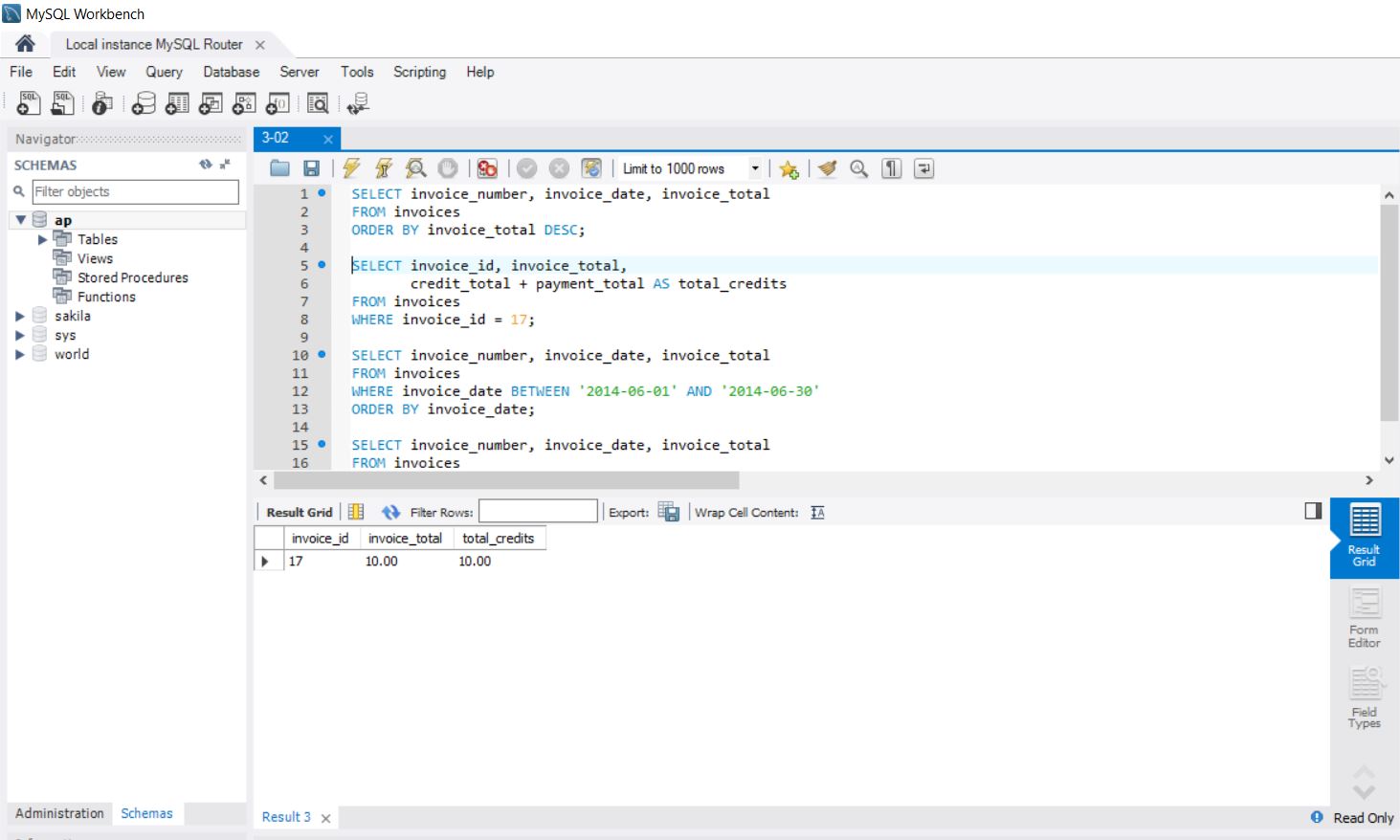
**Exercises**

In these exercises, you’ll use MySQL Workbench to review the tables in the AP database. In addition, you’ll use the MySQL Workbench to enter SQL statements and run them against these tables.

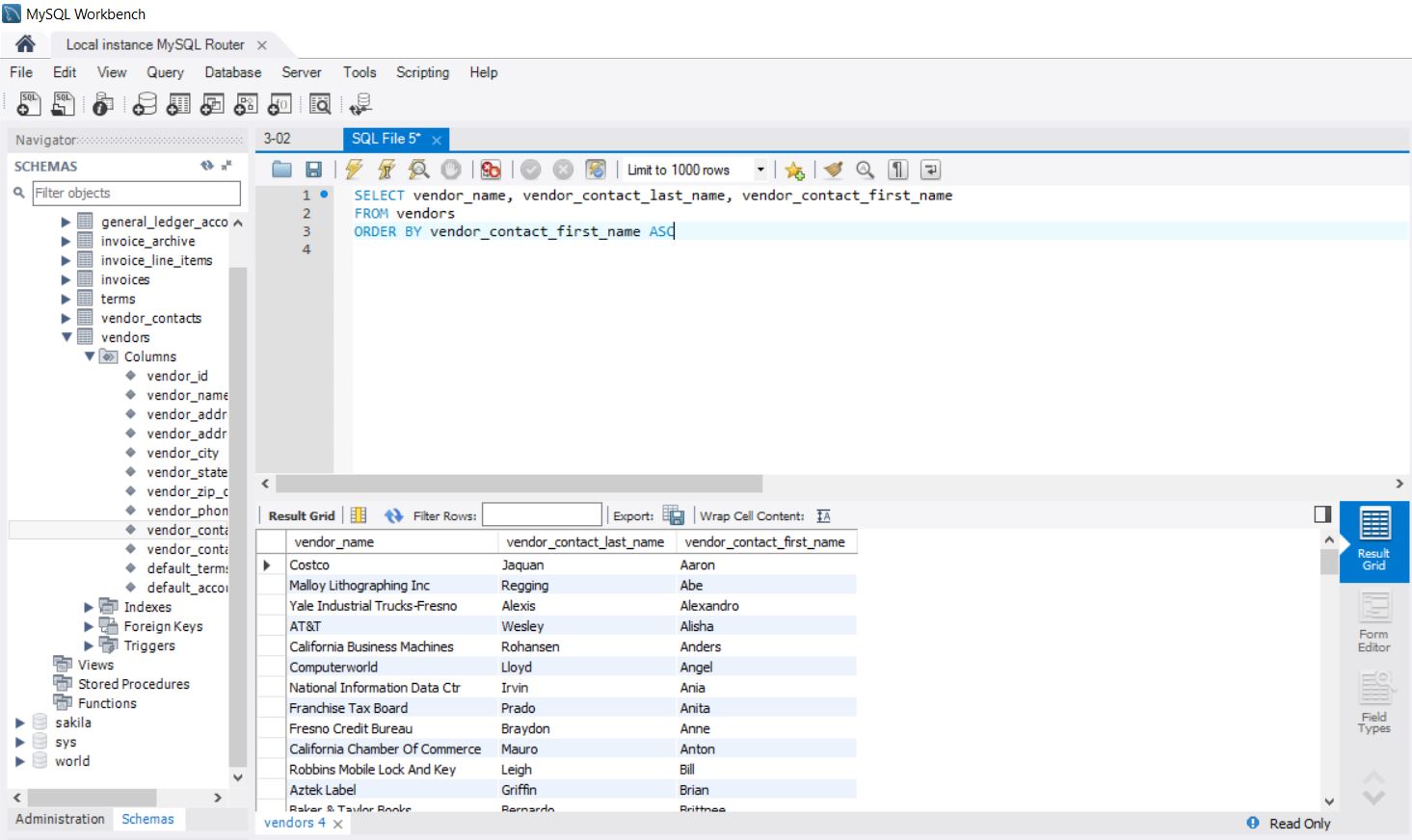
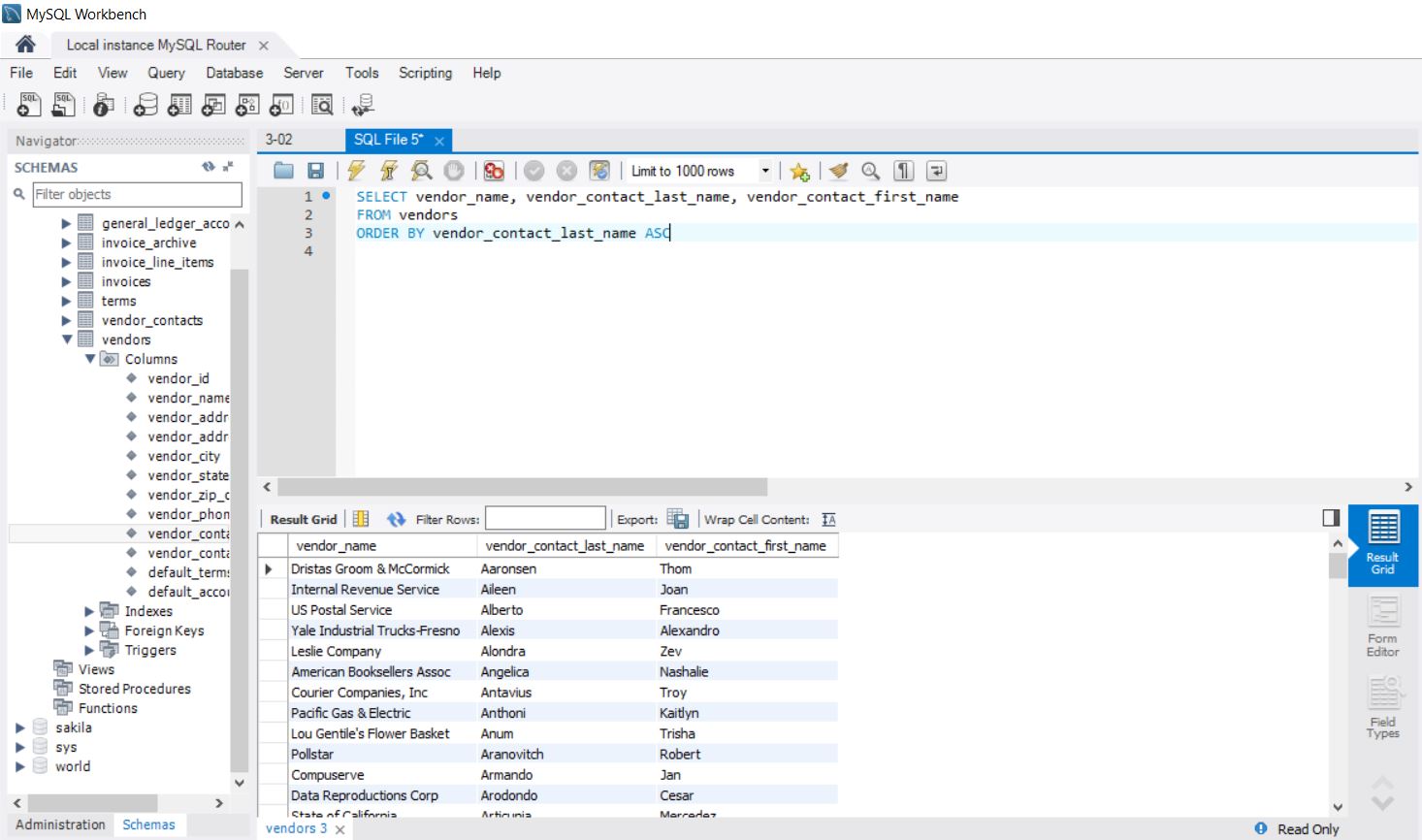
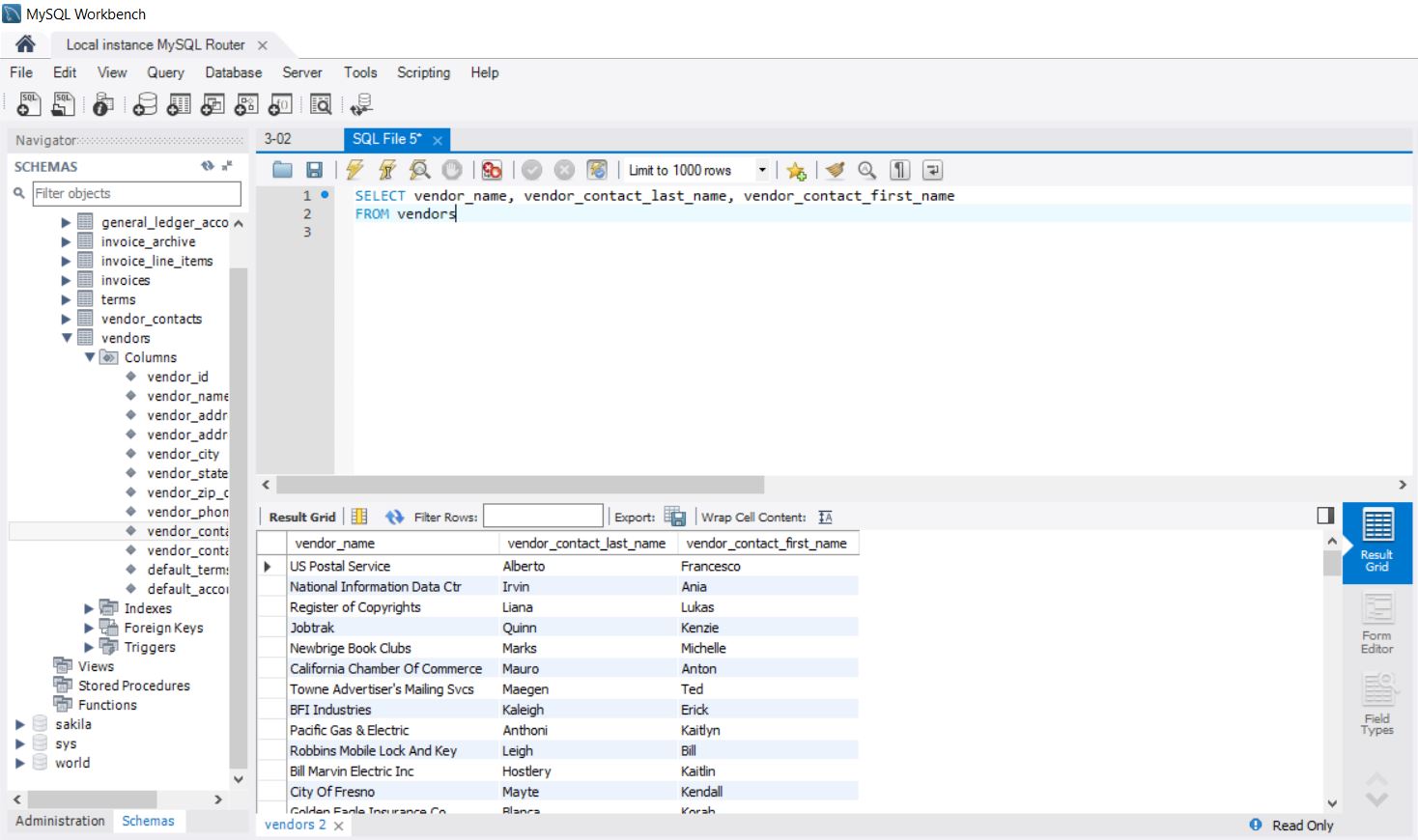
1. Start MySQL Workbench.
2. Open the query named 3-02.sql that you should find in this directory: *c:\murach\mysql\book\_scripts\ch03*. When it opens, you should see all of the queries for figure 3-2. Note that each of these queries has a semicolon at the end of it.
3. Move the insertion point into the first query and press Ctrl+Enter or click on the Execute Current Statement button to run the query. This shows you the date that’s in the Invoices table that you’ll be working with in this chapter.
4. Move the insertion point into the second query and run it.
5. Open and run the queries for any of the other examples in this chapter that you’re interested in reviewing.



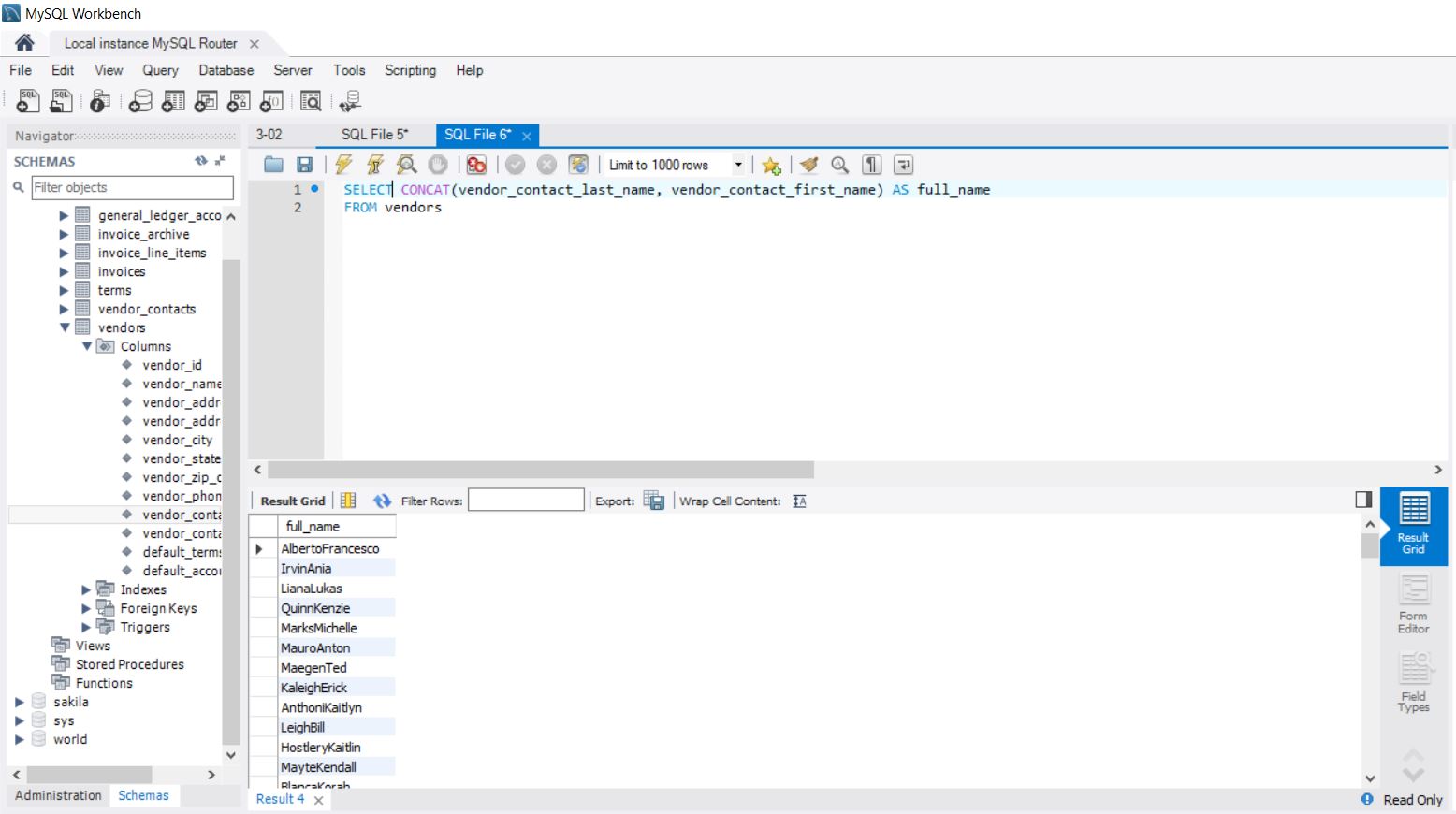
**Enter and run your own SELECT statements**

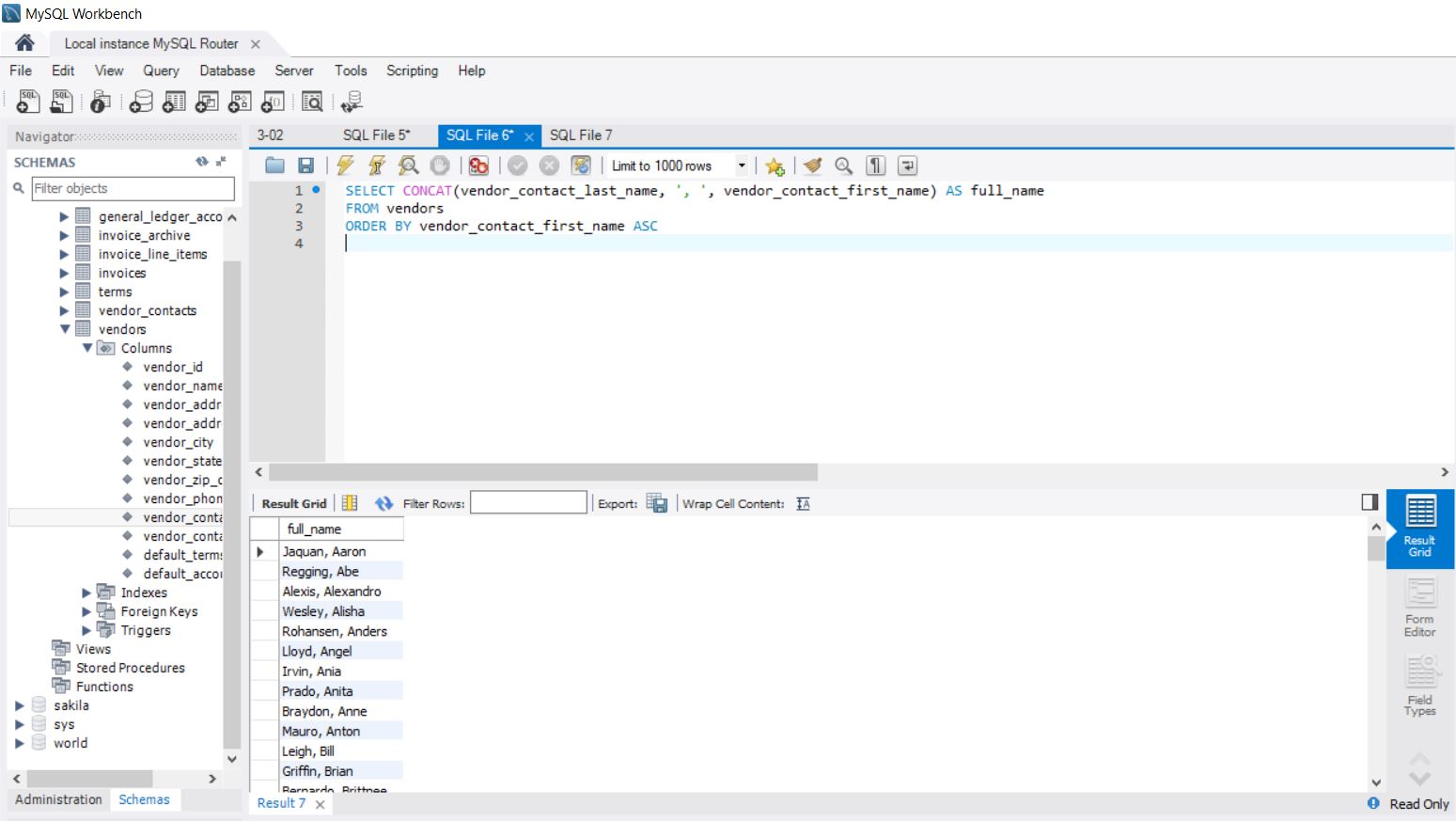
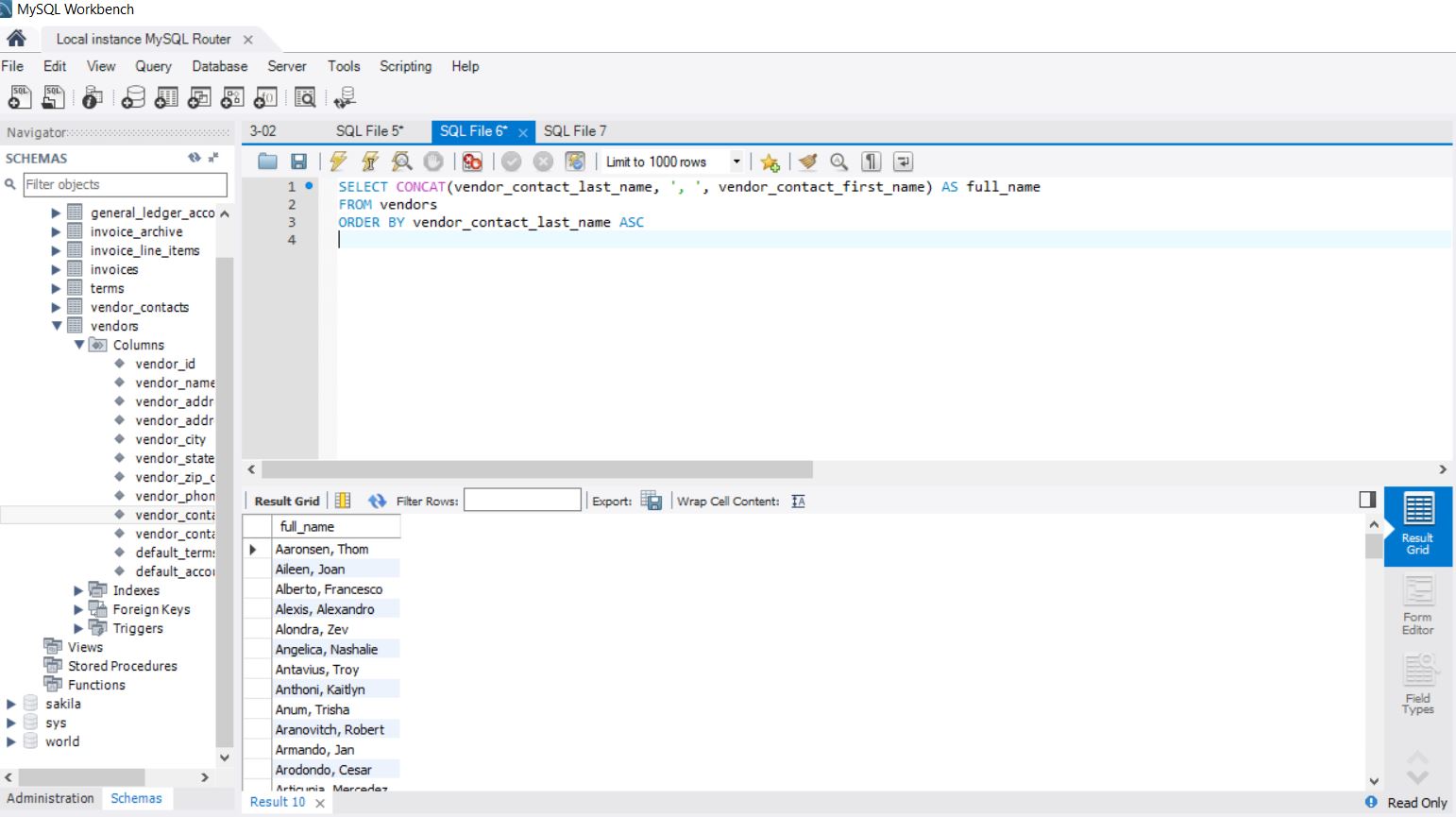
In these exercises, you’ll enter and run your own SELECT statements. To do that, you can open the query for an example that is similar to the statement you need to write, copy the statement into a new SQL tab, and modify the statement. That can save you both time and syntax errors.

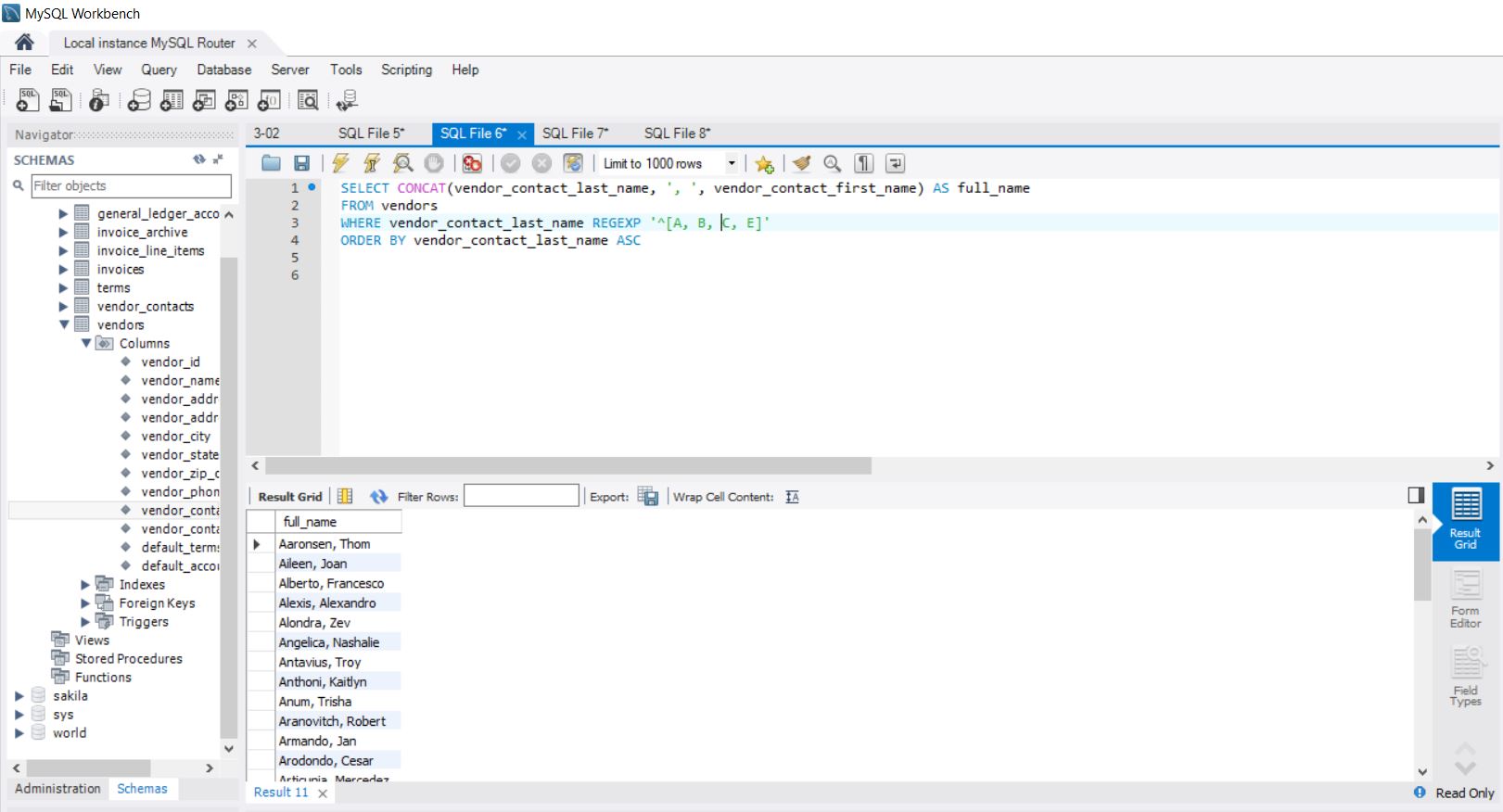
1. Write a **SELECT** statement that returns three columns from the Vendors table: *vendor\_name, vendor\_contact\_last\_name*, and *vendor\_contact\_first\_name*. Then, run this statement to make sure it works correctly.
   1. Add an **ORDER BY** clause to this statement that sorts the result set by last name and then first name, both in ascending 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.



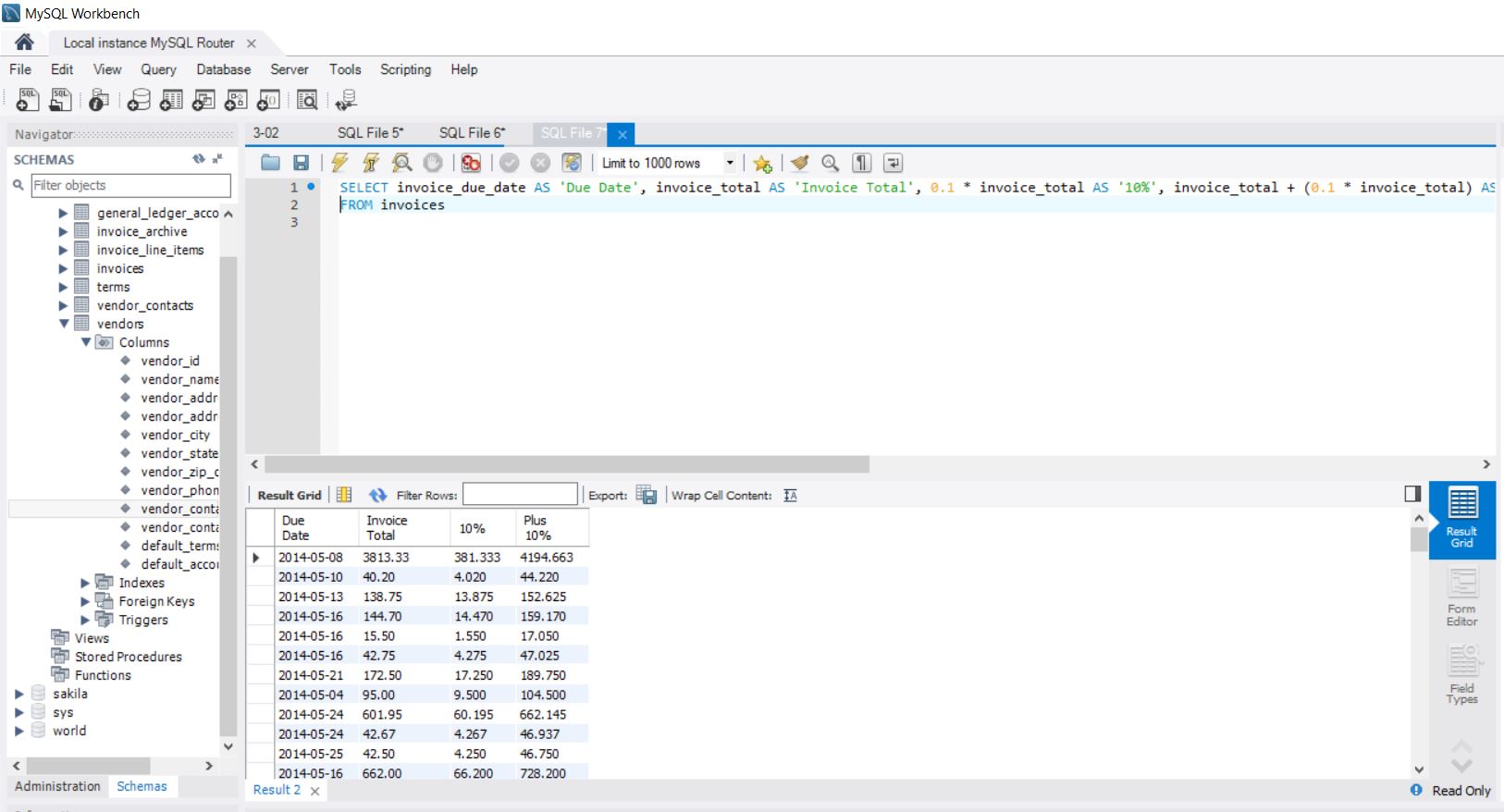
1. Write a **SELECT** statement that returns one column from the **Vendors** table named *full\_name* that joins the *vendor\_contact\_last\_name* and *vendor\_contact\_first\_name* columns.
   1. Format this column with the last name, a comma, a space, and the first name like this:
      1. **Doe, John**
   2. Sort the result set by last name and then first name in ascending sequence.
   3. Return only the contacts whose last name begins with the letter A, B, C, or E. This should retrieve 41 rows.

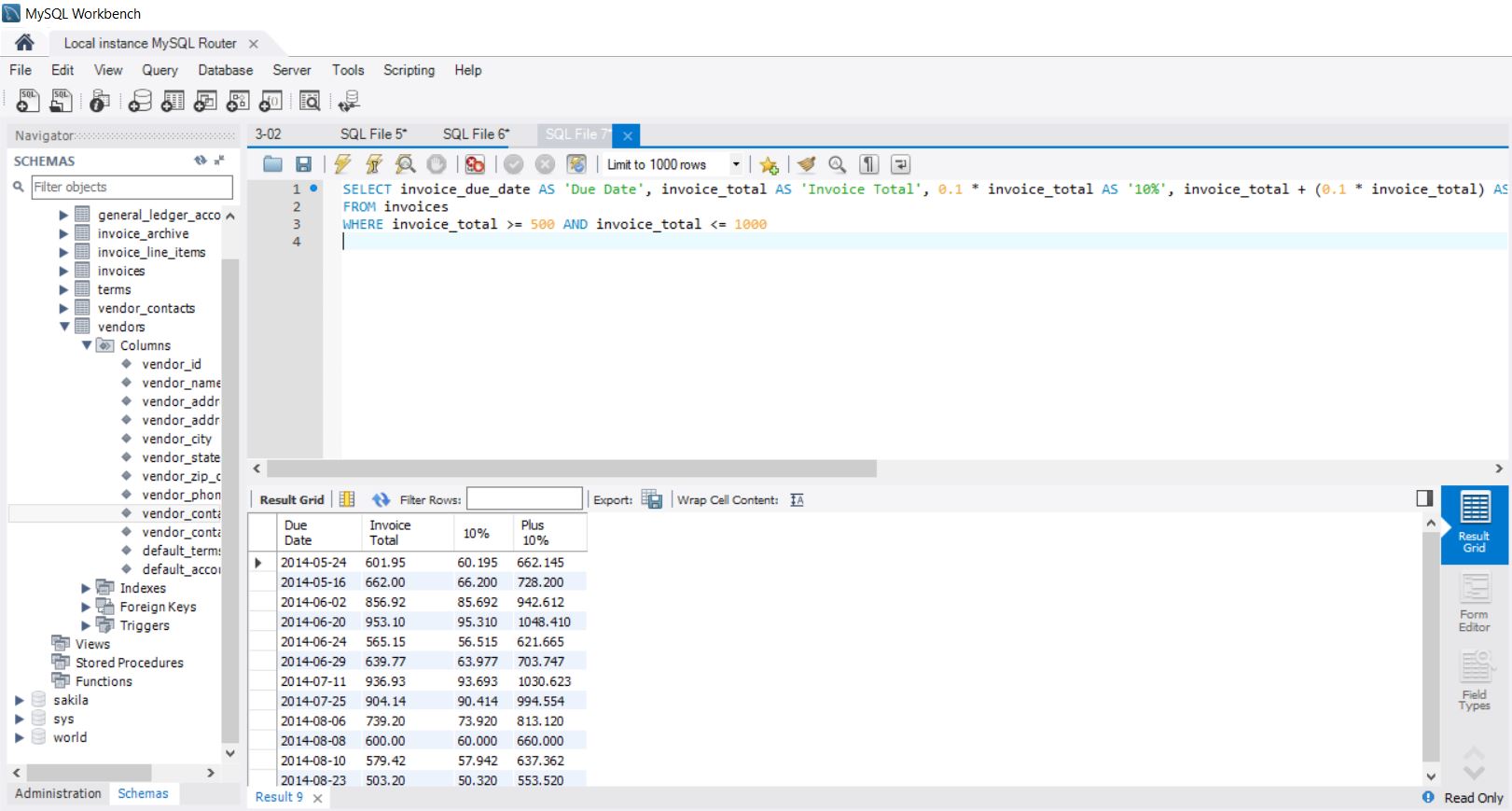


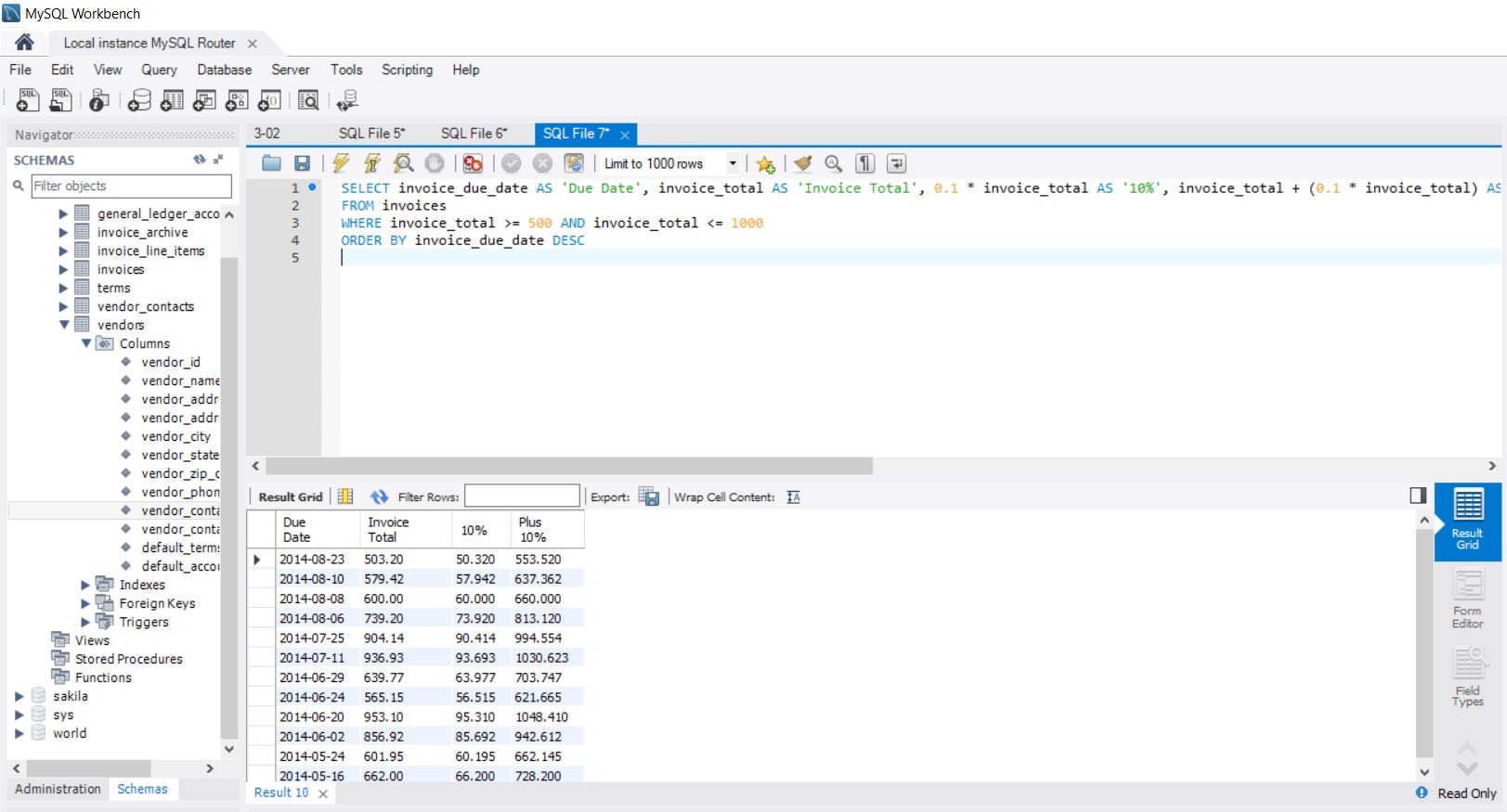




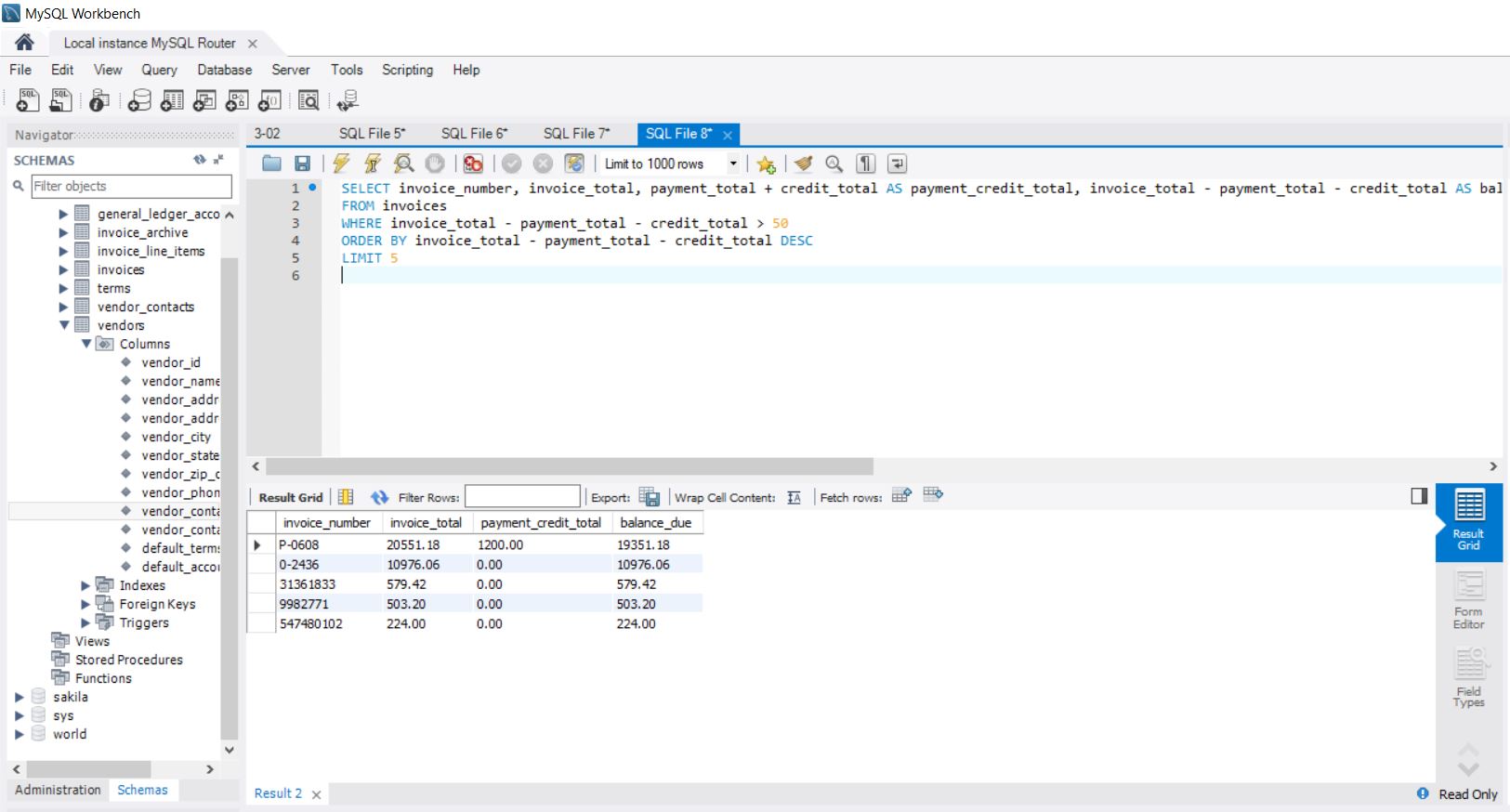
1. Write a **SELECT** statement that returns these column names and data from the Invoices table:
   * 1. Due Date—The *invoice\_due\_date* column
     2. Invoice Total—The *invoice\_total* column
     3. 10%—10% of the value of *invoice\_total*
     4. Plus 10%—The value of *invoice\_total* plus 10%
     5. Return only rows with an invoice total that’s greater than or equal to 500 and less than or equal to 1000. This should retrieve 12 rows.
   1. Sort the result set in descending sequence by *invoice\_due\_date.*





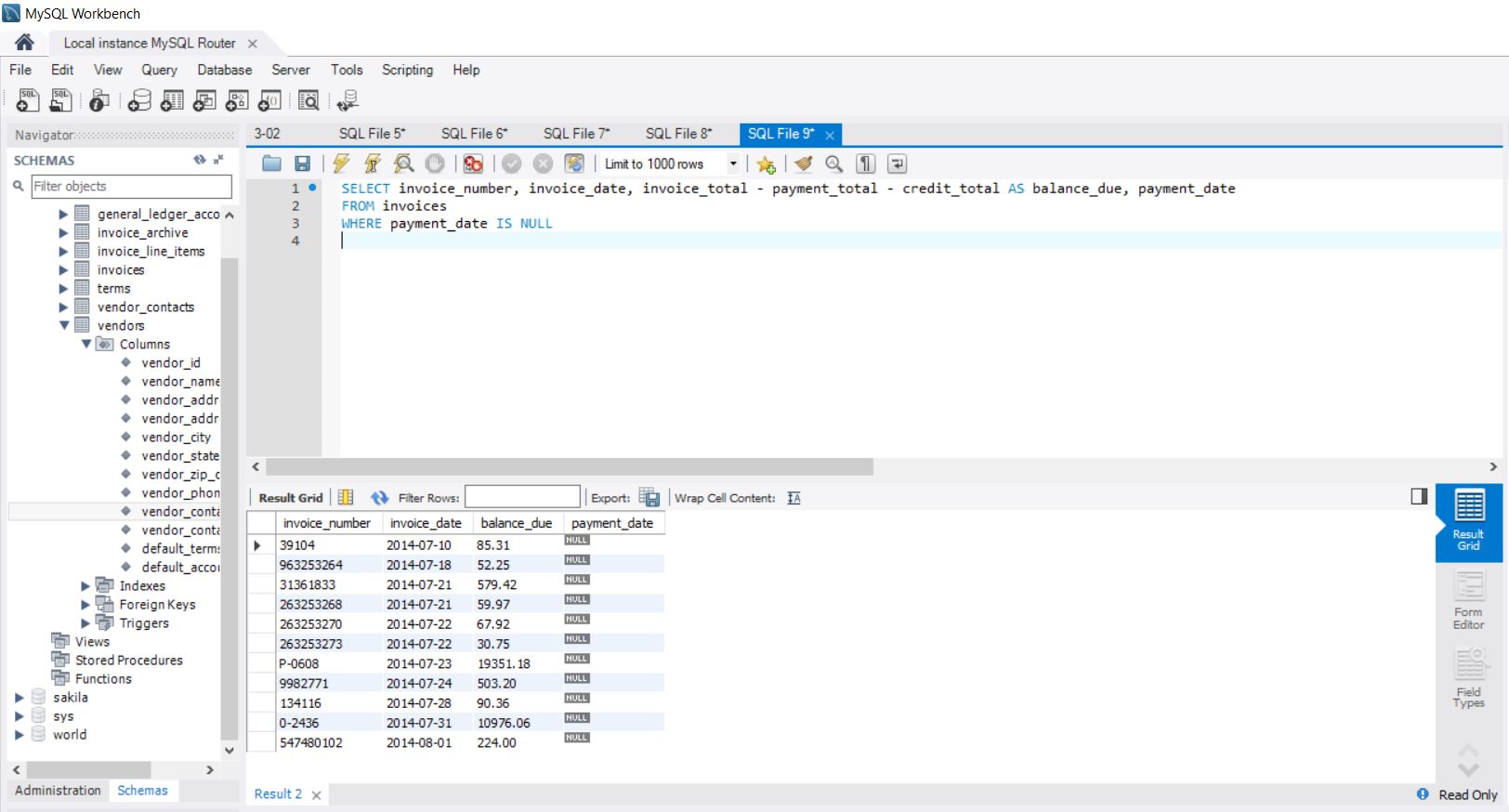


1. Write a SELECT statement that returns these columns from the Invoices table:
   * 1. Invoice\_number—The invoice\_number column
     2. Invoice\_total—The invoice\_total column
     3. Payment\_credit\_total—Sum of the payment\_total and credit\_total columns
     4. Balance\_due—The invoice\_total column minus the payment\_total and credit\_total columns
   1. Return only invoices that have a balance due that’s greater than $50.
   2. Sort the result set by balance due in descending sequence.
   3. Use the LIMIT clause so the result set contains only the rows with the 5 largest balances.

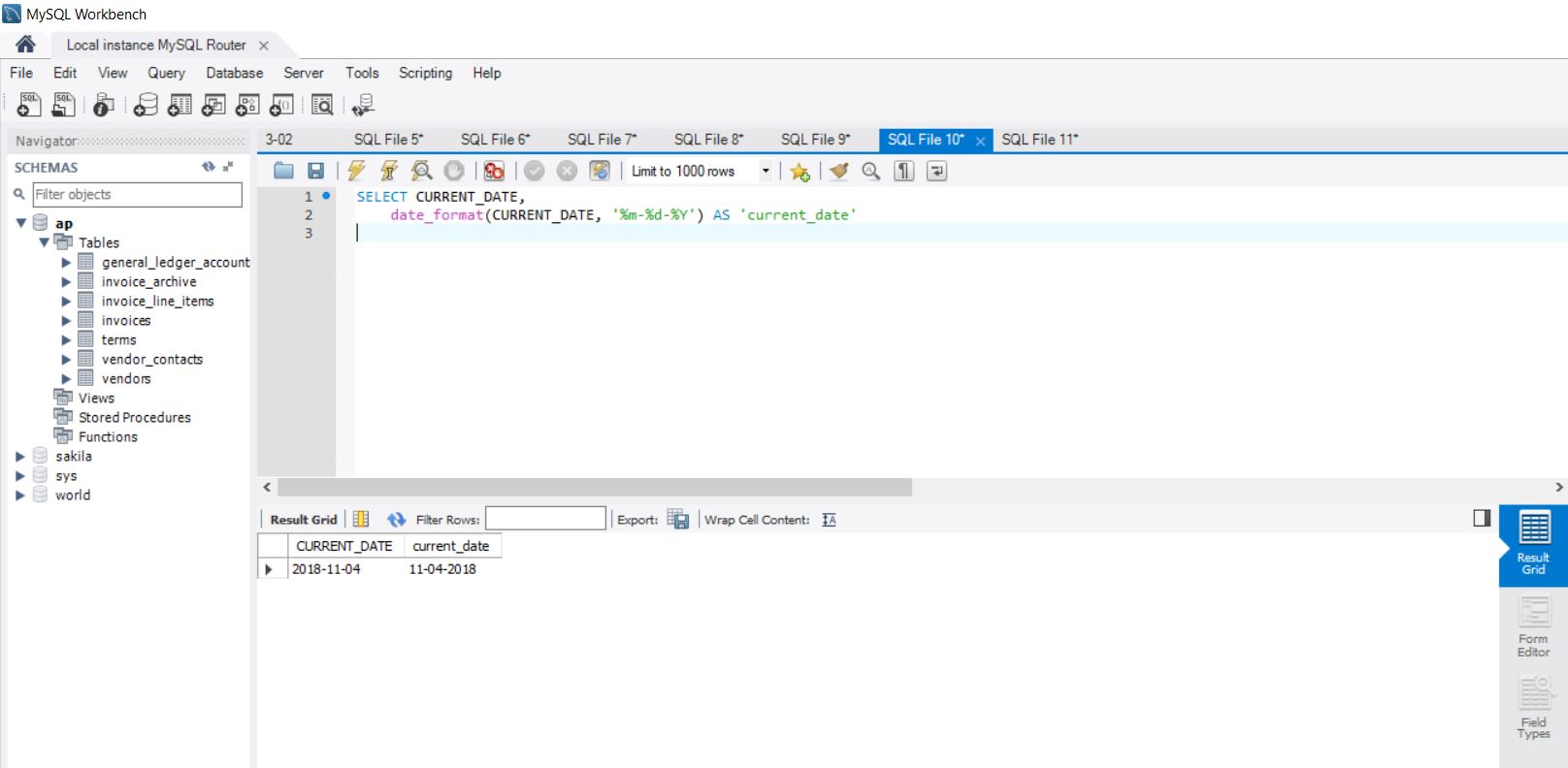


**Work with nulls and test expressions**

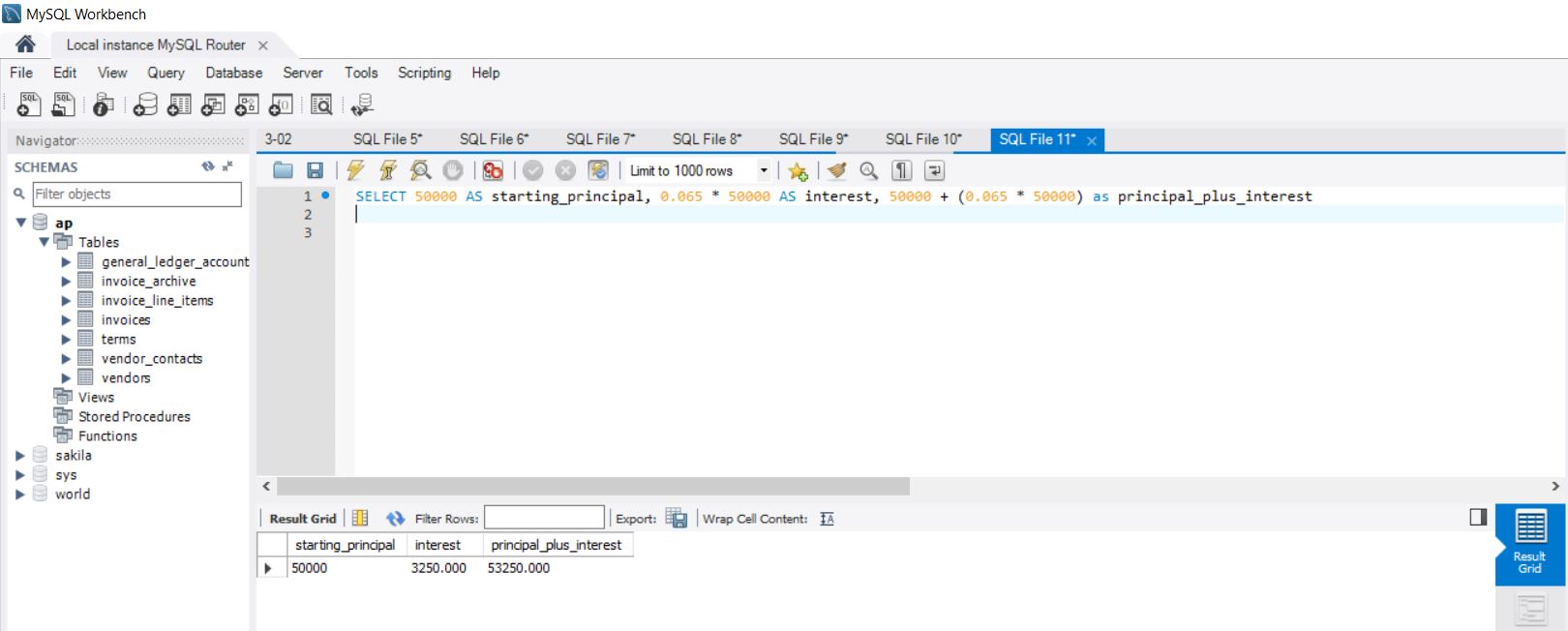
1. Write a **SELECT** statement that returns these columns from the Invoices table:
   * 1. Invoice\_number—The *invoice\_number* column
     2. Invoice\_date—The *invoice\_date* column
     3. Balance\_due—The *invoice\_total* column minus the *payment\_total* and *credit\_total* columns
     4. Payment\_date—The *payment\_date* column
   1. Return only the rows where the *payment\_date* column contains a null value. This should retrieve 11 rows.



1. Write a **SELECT** statement without a **FROM** clause that uses the **CURRENT\_DATE** function to return the current date in its default format.
   1. Use the **DATE\_FORMAT** function to format the current date in this format:
      1. **Mm-dd-yyyy**
      2. This displays the month, day, and four-digit year of the current date.
   2. Give this column an alias of *current\_date*. To do that, you must enclose the alias in quotes since that name is already used by the **CURRENT\_DATE** function.

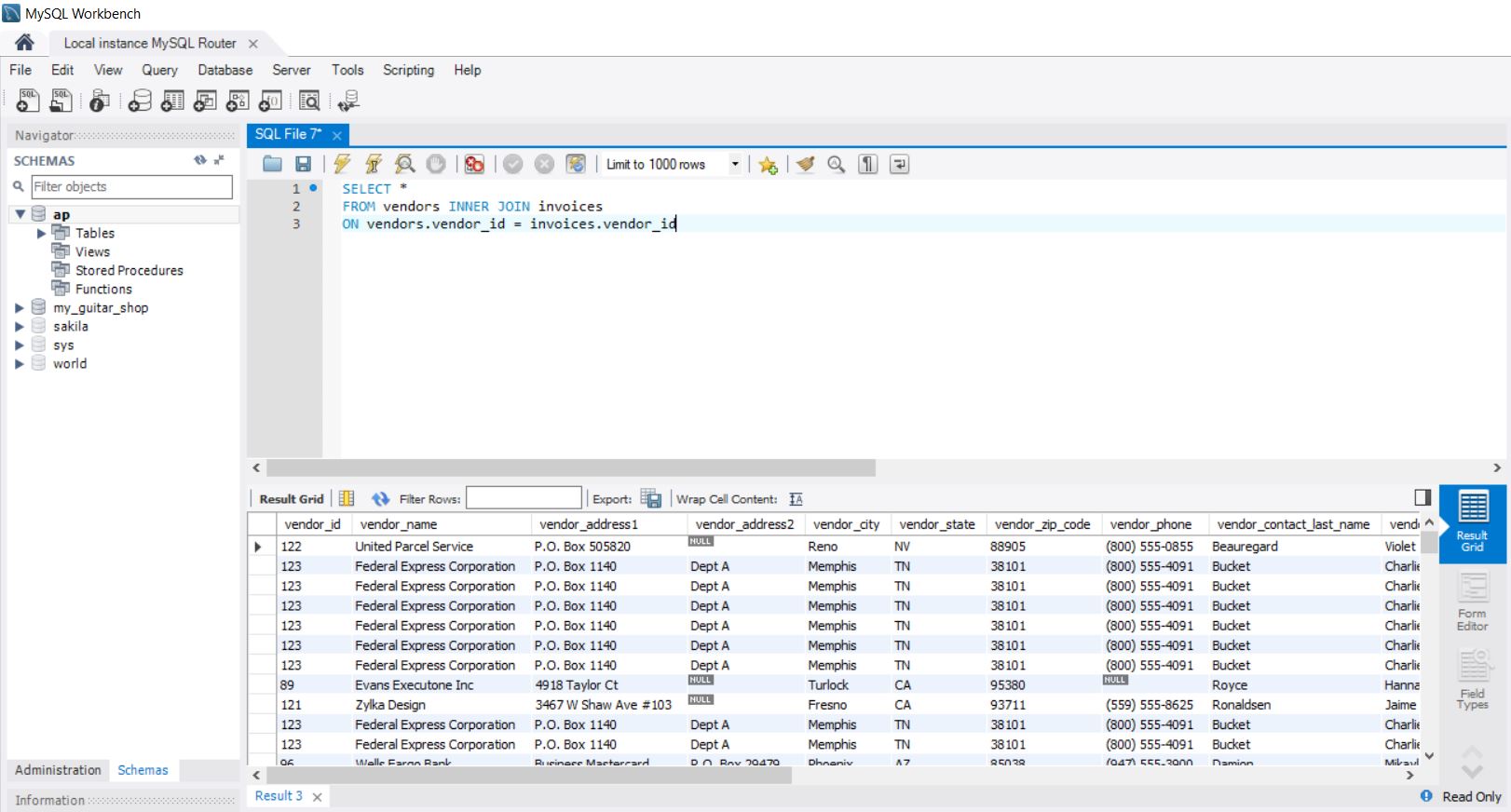


1. Write a **SELECT** statement without a FROM clause that creates a row with these columns:
   * 1. Starting\_principal—Starting principal of $50,000
     2. Interest—6.5% of the principal
     3. Principal\_plus\_interest—The principal plus the interest
   1. To calculate the third column, add the expressions you used for the first two columns.

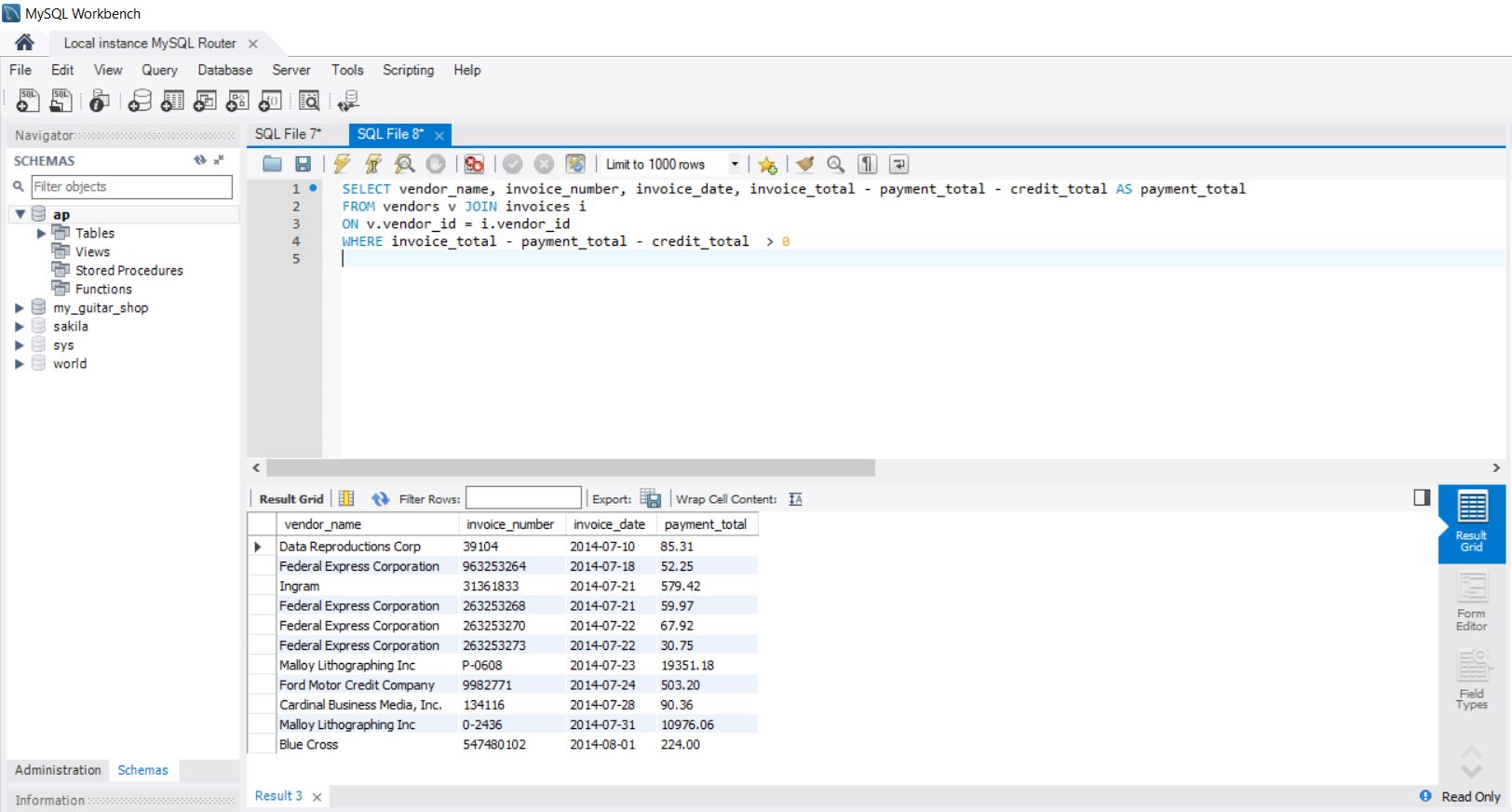


**Exercises**

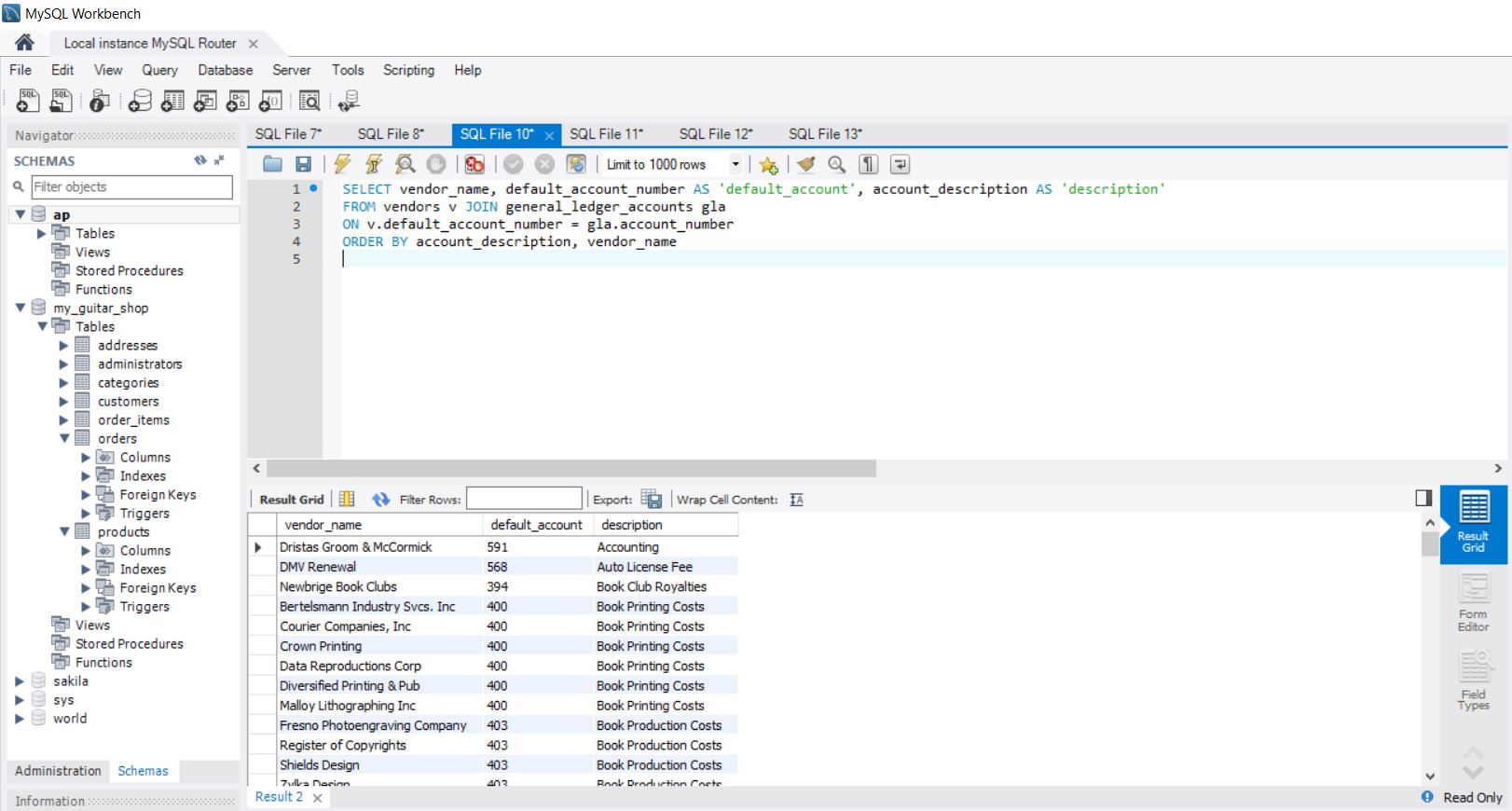
1. Write a SELECT statement that returns all columns from the Vendors table inner-joined with all columns from the Invoices table. This should return 114 rows. Hint: You can use an asterisk (\*) to select the columns from both tables.



1. Write a SELECT statement that returns these four columns:
   * 1. Vendor\_name - The vendor\_name column from the Vendors table
     2. Invoice\_number - The invoice\_number column from the Invoices table
     3. Invoice\_date - The invoice\_date column from the Invoices table
     4. Balance\_due - The invoice\_total column minus the payment\_total and credit\_total columns from the invoices table
   1. Use these aliases for the tables: v for Vendors and I for invoices.
   2. Return one row for each invoice with a non-zero balance. This should return 11 rows.



1. Write a SELECT statement that returns these three columns:
   * 1. Vendor\_name - The vendor\_name column from the Vendors table
     2. Default\_account - The default\_account\_number column from the Vendors table
     3. Description - The account\_description column from the General\_Ledger\_Accounts table
   1. Return one row for each vendor. This should return 122 rows.
   2. Sort the result by account\_description and then by vendor\_name.



1. Write a SELECT statement that returns these five columns:
   * 1. Vendor\_name - The vendor\_name column from the Vendors table
     2. Invoice\_date - The invoice\_date column from the Invoices table
     3. Invoice\_number - The invoice\_number column from the Invoices table
     4. Li\_sequence - The invoice\_sequence column from the Invoice\_Line\_Items table
     5. Li\_amount - The line\_item\_amount column from the Invoice\_Line\_Items table
   1. Use aliases for the tables. They should return 118 rows.
   2. Sort the final result set by vendor\_name, invoice\_date, invoice\_number, and invoice\_sequence.

SELECT vendor\_name, invoice\_date, invoice\_number, invoice\_sequence AS li\_sequence, line\_item\_amount AS li\_amount

FROM vendors v JOIN invoices i

ON v.vendor\_id = i.vendor\_id

JOIN invoice\_line\_items li

ON i.something = li.something

ORDER BY vendor\_name, invoice\_date, invoice\_number, invoice\_sequence

1. Write a SELECT sequence that returns three columns:
   * 1. Account\_number - The vendor\_ud column from the Vendors table
     2. Vendor\_name - The vendor\_name column from the Vendors table
     3. Contact\_name - A concatenation of the vendor\_contact\_first\_name and vendor\_contact\_last\_name columns with a space between
   1. Return one row for each vendor whose contact has the same name as another vendor’s contact. This should return 2 rows. Hint: Use a self-join to check that the vendor\_id columns aren’t equal but the vendor\_contact\_last\_name columns are equal.
   2. Sort the result set by vendor\_contact\_last\_name.
2. Write a SELECT statement that returns these three columns:
   * 1. Account\_number -