**Lab 7: Views**

**MySQL Views**

In MYSQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

**MySQL CREATE VIEW Statement**

The CREATE VIEW statement creates a new view in the database. Here is the basic syntax of the CREATE VIEW statement:

**CREATE [OR REPLACE] VIEW [db\_name.]view\_name [(column\_list)] AS select-statement;**

In this syntax:

First, specify the name of the view that you want to create after the CREATE VIEW keywords. The name of the view is unique in a database. Because views and tables in the same database share the same namespace, the name a view cannot the same as the name of an existing table.

Second, use the OR REPLACE option if you want to replace an existing view if the view already exists. If the view does not exist, the OR REPLACE has no effect.

Third, specify a list of columns for the view. By default, the columns of the view are derived from the select list of the SELECT statement. However, you can explicitly specify the column list for the view by listing them in parentheses following the view name.

Finally,  specify a SELECT statement that defines the view. The SELECT  statement can query data from tables or views.

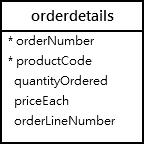
By default, the CREATE VIEW statement creates a view in the current database. If you want to explicitly create a view in a given database, you can qualify the view name with the database name.

**MySQL CREATE VIEW examples**

Let’s take some example of using the CREATE VIEW statement to create new views.

**1) Creating a simple view example**

Let’s take a look at the orderDetails table from the sample database named “classicmodels” that we have created in Lab-2:



This statement uses the CREATE VIEW statement to create a view that represents total sales per order.

**CREATE VIEW salePerOrder AS**

**SELECT**

**orderNumber,**

**SUM(quantityOrdered \* priceEach) total**

**FROM**

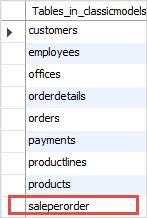
**orderDetails**

**GROUP by orderNumber**

**ORDER BY total DESC;**

If you use the SHOW TABLE command to view all tables in the classicmodels database, you will see the viewsalesPerOrder is showing up in the list.

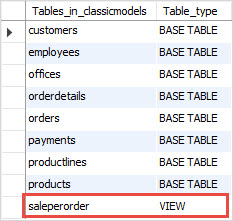
**SHOW TABLES**



This is because the views and tables share the same namespace as mentioned earlier.

To know which object is a view or table, you use the SHOW FULL TABLES command as follows:

**SHOW FULL TABLES**

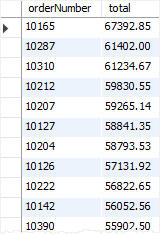


The table\_type column in the result set specifies the type of the object: view or table (base table). [Note: In phymyadmin, you can see the views under views section at the left submenu of the database]

If you want to query total sales for each sales order, you just need to execute a

simple SELECT  statement against the SalePerOrder  view as follows:

**SELECT \* FROM salePerOrder;**



**2) Creating a view based on another view example**

MySQL allows you to create a view based on another view.

For example, you can create a view called bigSalesOrder based on the salesPerOrder view to show every sales order whose total is greater than 60,000 as follows:

**CREATE VIEW bigSalesOrder AS**

**SELECT**

**orderNumber,**

**ROUND(total,2) as total**

**FROM**

**salePerOrder**

**WHERE**

**total > 60000;**

Now, you can query the data from the bigSalesOrder view as follows:

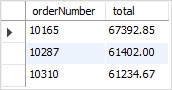
**SELECT**

**orderNumber,**

**total**

**FROM**

**bigSalesOrder;**



**3) Creating a view with explicit view columns example**

This statement uses the CREATE VIEW statement to create a new view based on the customers and orders tables with explicit view columns:

**CREATE VIEW customerOrderStats (**

**customerName ,**

**orderCount**

**)**

**AS**

**SELECT**

**customerName,**

**COUNT(orderNumber)**

**FROM**

**customers**

**INNER JOIN**

**orders USING (customerNumber)**

**GROUP BY customerName;**

This query returns data from the customerOrderStats view:

**SELECT**

**customerName,**

**orderCount**

**FROM**

**customerOrderStats**

**ORDER BY**

**orderCount,**

**customerName;**



**MySQL updatable views**

In MySQL, views are not only query-able but also updatable. It means that you can use the [INSERT](https://www.mysqltutorial.org/mysql-insert-statement.aspx) or [UPDATE](https://www.mysqltutorial.org/mysql-update-data.aspx) statement to insert or update rows of the base table through the updatable view. In addition, you can use [DELETE](https://www.mysqltutorial.org/mysql-delete-statement.aspx) statement to remove rows of the underlying table through the view.

However, to create an updatable [view](https://www.mysqltutorial.org/introduction-sql-views.aspx), the [SELECT statement](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) that defines the view must not contain any of the following elements:

* [Aggregate functions](https://www.mysqltutorial.org/mysql-aggregate-functions.aspx) such as [MIN](https://www.mysqltutorial.org/mysql-min/), [MAX](https://www.mysqltutorial.org/mysql-max-function/), [SUM](https://www.mysqltutorial.org/mysql-sum/), [AVG](https://www.mysqltutorial.org/mysql-avg/), and [COUNT](https://www.mysqltutorial.org/mysql-count/).
* [DISTINCT](https://www.mysqltutorial.org/mysql-distinct.aspx)
* [GROUP BY](https://www.mysqltutorial.org/mysql-group-by.aspx) clause.
* [HAVING](https://www.mysqltutorial.org/mysql-having.aspx) clause.
* [UNION](https://www.mysqltutorial.org/sql-union-mysql.aspx) or UNION ALL clause.
* [Left join](https://www.mysqltutorial.org/mysql-left-join.aspx) or outer join.
* [Subquery](https://www.mysqltutorial.org/mysql-subquery/)in the [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) clause or in the [WHERE](https://www.mysqltutorial.org/mysql-where/) clause that refers to the table appeared in the FROM clause.
* Reference to non-updatable view in the FROM clause.
* Reference only to literal values.
* Multiple references to any column of the base table.

**4) MySQL updatable view example**

Let’s create an updatable view.

First, we create a view named officeInfo  based on the offices  table in the [sample database](https://www.mysqltutorial.org/mysql-sample-database.aspx). The view refers to three columns of the offices  table:officeCode phone,  and city.

**CREATE VIEW officeInfo**

**AS**

**SELECT officeCode, phone, city**

**FROM offices;**

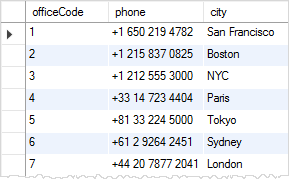
Next, we can query data from the officeInfo view using the following statement:

**SELECT**

**\***

**FROM**

**officeInfo;**



Then, we can change the phone number of the office with officeCode  4 through the officeInfo view using the following [UPDATE](https://www.mysqltutorial.org/mysql-update-data.aspx)statement.

**UPDATE officeInfo**

**SET**

**phone = '+33 14 723 5555'**

**WHERE**

**officeCode = 4;**

Finally, to verify the change, we can query the data from the officeInfo  view by executing the following query:

**SELECT**

**\***

**FROM**

**officeInfo**

**WHERE**

**officeCode = 4;**

mysql updateable view example with officeInfo View

**Checking updatable view information**

You can check if a view in a database in updatable by querying the is\_updatable column from the views table in the information\_schema database.

The following query gets all views from the [classicmodels database](https://www.mysqltutorial.org/mysql-sample-database.aspx) and shows which views are updatable.

**SELECT**

**table\_name,**

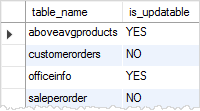
**is\_updatable**

**FROM**

**information\_schema.views**

**WHERE**

**table\_schema = 'classicmodels';**



So if we attempt to update saleperorder view;

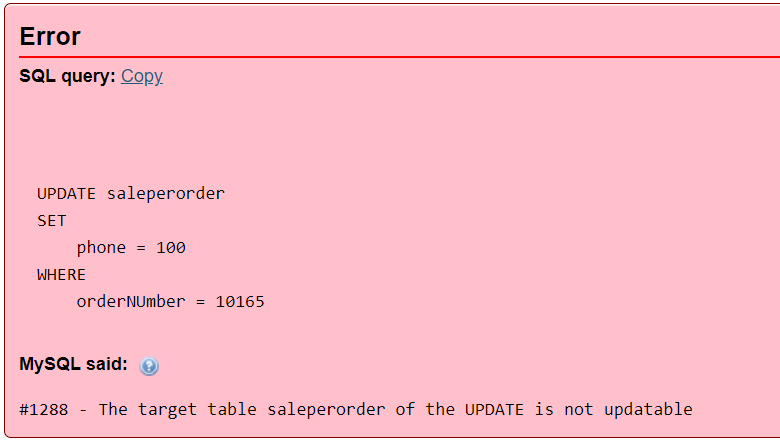
**UPDATE saleperorder**

**SET**

**total = 100**

**WHERE**

**orderNumber = 10165;**



Since the saleperorder is not an updatable view.

**Removing rows through the view**

First, we [create a table](https://www.mysqltutorial.org/mysql-create-table/) named items, [insert](https://www.mysqltutorial.org/mysql-insert-statement.aspx) some rows into the items table, and [create a view](https://www.mysqltutorial.org/create-sql-views-mysql.aspx) that contains items whose prices are greater than 700.

-- create a new table named items

**CREATE TABLE items (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**name VARCHAR(100) NOT NULL,**

**price DECIMAL(11 , 2 ) NOT NULL**

**);**

-- insert data into the items table

**INSERT INTO items(name,price)**

**VALUES('Laptop',700.56),('Desktop',699.99),('iPad',700.50) ;**

-- create a view based on items table

**CREATE VIEW LuxuryItems AS**

**SELECT**

**\***

**FROM**

**items**

**WHERE**

**price > 700;**

-- query data from the LuxuryItems view

**SELECT**

**\***

**FROM**

**LuxuryItems;**



Second, we use the DELETE statement to remove a row with id value 3.

**DELETE FROM LuxuryItems**

**WHERE**

**id = 3;**

MySQL returns a message saying that 1 row(s) affected.

Third, let’s check the data through the view again.

**SELECT**

**\***

**FROM**

**LuxuryItems;**

MySQL DELETE through View

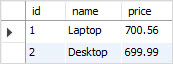
Fourth, we can also query the data from the base table items to verify if the DELETE statement actually deleted the row.

**SELECT**

**\***

**FROM**

**items;**



As you see, the row with id 3 was removed from the base table.

**Inserting Values into the views**

On the updatable views, values can be inserted and hence to the base table. Following inserts values into the view LuxuryItems

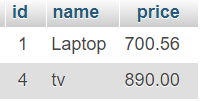
**Insert into LuxuryItems(name, price)**

**values (“TV”, 890)**

When we select records from the view as;

**Select \* from LuxuryItems**

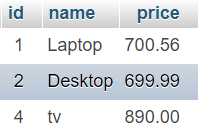
The result is:



**The insert operation has reflected changes in the base table items.**

**Select \* from Items**

The result is:



**5) Deleting the View:**

You can use the drop statement to delete the view. The syntax is: Drop view viewname

**Drop view LuxuryItems**

It will delete the viewe LuxuryItems.