# Murach 4e Chapter 6 Coding Assignment Instructions

**Exercises to be Completed**
Complete exercises as follows:

- Exercise 1 through 8 are regular exercises.
- Exercises 9, 10, and 11 are challenge exercises.
- Note that you only need to complete one of the challenge exercises to earn points for the challenge.

**General Instructions**
My expectations for your work on coding assignment exercises will grow as we progress through the course. In addition to applying any new coding techniques that have been covered in the current chapter, I will be expecting you to follow all of the good practices that we have adopted in the preceding weeks. Here is a quick summary of good practices that we have covered so far:

- Begin each script file that accesses the database with a USE statement (e.g., USE my_guitar_shop;).
- Use the *beautify* feature of the MySQL Workbench to *pretty-print* your code.
- End each statement in your script with a semicolon.
- Use the SQL features requested in the exercise description and/or covered in the chapter.
- Always include an ORDER BY in SELECT statements unless directed otherwise. If the exercise instructions ask for a particular order, then use that. Otherwise, choose any reasonable order.
- In SELECT statements that use JOIN, always use the explicit (SQL-92) JOIN syntax implemented in the FROM clause. Do NOT use the implicit JOIN syntax implemented using the WHERE clause.
- Do NOT include extra or unnecessary code in the script.

**Tools**
Use MySQL Workbench to create and test all scripts.

**Submission Method**
Use the following process to submit your work for this assignment:

- Locate the properly named directory associated with your assignment in the file system (see *File and Directory Naming,* below).
- Compress that directory into a single .ZIP file using a utility program. NOTE: Only one file may be submitted. File types other than .ZIP will not be graded.
- Submit the properly named .ZIP file to the submission activity for this assignment.

**File and Directory Naming**
Please note that file and directory names must be in all lower case.  Deductions will be made for submissions that do not follow this standard.

Please use the following naming scheme for the directory that holds your scripts:

```
surname_givenname_mgs_chap_06
```

If this were my own project, I would name my PyCharm project as follows:

```
trainor_kevin_mgs_chap_06
```

A separate solution script file must be submitted for each exercise.  Solution scripts must be named using the following form: ex_xx_yy.sql (where xx is the two-digit chapter number [04] and yy is the two-digit exercise number [01]). So, an example of a properly formed solution script file example would be:

```
ex_06_01.sql
```

Use a zip utility to create one zip file that contain the PyCharm project directory. The zip file should be named according to the following scheme:

```
surname_givenname_mgs_chap_06.zip
```

If this were my own project, I would name the zip file as follows:

```
trainor_kevin_mgs_chap_06.zip
```

**Due By**
Please submit this assignment by the date and time shown in the Weekly Schedule.

**Last Revised**
2025-03-10

## Please see the exercises on the attached sheets!

# Chapter 6

# 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 column 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 column from the Products table

   The total amount for each product in the Order_Items table (*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 column from the Customers table

   The count of distinct products from the customer's orders

   Sort the result set in ascending sequence by the email_address column.

8. Write a SELECT statement that answers this question: What is the total quantity purchased for each product within each category? Return these columns:

   The category_name column from the category table

   The product_name column from the products table

   The total quantity purchased for each product with orders in the Order_Items table

   Use the WITH ROLLUP operator to include rows that give a summary for each category name as well as a row that gives the grand total.

   Use the IF and GROUPING functions to replace null values in the category_name and product_name columns with literal values if they're for summary rows.

9. [*Challenge Exercise*] Write a SELECT statement that uses an aggregate window function to get the total amount of each order. Return these columns:

   The order_id column from the Order_Items table

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

   The total amount for each order

   Sort the result set in ascending sequence by the order_id column.

10. [*Challenge Exercise*] Modify the solution to exercise 9 so the column that contains the total amount for each order contains a cumulative total by item amount.

    Add another column to the SELECT statement that uses an aggregate window function to get the average item amount for each order.

    Modify the SELECT statement so it uses a named window for the two aggregate functions.

11. [*Challenge Exercise*] Write a SELECT statement that uses aggregate window functions to calculate the order total for each customer and the order total for each customer by date. Return these columns:

    The customer_id column from the Orders table

    The order_date column from the Orders table

    The total amount for each order item in the Order_Items table

    The sum of the order totals for each customer

    The sum of the order totals for each customer by date *(Hint: You can create a peer group to get these values)*