# **CIS-336: Lab7: Working with Views**

**LAB OVERVIEW**

Lab 7 will introduce the concept of database views. This lab may be completed using either DeVry’s Omnymbus EDUPE-APP lab environment, or a local copy of the MySQL database running on your own computer using the OM database tables. The lab will utilize a set of tables that are represented by the ERD (**OM\_ERD.docx**) and are created and populated by the script file (**create\_OM\_db.sql**). Follow the instructions in the file **CreateOMTables.docx** to create your database, tables, and data.

**A few IMPORTANT things to note if using EDUPE MySQL:**

\*\*There can be NO SPACES in alias names given to a column. For example:

Select unit\_price as “Retail Price “ from items; --this does NOT work in EDUPE MySQL.

Any of the following WILL WORK:

Select unit\_price as "RetailPrice" from items;

Select unit\_price as "Retail\_Price" from items;

Select unit\_price as Retail\_Price from items;

Select unit\_price as RetailPrice from items;

\*\*Any calculated fields MUST be given an alias (and note above NO SPACES in alias). For example:

select unit\_price \* 2 from items; --this does NOT work in EDUPE MySQL

This will work:

select unit\_price \* 2 as NewPrice from items;

**Deliverables**

* Lab Report (Answer Sheet) containing both the student-created SQL command(s) for each exercise, and the output showing the results obtained. **Be sure your name is on the file.**

**LAB STEPS: Complete each of the exercises below.**

1. Use an ALTER TABLE statement to update the customers table so that the Primary Key field is an auto-increment field, then create TWO insert statements to test proper operation, using your own first and last name for one (and a name of your choice for the second one), and any data you care to imagine for the remaining fields.

IMPORTANT NOTE: When using a LOCAL copy of MySQL, if you attempt to simply issue the ALTER TABLE command you have composed by itself, you should receive an error similar to the following (try it for yourself!).

ERROR 1833: Cannot change column 'customer\_id': used in a foreign key constraint 'orders\_fk\_customers' of table 'om.orders'

(Note – EDUPE will not give this error message, however you should still follow the CORRECT procedure as discussed here to complete this problem).

The reason for this is that you are attempting to alter data in one column that has a defined PK:FK relationship to a field in another table. Referential Integrity rules prevent this. So, how do you resolve such a problem?

One approach to solving this dilemma is to turn off the foreign key checks that implement referential integrity rules. However, the danger here is that other users and processes operating on the database while these constraints are suspended could create or modify data in a way that compromises integrity. We can solve this second problem by preventing other users and processes from altering the data in the table in which we are working until we have turned the foreign key checks back on. We therefore need to construct a script that does the following.

1. Locks the customer table -- lock table customers write;
2. Turns off FK checks -- set foreign\_key\_checks = 0;
3. Alters the table to add the auto\_increment feature to the PK field
4. Turns FK checks back on -- set foreign\_key\_checks = 1;
5. Unlocks the customer table -- unlock tables;

It is VERY important to consider that altering tables can require a bit of time for very large tables, and that while the table is locked, other users and processes cannot operate. Consequently, this kind of modification should not be done during peak operating hours in a production operation (as a student in a lab exercise, working on your own database, you may do this at any time) but ideally in hours during which the business does not normally operate. In cases where round-the-clock, high availability of a database is required, other approaches may be required. Addressing this problem in a high-availability, high-demand environment is an advanced topic, study of which is outside the scope of this course. Use the outline below to construct your script. Show all commands in your answer sheet along with the output of the commands.

lock table customers write;

set foreign\_key\_checks = 0;

-- Replace this comment with your ALTER TABLE command to add the auto\_increment feature to the PK field

set foreign\_key\_checks = 1;

unlock tables;

--statements to insert two rows into the table

--verify auto\_increment with a select statement

1. The Vice President of Marketing for your firm wants the firm’s sales representatives to be able to directly view and edit customer details, but only for the state to which a particular sales representative is assigned. You have suggested that this need can be addressed with a view. For example, a view could be created for one particular state, and user account permissions for accessing that view granted only to sales representatives from that state. The VP has asked you to quickly create a simple proof-of-concept demonstrating how this might work. Complete the following steps:

a. Construct a view on the customers table called CA\_CUSTOMERS that consists of all data about customers that live in California.

b. Display the data **using this view** to verify that only customers that reside in California are visible.

c. Prove that It is possible to add or update records through this view by updating the record for Karina Lacy to change the spelling of Karina’s last name to Lacie.

d. Display the **data using the customer table** to verify that the change has been made.

Show all commands in your answer sheet along with the output of the commands.

1. The Senior Customer Service Manager has requested the ability to create a report at any time that will show shipped orders that took some specified number of days to fulfill.

a. Create a view named SHIPPING\_TIME that lists only customer\_first\_name, customer\_last\_name, order\_date, shipped\_date, and the calculated field days\_to\_fulfill (use the DATEDIFF function) showing the number of days between when the customer placed the order and when it was shipped. Show the data from this view.

Now let’s do some queries by adding sorting and filters **USING THIS VIEW, WITHOUT CHANGING IT**.

b. Use the view to display the data sorted by highest to lowest days to ship

c. Use the view to display only the orders that took less than 10 days to ship.

d. Use the view to display only the orders that took more than 30 days to ship.

1. Queries that require joins and aggregate functions can be easier to construct when using a view as a “temporary” table. Consider a report to show total sales by artist.

a. First create a view called SalesData that displays the order\_id, item\_id, the calculated field ItemTotal (which is quantity times price), the title and artist\_id.

b. Display the data in the SalesData view sorted by artist\_id. Does this help you to “visualize” how to group the data to create the totals?

c. Create a query USING THIS VIEW and the appropriate aggregate function to display artist\_id and the total sales for each artist.

d. Now join to the artist table in order to display the artist\_name along with the total sales.

1. Now use this same method to display the total sales per customer.

a. Create a view called SalesData with the appropriate data. At a minimum you will need customer\_id and the calculated item total. DO NOT use the customer table in this view, it will be joined later.

b. Display the data in your view sorted by customer\_id. Does this help you to “visualize” how to group the data to create the totals?

c. Create a query USING THIS VIEW and the appropriate aggregate function to display customer\_id and the total sales for each customer.

d. Now join to the customer table in order to display the customer\_name as a single field named Customer along with the total sales. Sort the report by Total sales in descending order.

**This is the end of Lab 7.**