the rows where a salesman has more than one customer

-- Using a subquery to count the number of customers for each salesman and comparing it to 1

AS SELECT \*

FROM salesman a

WHERE 1 <

(SELECT COUNT(\*)

FROM customer b

WHERE a.salesman\_id = b.salesman\_id);

1. From the following table, create a view that shows all matching customers with salespeople, ensuring that at least one customer in the city of the customer is served by the salesperson in the city of the salesperson.

**Sample table: salesman**

**Sample table: customer**

-- Creating a VIEW named 'citymatch' with columns 'custcity' and 'salescity'

CREATE VIEW citymatch(custcity, salescity)

-- Selecting distinct pairs of customer city and salesman city

-- Matching rows from 'customer' and 'salesman' tables where the salesman has customers

-- Using a common column 'salesman\_id' for the match

AS SELECT DISTINCT a.city, b.city

FROM customer a, salesman b

WHERE a.salesman\_id = b.salesman\_id;

1. From the following table, create a view to display the number of orders per day. Return order date and number of orders.

**Sample table: orders**

-- Creating a VIEW named 'dateord' with columns 'ord\_date' and 'odcount'

CREATE VIEW dateord(ord\_date, odcount)

-- Selecting distinct order dates and counting the number of orders for each date

-- Using the 'orders' table and grouping by 'ord\_date'

AS SELECT ord\_date, COUNT(\*)

FROM orders

GROUP BY ord\_date;

1. From the following table, create a view to find the salespeople who placed orders on October 10th, 2012. Return all the fields of salesperson.

**Sample table: salesman**

**Sample table: orders**

-- Creating a VIEW named 'salesmanonoct' with all columns from the 'salesman' table

CREATE VIEW salesmanonoct

-- Selecting rows from the 'salesman' table where the 'salesman\_id' is present in the result of the subquery

-- The subquery selects 'salesman\_id' from the 'orders' table where 'ord\_date' is '2012-10-10'

AS SELECT \*

FROM salesman

WHERE salesman\_id IN

(SELECT salesman\_id

FROM orders

WHERE ord\_date = '2012-10-10');

1. From the following table, create a view to find the salespersons who issued orders on either August 17th, 2012 or October 10th, 2012. Return salesperson ID, order number and customer ID.

**Sample table: orders**

-- Creating a VIEW named 'sorder' with selected columns from the 'orders' table

CREATE VIEW sorder

-- Selecting columns 'salesman\_id', 'ord\_no', and 'customer\_id' from the 'orders' table

-- Filtering rows where 'ord\_date' is in the range from '2012-08-17' to '2012-10-10'

AS SELECT salesman\_id, ord\_no, customer\_id

FROM orders

WHERE ord\_date IN ('2012-08-17', '2012-10-10');