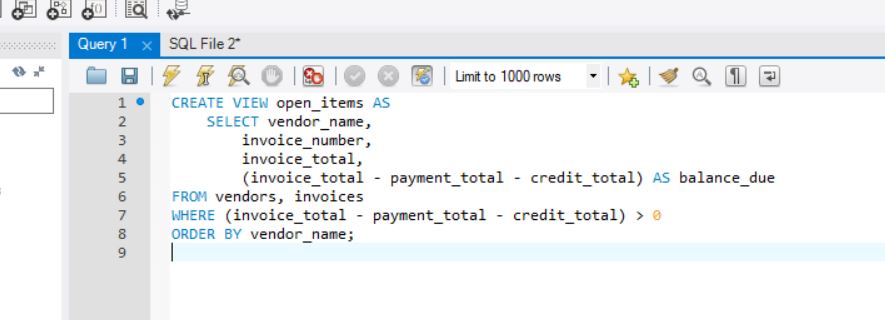
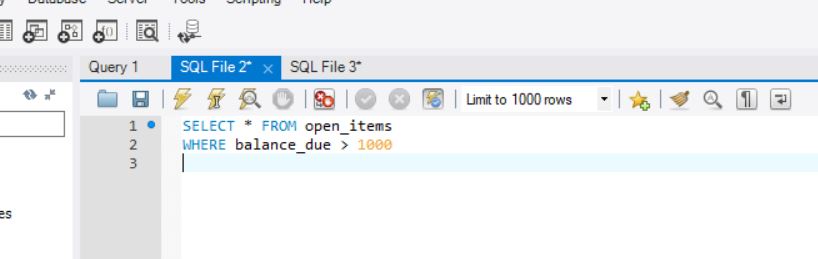
Chapter 12:

**Exercises**

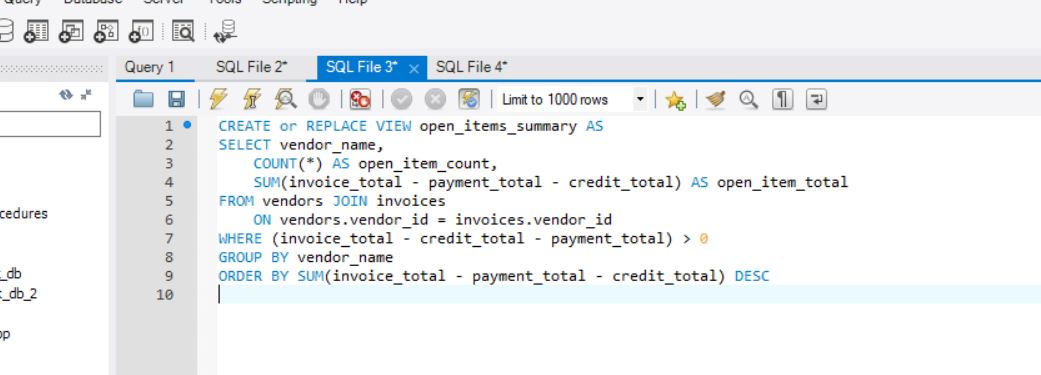
1. Create a view named **open\_items** that shows the invoices that haven’t been paid.
   1. This view should return four columns from the **Vendors** and **Invoices** tables:
      1. *vendor\_name, invoice\_number, invoice\_total,* and *balance\_due* (*invoice\_total - payment\_total - credit\_total*).
   2. A row should only be returned when the balance is greater than zero, and the rows should be in sequence by *vendor\_name*.



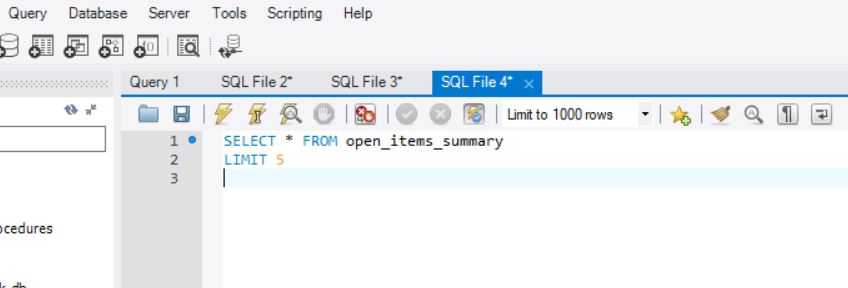
1. Write a **SELECT** statement that returns all of the columns in the **open\_items** view that you created in exercise 1, with one row for each invoice that has a balance due of $1000 or more.



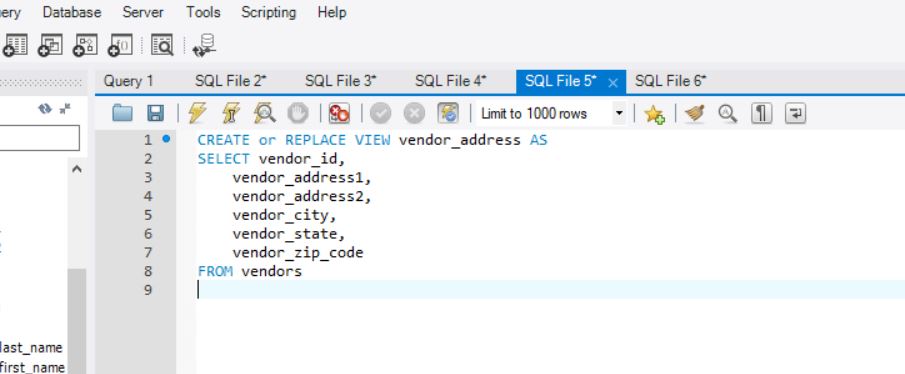
1. Create a view named **open\_items\_summary** that returns one summary row for each vendor that has invoices that haven’t been paid.
   1. Each row should include *vendor\_name, open\_item\_count* (the number of invoices with a balance due), and *open\_item\_total* (the total of the balance due amounts)
   2. The rows should be sorted by the open item totals in descending sequence.



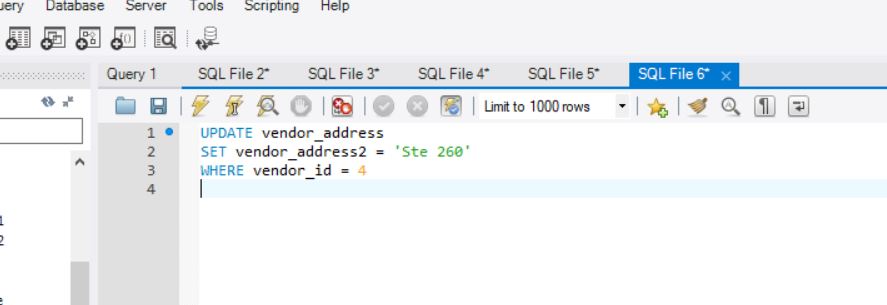
1. Write a **SELECT** statement that returns just the first 5 rows from the **open\_items\_summary** view that you created in exercise 3.



1. Create an updatable view named **vendor\_address** that returns the *vendor\_id* column and all of the address columns for each vendor.



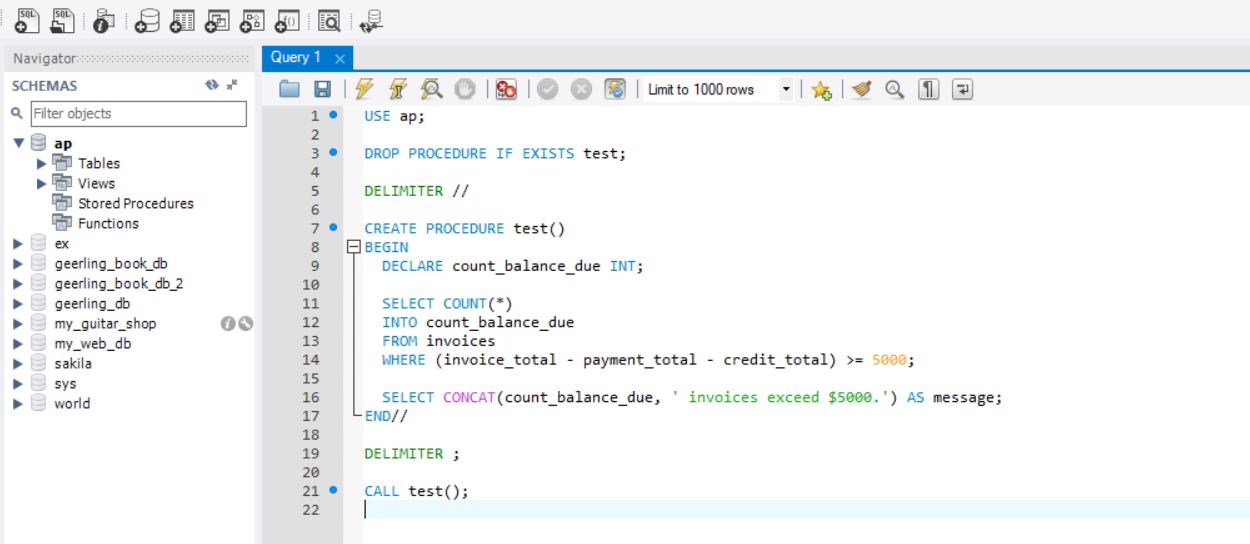
1. Write an **UPDATE** statement that changes the addresses for the row with a vendor ID of 4 so the suite number (Ste 260) is stored in the *vendor\_address2* column instead of the *vendor\_address1* column.



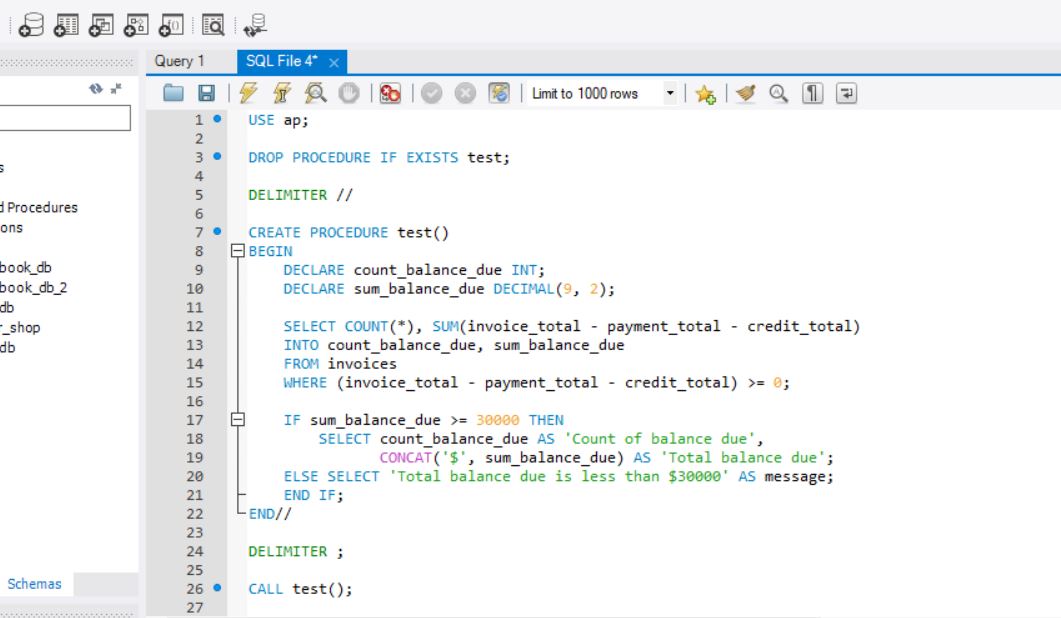
Chapter 13:

**Exercises**

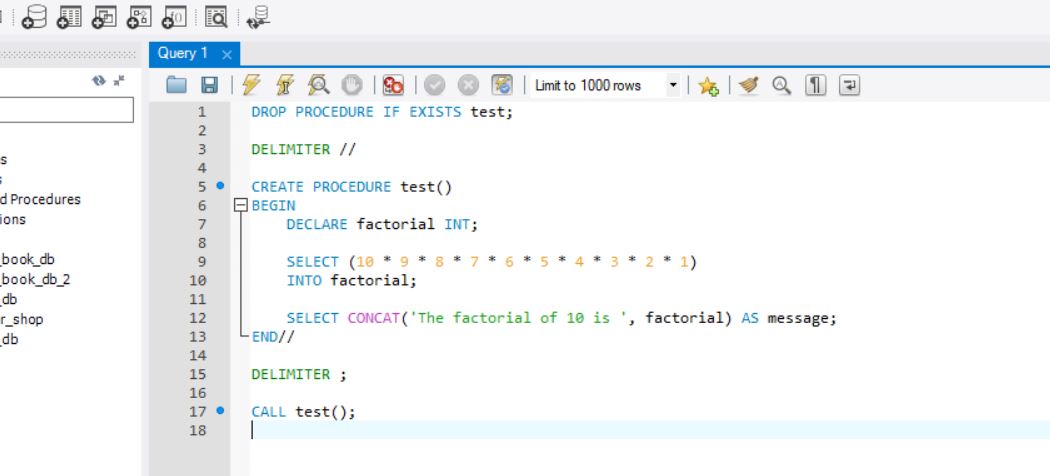
1. Write a script that creates and calls a stored procedure named **test**. This stored procedure should declare a variable and set it to the count of all rows in the **Invoices** table that have a balance due that’s greater than or equal to $5000. Then, the stored procedure should display a result set that displays the variable in a message like this:
   1. 3 invoices exceed $5,000.



1. Write a script that creates and calls a stored procedure named **test**. This stored procedure should use two variables to store (1) the count of all the invoices in the **Invoices** table that have a balance due and (2) the sum of the balances due for all of those invoices. If that total balance due is greater than or equal to $30,000, the stored procedure should display a result set that displays the values of both variables. Otherwise, the procedure should display a result set that displays a message like this:
   1. Total balance due is less than $30,000.

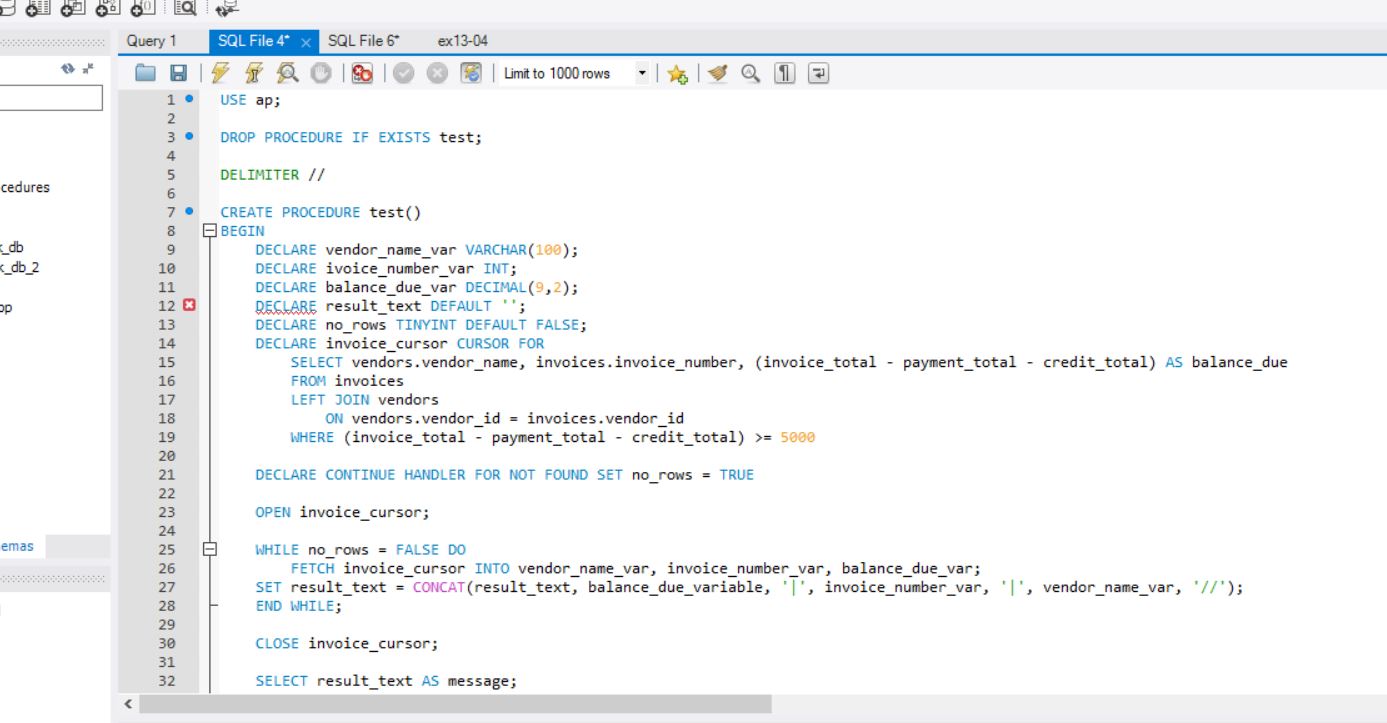


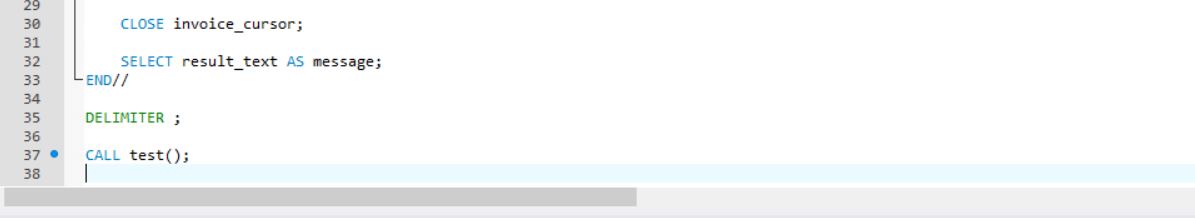
1. Write a script that creates and calls a stored procedure named **test**. This procedure should calculate the factorial for the number 10. (To calculate a factorial, you multiply an integer by every positive integer less than itself.) Then, it should display a string that includes the factorial like this:
   1. The factorial of 10 is: 3,628,800.



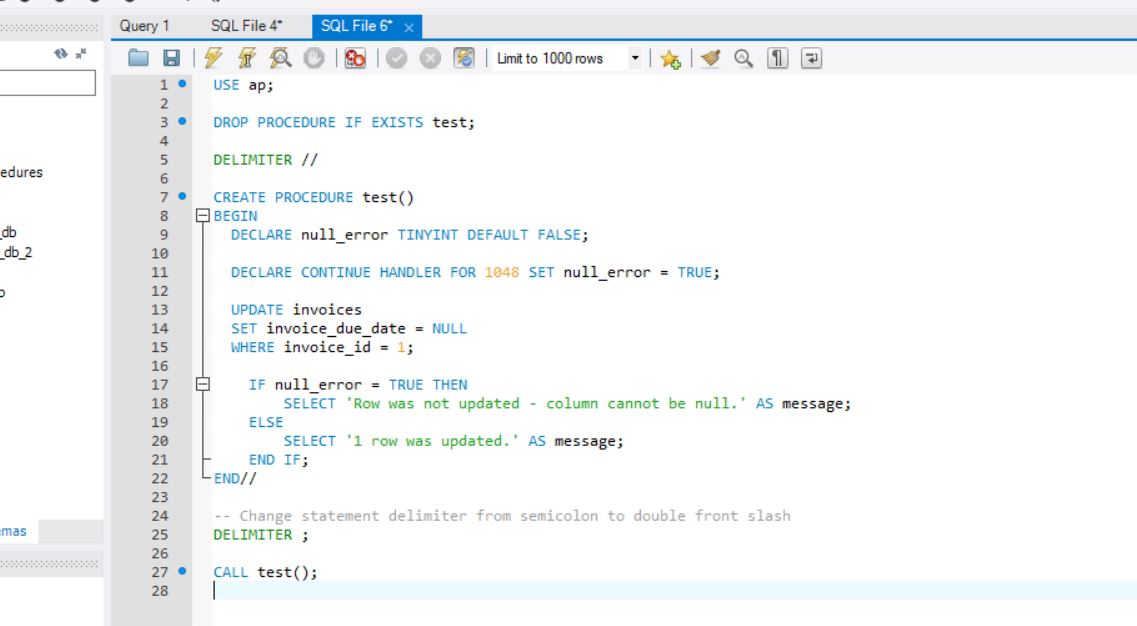
1. Write a script that creates and calls a stored procedure named **test**. This procedure should create a cursor for a result set that consists of the, *vendor\_name, invoice\_number,* and *balance\_due* columns for each invoice with a balance due that’s greater than or equal to $5,000. The rows in this result set should be sorted in descending sequence by balance due. Then, the procedure should display a string variable that includes the balance due, invoice number, and vendor name for each invoice so it looks something like this:
   * 1. 11130.70 | P-0608 | Malloy Lithographing Inc//6585.62 | 0-2436 | Malloy Lithographing Inc//
   1. Here, each column is separated by a pipe character (|) and each row is separated by two front slashes (//).

I’m not sure what I’m doing wrong on this one, I know I’m so close but I have to have an error somewhere.

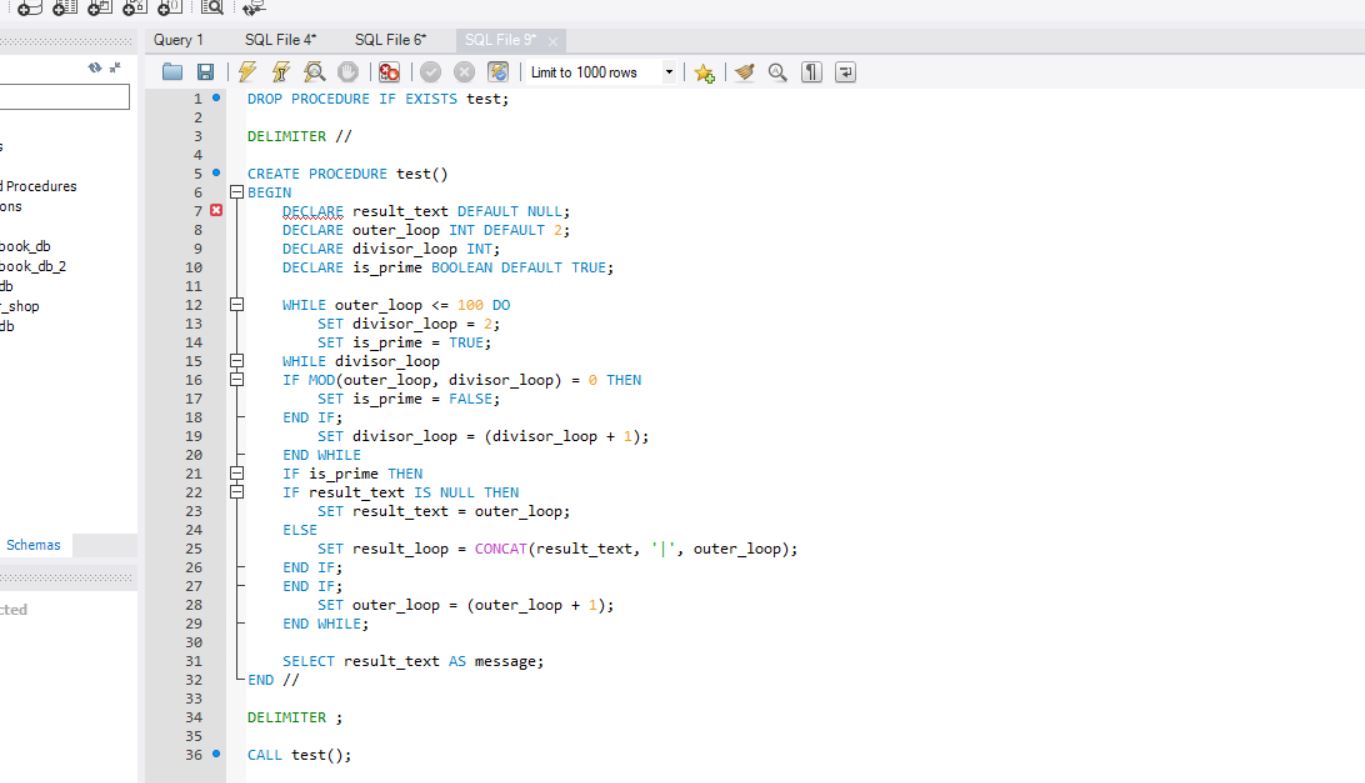




1. Write a script that creates and calls a stored procedure named test. This procedure should attempt to update the invoice\_due\_date column so it’s equal to NULL for the invoice with an invoice ID of 1. If the update is successful, the procedure should display this message:
   * 1. 1 row was updated.
   1. If the update is unsuccessful, the procedure should display this message:
      1. Row was not updated - column cannot be null.



1. Write a script that creates and calls a stored procedure named test. This procedure should identify all of the prime numbers less than 100. (A prime number is an integer that can’t be divided by another integer other than 1 and itself.) Then it should display a string variable that includes the prime numbers like this:
   * 1. 2 | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | …
   1. *Hint: to get this to work, you will need to nest one loop within another loop. In addition, you will need to code an IF statement within the inner loop.*



1. Enhance your script for exercise 4 so it shows the invoice data in three groups based on the balance due amount with these headings:
   * 1. $20,000 or More
     2. $10,000 to $20,000
     3. $5,000 to $10,000
   1. When you’re done, the string variable that’s returned should be in this format:
      1. $20,000 or More: $10,000 to $20,000: 11130.70 | P-0608 | Malloy Lithographing Inc//$5,000 to $10,000: 6585.62 | 0-2436 | Malloy Lithographing Inc//
   2. To accomplish this, you can loop through the cursor three times by opening and closing the cursor for each loop. *Hint: For each group of invoices, you can code a separate block of code that contains an EXIT handler for the NOT FOUND condition.*

