# **This PE is due by the date on the Assignment dropbox.**

# **Please note: This document will be in MyCourses as a zip file named “PE10\_Workbench.zip”. It will contain the three product scripts (product\_details.sql, product\_summary.sql, product\_statements.sql), the guitar database script (my\_guitar\_shop.sql) and the model for the database (my\_guitar\_shop.mwb) as well as this problem description.**

**You are welcome to use the command line interface or Workbench to complete this assignment.**

**If you decide to use Workbench, please download it the first day the assignment is introduced.**

**ISTE 330 Database Connectivity, the next course after ISTE 230 or ISTE608, (if you are interested in databases) uses Workbench the entire semester.**

# **Getting Started with MySQL Workbench**

1. Start MySQL Workbench and open a connection for managing the server (you will have to enter your password).
2. Open the script file named *my\_guitar\_shop.sql* by clicking the **Open SQL Script File** (**Ctrl+Shift+O**) button in the SQL Editor toolbar. Then, use the resulting dialog box to locate and open the file.
3. Execute the entire script by clicking the **Execute SQL Script** (it looks like a lightning bolt) button in the code editor toolbar or by pressing **Ctrl+Shift+Enter**. You can also execute just the query where your cursor is currently located by pressing **Ctrl+Enter**. When you do, the Output window displays messages that indicate whether the script executed successfully. (check this with highlighting and without highlighting)
4. Open the script named *product\_details.sql*. Note that this script contains just one SQL statement. Then, run the statement.
5. Open the script named *product\_summary.sql*. Note that this opens another code editor tab.
6. Open the script named *product\_statements.sql*. Notice that this script contains two SQL statements that end with semicolons.
7. In the **Object Browser** window (the left-most pane), expand the node for the database named *my\_guitar\_shop* so you can see all the database objects it contains. If it isn’t displayed in the Object Browser window, you may need to click on the **Refresh** (blue circular looking arrows) button to display it.

Retrieve Data from a Single Table

***The goal of this question is to produce a simple select with four columns ordered by “list\_price”, descending.***

1. Write a SELECT statement that returns four columns from the Products table: product\_code, product\_name, list\_price, and discount\_percent. Then, run this statement to make sure it works correctly.

Add an ORDER BY clause to this statement that sorts the result set by list price in descending sequence. Then, run this statement again to make sure it works correctly. This is a good way to build and test a statement, one clause at a time.

***The goal of this question is to concatenate two columns into one as an ALIAS ordered by “last\_name”, ascending. Only customers with “last\_name” starting with M to Z should be shown.***

1. Write a SELECT statement that returns one column from the Customers table named full\_name that joins the last\_name and first\_name columns.

Format this column with the last name, a comma, a space, and the first name like this:

Doe, John

Sort the result set by last name in ascending sequence.

Return only the customers whose last name begins with letters from M to Z.

***The goal of this question is to produce a select with three columns where the list price is between 500 and 2000 (not inclusive) ordered by “date\_added” descending.***

1. Write a SELECT statement that returns these columns from the Products table:

product\_name The product\_name column

list\_price The list\_price column

date\_added The date\_added column

Return only the rows with a list price that’s greater than 500 and less than 2000.

Sort the result set in descending sequence by the date\_added column.

***The goal of this question is to select five columns, CALCULATE two of them into a single column using an ALIAS, use that ALIAS to CALCULATE another column, and display only the first 5 rows ordered by “discount\_price” descending. The SELECT used with an ALIAS is to save time when performing the math. You will NOT get full credit for this question if you compute “discount\_amount twice”!***

1. Write a SELECT statement that returns these column names and data from the Products table:

product\_name The product\_name column

list\_price The list\_price column

discount\_percent The discount\_percent column

discount\_amount A column that’s calculated from the previous two columns

discount\_price A column that’s calculated from the previous three columns

Round the discount\_amount and discount\_price columns to 2 decimal places.

Sort the result set by discount price in descending sequence.

Use the LIMIT clause so the result set contains only the first 5 rows.

***The goal of this question is to select seven columns and return only the rows HAVING an “item\_total” greater than 500 ordered by “item\_total” descending. In this question, you must use the word HAVING and an ALIAS. HAVING has two purposes: In this question, the purpose of HAVING is to substitute for the WHERE clause. The reason for that is that the WHERE clause cannot have an ALIAS. In MySQL, physically where you place the HAVING, changes the meaning of the word HAVING.***

1. Write a SELECT statement that returns these column names and data from the Order\_Items table:

item\_id The item\_id column

item\_price The item\_price column

discount\_amount The discount\_amount column

quantity The quantity column

price\_total A column that’s calculated by multiplying the item price by the quantity

discount\_total A column that’s calculated by multiplying the discount amount by the quantity

item\_total A column that’s calculated by subtracting the discount amount from the item price and then multiplying by the quantity

Only return rows where the item\_total is greater than 500.

Sort the result set by item total in descending sequence.

***The goal of this question is to select three columns and return only the rows where “ship\_date” contains a NULL value.***

1. Write a SELECT statement that returns these columns from the Orders table:

order\_id The order\_id column

order\_date The order\_date column

ship\_date The ship\_date column

Return only the rows where the ship\_date column contains a null value.

***The goal of this question is to produce a select with the NOW() function. You will have two columns: One called “today\_unformatted” and one called “today\_formatted”.***

1. Write a SELECT statement without a FROM clause that uses the NOW function to create a row with these columns:

today\_unformatted The NOW function unformatted

today\_formatted The NOW function in this format:   
DD-Mon-YYYY

This displays a number for the day, an abbreviation for the month, and a four-digit year.

***The goal of this question is to produce a select with four columns. You will have to perform two CALCULATIONS using some of the columns. This requires an ALIAS (similar to question 4).***

1. Write a SELECT statement without a FROM clause that creates a row with these columns:

price 100 (dollars)

tax\_rate .07 (7 percent)

tax\_amount The price multiplied by the tax

total The price plus the tax

To calculate the fourth column, add the expressions you used for the first and third columns.

Retrieve Data from Two or More Tables

**Please note: Any time you JOIN tables, you must also use the USING or ON keyword to join them properly!**

***The goal of this question is to produce a select that joins two tables together and returns three columns. You will then order the output by both the “category\_name” and “product\_name”.***

1. Write a SELECT statement that joins the Categories table to the Products table and returns these columns: category\_name, product\_name, list\_price.

Sort the result set by category\_name and then by product\_name in ascending sequence.

***The goal of this question is to produce a select that joins two tables and returns six columns. It must return one row for each address where the email address is “allan.sherwood@yahoo.com”.***

1. Write a SELECT statement that joins the Customers table to the Addresses table and returns these columns: first\_name, last\_name, line1, city, state, zip\_code.

Return one row for each address for the customer with an email address of allan.sherwood@yahoo.com.

***The goal of this question is to produce a select that joins two tables and returns six columns. It must return one row for each customer, but only return addresses that are also the “shipping\_address” for a customer.***

1. Write a SELECT statement that joins the Customers table to the Addresses table and returns these columns: first\_name, last\_name, line1, city, state, zip\_code.

Return one row for each customer, but only return addresses that are the shipping address for a customer.

***The goal of this question is to produce a select that joins four tables and returns seven columns. You must use ALIASES for the tables! The output will then be ordered by “Last\_name”, “order\_date”, and “product\_name”.***

1. Write a SELECT statement that joins the Customers, Orders, Order\_Items, and Products tables. This statement should return these columns: last\_name, first\_name, order\_date, product\_name, item\_price, discount\_amount, and quantity.

Use aliases for the tables.

Sort the final result set by last\_name, order\_date, and product\_name.

***The goal of this question is to produce a select that returns the “product\_name” and “list\_price”. It must return one row for each product that has the same “list\_price” as another product (this is called a self-join).***

1. Write a SELECT statement that returns the product\_name and list\_price columns from the Products table.

Return one row for each product that has the same list price as another product.   
*Hint: Use a self-join to check that the product\_id columns aren’t equal but the list\_price columns are equal.*

Sort the result set by product\_name.

***The goal of this question is to produce a select that returns two columns. It must return one row where the “product\_id” contains a NULL value.***

1. Write a SELECT statement that returns these two columns:

category\_name The category\_name column from the Categories table

product\_id The product\_id column from the Products table

Return one row for each category that has never been used. *Hint: Use an outer join and only return rows where the product\_id column contains a null value*