Views - Stored Queries

DSC 301: Lecture 15

April 7, 2021

Lecture Objectives

In this lesson you will learn about **views**:

- What is a view?
- Why use views?
- How to `CREATE` and `DROP` views?
- When to use views?

What is a view?

A **view** is a stored query. When run, a view looks like and acts like a table from the users point of view. A view is a database object and provides some advantages over directly interacting with tables.

Why use views?

- Frequent queries - a view can replace frequently written queries since there is no other way to save a query within the database itself. A query needs to be saved and stored as a `.sql` file, whereas a view is a database object.
- Data security - views provide security by restricting access to table data including rows and columns. For example, a view can limit results to only records meeting a particular criteria or limit access to columns such as email address or other personal information of the customers.
- Query simplification - A view can be based on nearly any `SELECT` statement, including complex queries that contain joins and subqueries. Subsequently, a new (simple) query can be based on that view. In particular, a view can be created that contains multiple joins, which returns a result set. A new (simple) query can be written to select data from the results of the view.

- Design independence - the design of a table may be modified, then views can be modified without re-programming the application that accesses the view or without users needing to rewrite queries.
- User-friendliness - data can be formatted in more front-end user-friendly manner with views.

How to create a view?

Before specifying the syntax for creating a view, here are some View rules:

- Names must be unique - cannot be named the same as another table or view
- There is no limit to the number of views that can be created
- Privileges must be given to create views (contact your DBA)
- Views can be nested, that is a view may be built on other views. Nesting levels is depended on the DBMS and a view based on multiple other views (i.e., multiple nested views) will degradate performance.
- Some DBMS require column aliasing and prohibit `ORDER BY` clause. Check the documentation of your DBMS.
- Views cannot be indexed like tables.
- Views can be used to create tables.

```
CREATE VIEW <view_name> AS
    SELECT
        <columns>
    FROM
        <TableX>
    [WHERE condition];
```

Example 1. *Create a view of low inventory products (this is probably under frequent consideration).*

```
CREATE VIEW `LowInventory` AS
    SELECT
        id,  inventory
    FROM
        Products
    WHERE
        inventory < 5;
```

Add `OR REPLACE` keywords after the `CREATE` keyword in order to automatically drop a view that has already been created or has the same name as the view you are creating. The specific syntax in this case is:

```
CREATE OR REPLACE VIEW <view_name> AS
    SELECT
        <columns>
    FROM
        <TableX>
    [WHERE condition];
```

Example 2. *Create a list of all customers from West Virginia. Use the `OR REPLACE` keywords in this case.*

```
CREATE OR REPLACE VIEW `WVCustomers` AS
    SELECT
        id, fname, lname, phone, email
    FROM
        Customers AS C
            INNER JOIN
        Zipcodes AS Z ON C.zip = Z.zip
    WHERE
        Z.state = 'WV';
```

Note: In MySQL Workbench, you can also create a view by right-clicking on *Views* icon in the *Schemas* panel, then choosing "Create View", and type your select statements.

Drop a view

Use the `DROP VIEW` command to delete a view from the database. The syntax is:

```
DROP VIEW <view_name>;
```

Example 3. *Drop the West Virginia customers view created in previous example.*

```
DROP VIEW `WVCustomers`;
```

MySQL Workbench provides means to drop[1] a view via point-and-click (see Figure 1).

---

[1] MySQL Workbench provides the ability create, drop, and alter through the interface.

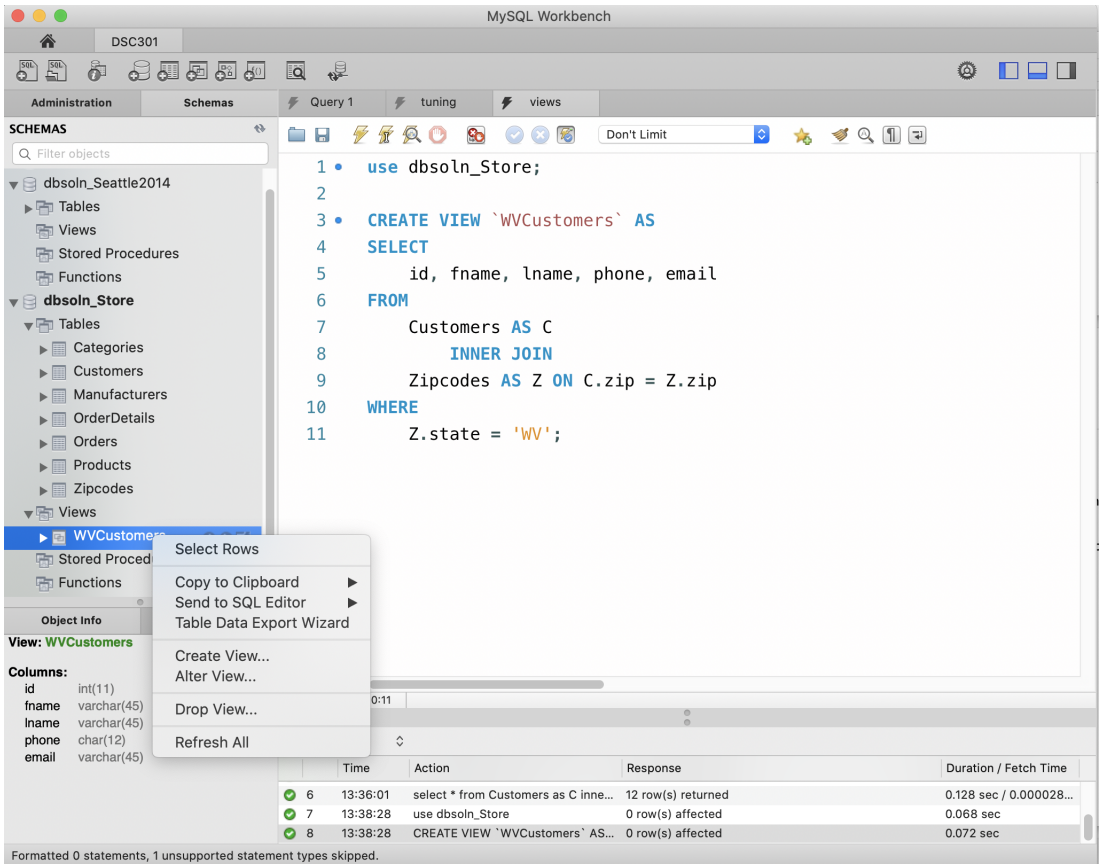

Figure 1: MySQL Workbench Views

Updatable views - writable views

An *updatable view* (as opposed to read-only views) is a view that can be used with `INSERT`, `UPDATE`, or `DELETE` statement to update the data in a base table.

Updating the underlying data from a view can be accomplished as long as:

- The view does not contain joins.
- The view does not include aggregate functions.
- The view does not use the `GROUP BY` or `HAVING` clause.
- The view does not contain a `UNION` statement.
- `DISTINCT` is not used.

First create a updatable view. For example,

```
CREATE VIEW `LowInventory` AS
    SELECT
        id,  inventory
    FROM
        Products
    WHERE
        inventory < 5;
```

Next, update this view by setting all inventory fields to 10.

```
UPDATE `LowInventory`
    SET
        inventory = 10;
```

A clause that can prevent an update statement from excluding rows of the view can be added when creating a view. Specifically, the `WITH CHECK OPTION` can be added to the end of the `CREATE VIEW` statement.

```
CREATE OR REPLACE VIEW <view_name> AS
    SELECT
        <columns>
    FROM
        <TableX>
    [WHERE condition]
WITH CHECK OPTION;
```