How to use MySQL Workbench and other development tools

Before you start the exercises…

Before you start these exercises, you need to install the MySQL server and MySQL Workbench.

In addition, you'll need to get the mgs_ex_starts directory from your instructor. This directory contains some script files that you need to do the exercises.

Exercises

In these exercises, you'll use MySQL Workbench to create the My Guitar Shop database, to review the tables in this database, and to enter SQL statements and run them against this database.

Make sure the MySQL server is running

1. Start MySQL Workbench and open a connection for managing the server.
2. Check whether the MySQL server is running. If it isn't, start it. When you're done, close the Admin tab.

Use MySQL Workbench to create the My Guitar Shop database

3. Open a connection to start querying.
4. Open the script file named create_my_guitar_shop.sql that's in the mgs_ex_starts directory by clicking the Open SQL Script File button in the SQL Editor toolbar. Then, use the resulting dialog box to locate and open the file.
5. Execute the entire script by clicking the Execute SQL Script button in the code editor toolbar or by pressing Ctrl+Shift+Enter. When you do, the Output window displays messages that indicate whether the script executed successfully.

Use MySQL Workbench to review the My Guitar Shop database

6. In the Object Browser window, expand the node for the database named my_guitar_shop so you can see all of the database objects it contains. If it isn't displayed in the Object Browser window, you may need to click on the Refresh button to display it.
7. View the data for the Categories and Products tables.
8. Navigate through the database objects and view the column definitions for at least the Categories and Products tables.

Use MySQL Workbench to enter and run SQL statements

9. Double-click on the my_guitar_shop database to set it as the default database. When you do that, MySQL Workbench should display the database in bold.
10. Open a code editor tab. Then, enter and run this SQL statement:

```
SELECT product_name FROM products
```

11. Delete the *e* at the end of product_name and run the statement again. Note the error number and the description of the error.
12. Open another code editor tab. Then, enter and run this statement:

```
SELECT COUNT(*) AS number_of_products
FROM products
```

Use MySQL Workbench to open and run scripts

13. Open the script named product_details.sql that's in the mgs_ex_starts directory. Note that this script contains just one SQL statement. Then, run the statement.
14. Open the script named product_summary.sql that's in the mgs_ex_starts directory. Note that this opens another code editor tab.
15. Open the script named product_statements.sql that's in the mgs_ex_starts directory. Notice that this script contains two SQL statements that end with semicolons.
16. Press the Ctrl+Shift+Enter keys or click the Execute SQL Script button to run both of the statements in this script. Note that this displays the results in two Result tabs. Make sure to view the results of both SELECT statements.
17. Move the insertion point into the first statement and press Ctrl+Enter to run just that statement.
18. Move the insertion point into the second statement and press Ctrl+Enter to run just that statement.
19. Exit from MySQL Workbench.

How to retrieve data from a single table

Exercises

Enter and run your own SELECT statements

In these exercises, you'll enter and run your own SELECT statements.

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.

2. 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.

3. 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.

4. 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.

5. 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.

Work with nulls and test expressions

6. 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.

7. 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.

8. 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.

How to retrieve data from two or more tables

Exercises

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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.*

7. Use the UNION operator to generate a result set consisting of three columns from the Orders table:

ship_status	A calculated column that contains a value of SHIPPED or NOT SHIPPED
order_id	The order_id column
order_date	The order_date column

If the order has a value in the ship_date column, the ship_status column should contain a value of SHIPPED. Otherwise, it should contain a value of NOT SHIPPED.

Sort the final result set by order_date.

How to code summary queries

Exercises

1. Write a SELECT statement that returns these columns:

   The count of the number of orders in the Orders table

   The sum of the tax_amount columns in the Orders table

2. Write a SELECT statement that returns one row for each category that has products with these columns:

   The category_name column from the Categories table

   The count of the products in the Products table

   The list price of the most expensive product in the Products table

   Sort the result set so the category with the most products appears first.

3. Write a SELECT statement that returns one row for each customer that has orders with these columns:

   The email_address column from the Customers table

   The sum of the item price in the Order_Items table multiplied by the quantity in the Order_Items table

   The sum of the discount amount column in the Order_Items table multiplied by the quantity in the Order_Items table

   Sort the result set in descending sequence by the item price total for each customer.

4. Write a SELECT statement that returns one row for each customer that has orders with these columns:

   The email_address from the Customers table

   A count of the number of orders

   The total amount for each order (*Hint: First, subtract the discount amount from the price. Then, multiply by the quantity.*)

   Return only those rows where the customer has more than 1 order.

   Sort the result set in descending sequence by the sum of the line item amounts.

5. Modify the solution to exercise 4 so it only counts and totals line items that have an item_price value that's greater than 400.

6. Write a SELECT statement that answers this question: What is the total amount ordered for each product? Return these columns:

   The product name from the Products table

   The total amount for each product in the Order_Items (*Hint: You can calculate the total amount by subtracting the discount amount from the item price and then multiplying it by the quantity*)

   Use the WITH ROLLUP operator to include a row that gives the grand total.

   Note: Once you add the WITH ROLLUP operator, you may need to use MySQL Workbench's Execute SQL Script button instead of its Execute Current Statement button to execute this statement.

7. Write a SELECT statement that answers this question: Which customers have ordered more than one product? Return these columns:

   The email address from the Customers table

   The count of distinct products from the customer's orders

How to code subqueries

Exercises

1. Write a SELECT statement that returns the same result set as this SELECT statement, but don't use a join. Instead, use a subquery in a WHERE clause that uses the IN keyword.

```
SELECT DISTINCT category_name
FROM categories c JOIN products p
  ON c.category_id = p.category_id
ORDER BY category_name
```

2. Write a SELECT statement that answers this question: Which products have a list price that's greater than the average list price for all products?

   Return the product_name and list_price columns for each product.

   Sort the results by the list_price column in descending sequence.

3. Write a SELECT statement that returns the category_name column from the Categories table.

   Return one row for each category that has never been assigned to any product in the Products table. To do that, use a subquery introduced with the NOT EXISTS operator.

4. Write a SELECT statement that returns three columns: email_address, order_id, and the order total for each customer. To do this, you can group the result set by the email_address and order_id columns. In addition, you must calculate the order total from the columns in the Order_Items table.

   Write a second SELECT statement that uses the first SELECT statement in its FROM clause. The main query should return two columns: the customer's email address and the largest order for that customer. To do this, you can group the result set by the email_address.

5. Write a SELECT statement that returns the name and discount percent of each product that has a unique discount percent. In other words, don't include products that have the same discount percent as another product.

   Sort the results by the product_name column.

6. Use a correlated subquery to return one row per customer, representing the customer's oldest order (the one with the earliest date). Each row should include these three columns: email_address, order_id, and order_date.

How to insert, update, and delete data

Exercises

To test whether a table has been modified correctly as you do these exercises, you can write and run an appropriate SELECT statement.

1. Write an INSERT statement that adds this row to the Categories table:

   category_name: Brass

   Code the INSERT statement so MySQL automatically generates the category_id column.

2. Write an UPDATE statement that modifies the row you just added to the Categories table. This statement should change the product_name column to "Woodwinds", and it should use the category_id column to identify the row.

3. Write a DELETE statement that deletes the row you added to the Categories table in exercise 1. This statement should use the category_id column to identify the row.

4. Write an INSERT statement that adds this row to the Products table:

product_id:	The next automatically generated ID
category_id:	4
product_code:	dgx_640
product_name:	Yamaha DGX 640 88-Key Digital Piano
description:	Long description to come.
list_price:	799.99
discount_percent:	0
date_added:	Today's date/time.

   Use a column list for this statement.

5. Write an UPDATE statement that modifies the product you added in exercise 4. This statement should change the discount_percent column from 0% to 35%.

6. Write a DELETE statement that deletes the row that you added to the Categories table in exercise 1. When you execute this statement, it will produce an error since the category has related rows in the Products table. To fix that, precede the DELETE statement with another DELETE statement that deletes all products in this category. (Remember that to code two or more statements in a script, you must end each statement with a semicolon.)

7. Write an INSERT statement that adds this row to the Customers table:

email_address:	rick@raven.com
password:	(empty string)
first_name:	Rick
last_name:	Raven

   Use a column list for this statement.

8. Write an UPDATE statement that modifies the Customers table. Change the password column to "secret" for the customer with an email address of rick@raven.com.

9. Write an UPDATE statement that modifies the Customers table. Change the password column to "reset" for every customer in the table. If you get an error due to safe-update mode, you can add a LIMIT clause to update the first 100 rows of the table. (This should update all rows in the table.)

10. Open the script named create_my_guitar_shop.sql that's in the mgs_ex_starts directory. Then, run this script. That should restore the data that's in the database.

How to design a database

Exercises

1. Use MySQL Workbench to create an EER model from the script file named create_my_guitar_shop.sql that's in the mgs_ex_starts folder.

   From the model, create an EER diagram that shows the relationships between the seven tables in the database. (The administrators table is not related to the other six tables.)

2. Use MySQL Workbench to create an EER diagram for a database that stores information about the downloads that users make.

   Each user must have an email address, first name, and last name.

   Each user can have one or more downloads.

   Each download must have a filename and download date/time.

   Each product can be related to one or more downloads.

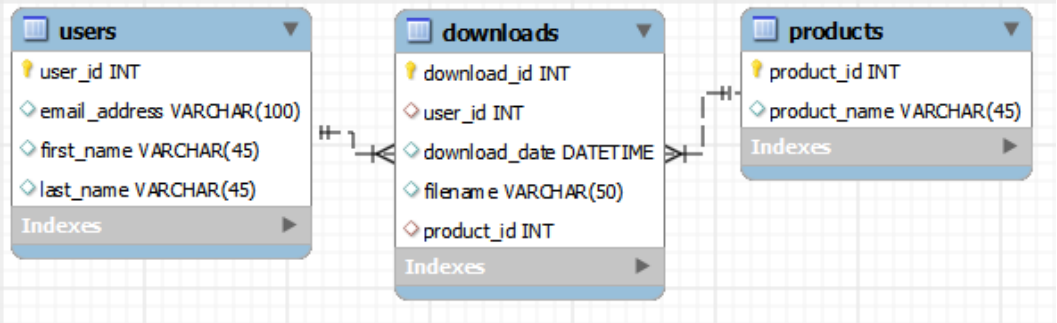
   Each product must have a name.

3. Use MySQL Workbench to open the EER diagram that you created in exercise 2. Then, export a script that creates the database and save this script in a file named ex10-3.sql. Next, use MySQL Workbench to open this file and review it.

How to create databases, tables, and indexes

Exercises

1. Write a script that adds an index to the my_guitar_shop database for the zip code field in the Customers table.
2. Write a script that implements the following design in a database named my_web_db:



   In the Downloads table, the user_id and product_id columns are the foreign keys.

   Include a statement to drop the database if it already exists.

   Include statements to create and select the database.

   Include any indexes that you think are necessary.

   Specify the utf8 character set for all tables.

   Specify the InnoDB storage engine for all tables.

3. Write a script that adds rows to the database that you created in exercise 2.

   Add two rows to the Users and Products tables.

   Add three rows to the Downloads table: one row for user 1 and product 2; one row for user 2 and product 1; and one row for user 2 and product 2. Use the NOW function to insert the current date and time into the download_date column.

   Write a SELECT statement that joins the three tables and retrieves the data from these tables like this:

email_address	first_name	last_name	download_date	filename	product_name
johnsmith@gmail.com	John	Smith	2012-04-24 16:15:38	pedals_are_falling.mp3	Local Music Vol 1
janedoe@yahoo.com	Jane	Doe	2012-04-24 16:15:38	turn_signal.mp3	Local Music Vol 1
janedoe@yahoo.com	Jane	Doe	2012-04-24 16:15:38	one_horse_town.mp3	Local Music Vol 2

   Sort the results by the email address in descending sequence and the product name in ascending sequence.