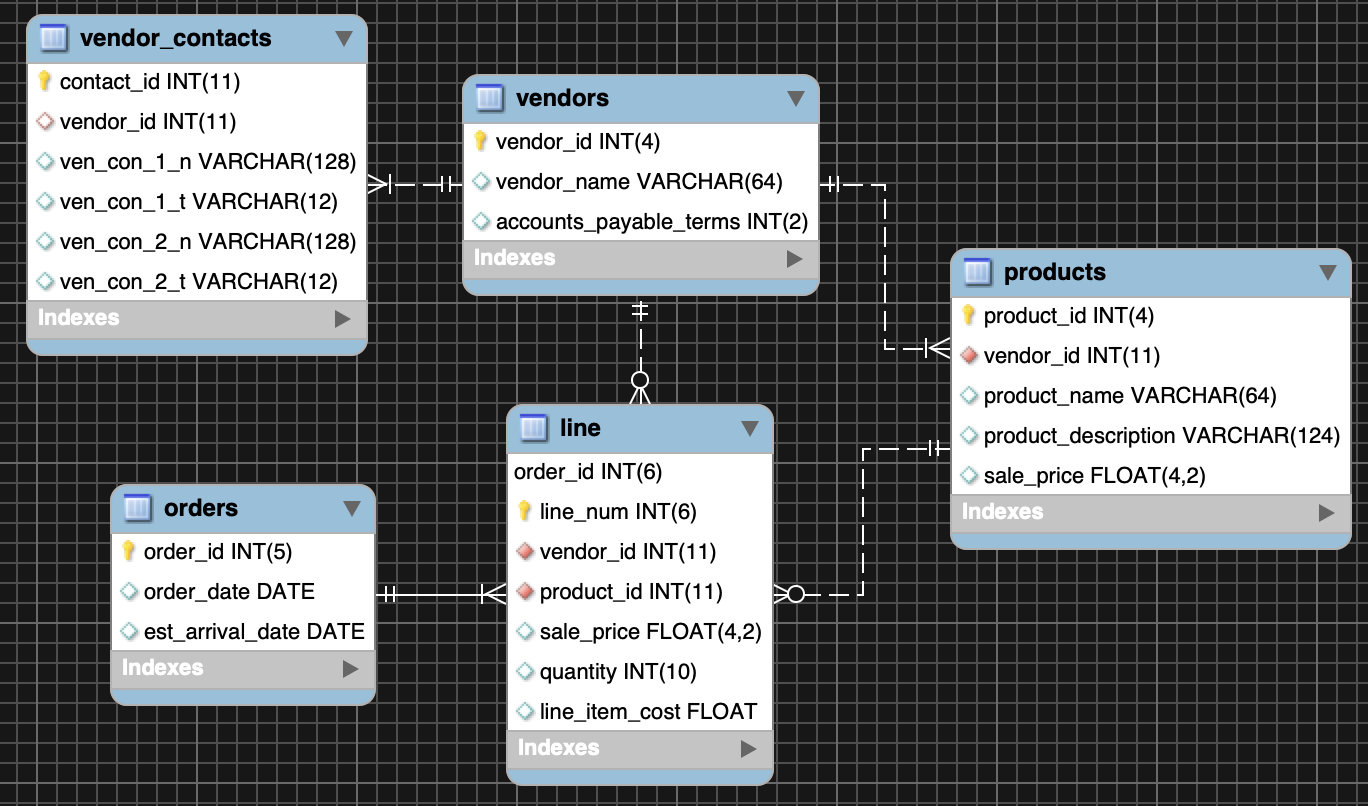
Name: \_\_\_\_Adam Capdeville\_\_\_\_\_\_\_\_ Lab 5

Database Systems Oct. 2, 2019

Background: AirSupplyData-2019.xlsx contains transactional data from supply orders placed with multiple vendors. While items may share product names, they do not share item numbers or descriptions. A/P refers to the accounts / payable for each vendor, or the number of days before a payment is due. The default arrival date should be set to NULL for new orders. Cost per order can be calculated by multiplying sale price and quantity. Price should be captured is both product and line tables since it is both transactional (invoice-level data) and historical data (product-level data).

**Part I. (20pts)**

1. Create a normalized database to 3NF for the transactional data found in AirSupplyData-F19.xlsx. Reverse engineer your database in MySQL Workbench and paste the result below.

****

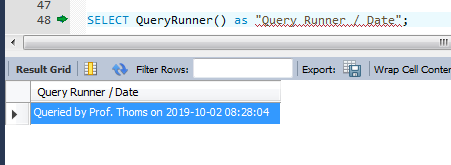
**Part II. (5pts)**

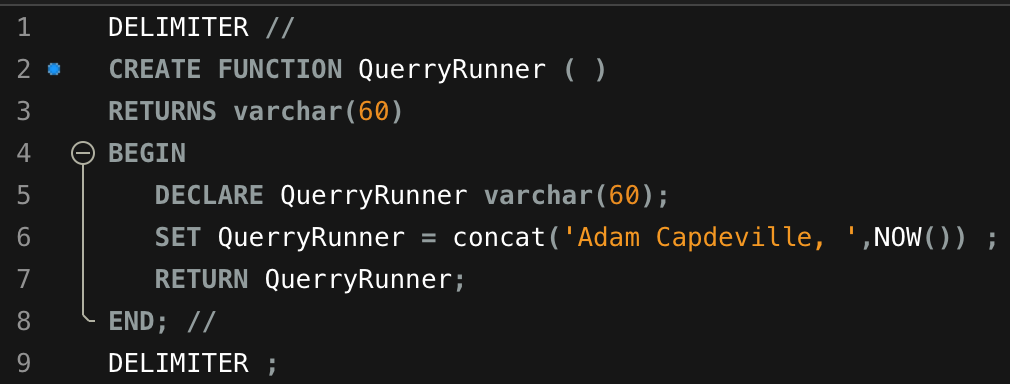
1. Insert the data found in AirSupplyData-2019.xlsx.
   1. Note: Data cleansing may be required. Do not include data for derived attributes.
   2. Detail the steps you took to import your data.

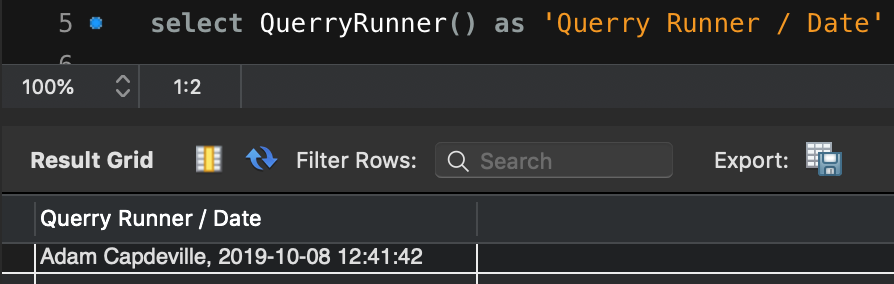
**vendors**: cleared out repeated vendors.  
**products**: cleared out repeating products unless made by a different vendor.  
**orders**: removed redundant data (repeating orders\_id) and separated the invoice of orders into a line table.  
**vendor\_contacts**: cleared out repeating vendors.  
**line**: the line table is composed of the order invoice line\_num and the product with purchase price.

**Part III. (5pts)**

1. Create a user-defined function to return a valid query runner and date. This function will be used by all new queries, views and stored procedures going forwards. For creating user-defined functions refer to the following online resource: <https://dev.mysql.com/doc/refman/8.0/en/create-function-udf.html> and the updated Week 6 slides.





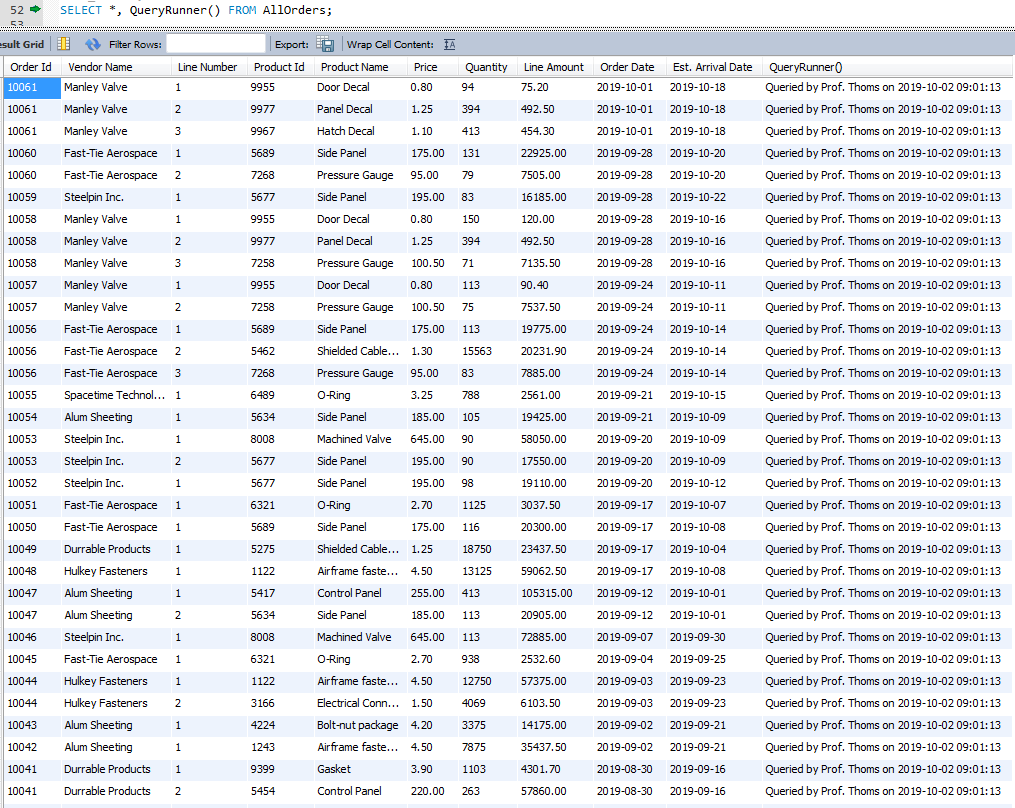


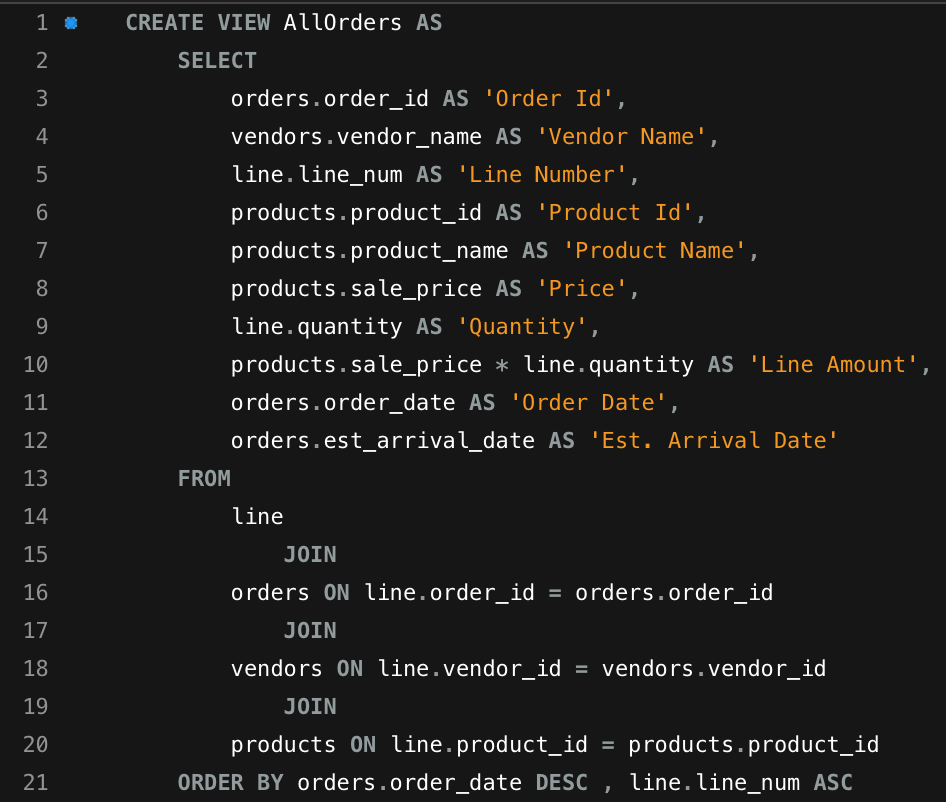
**Part IV. (20pts)**

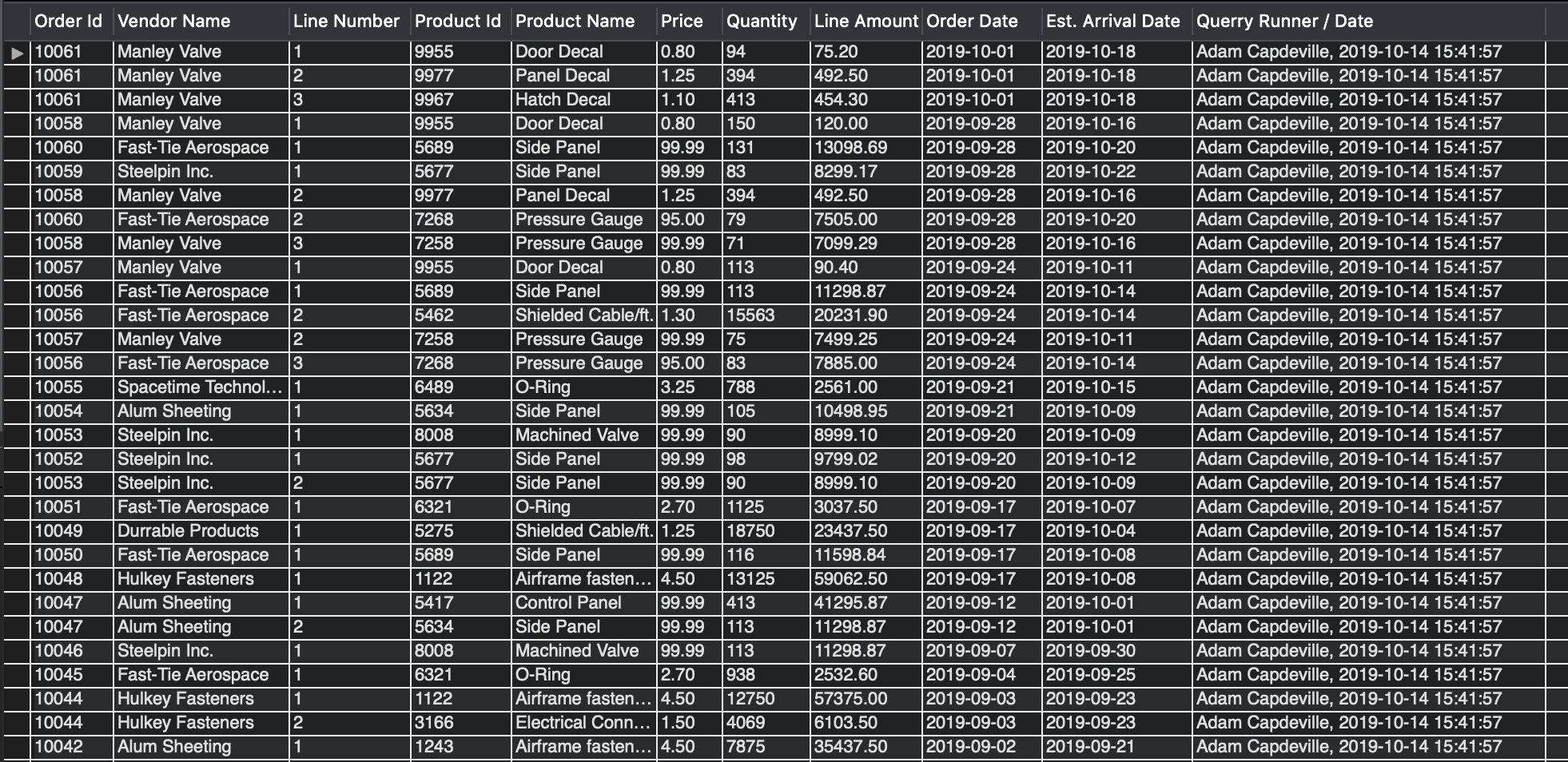
For all views and stored procedures, return QueryRunner() in your output to show your name and date as shown in the screenshot below.

*VIEWS*

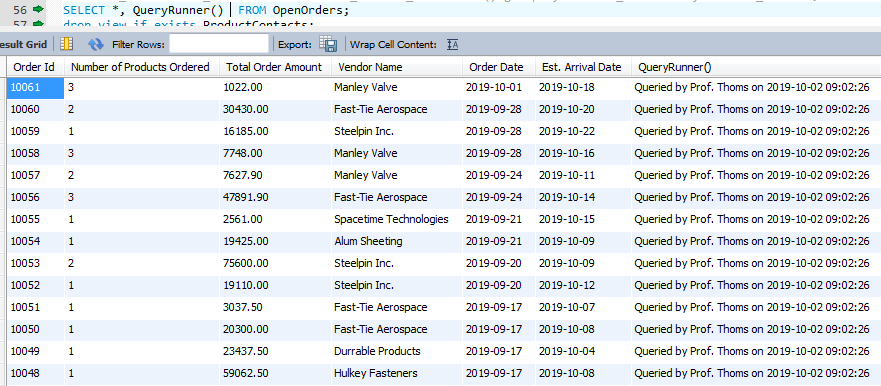
1. Create the SQL view `AllOrders ` that returns all order information as shown below. Order by the order date descending, line number ascending.

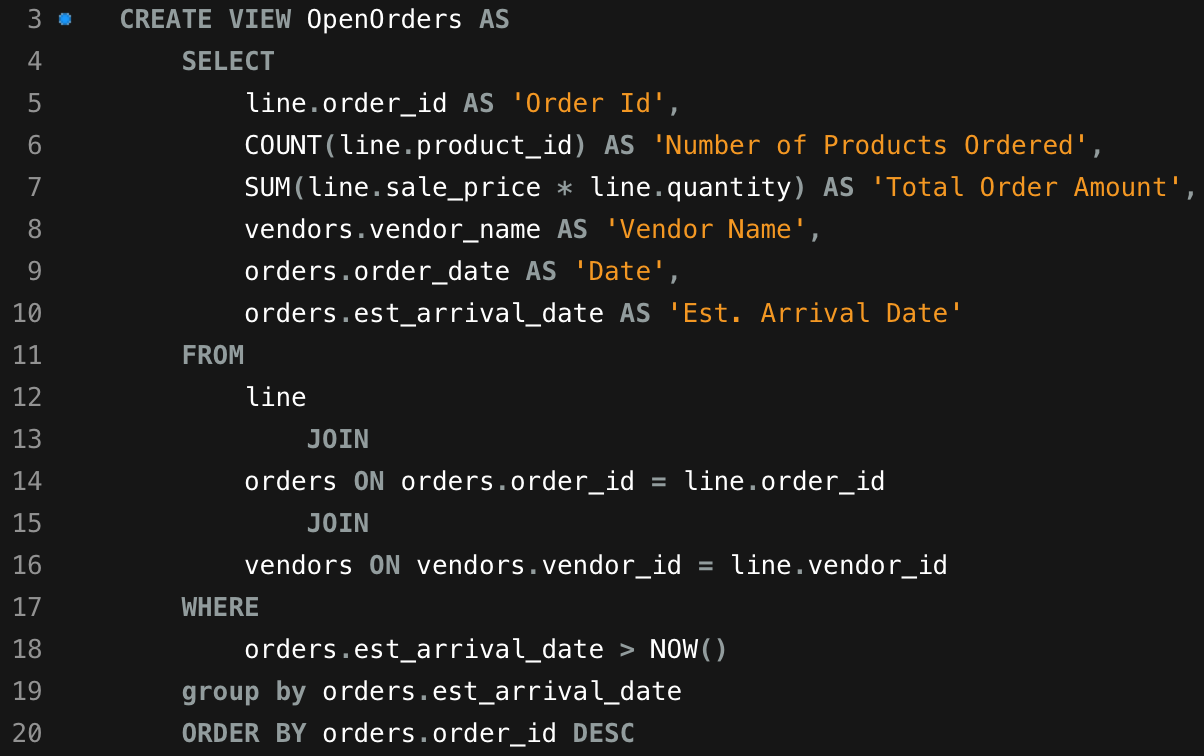


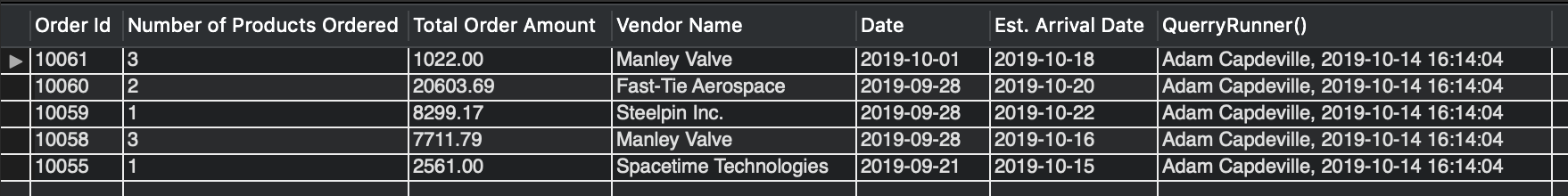


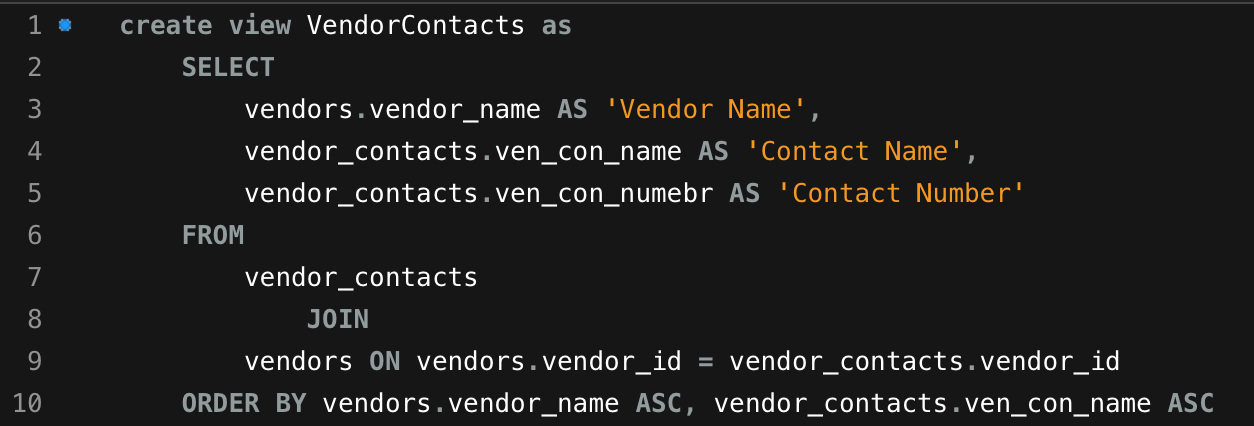
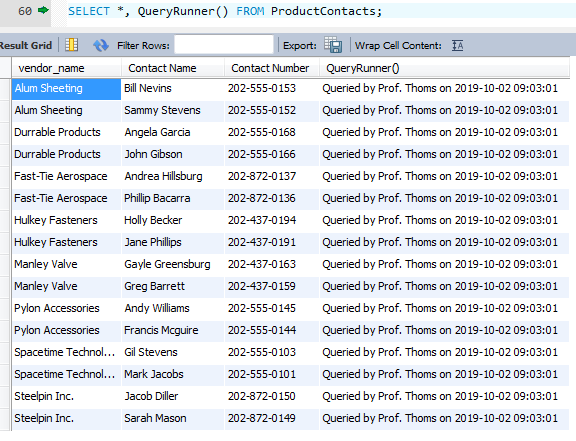
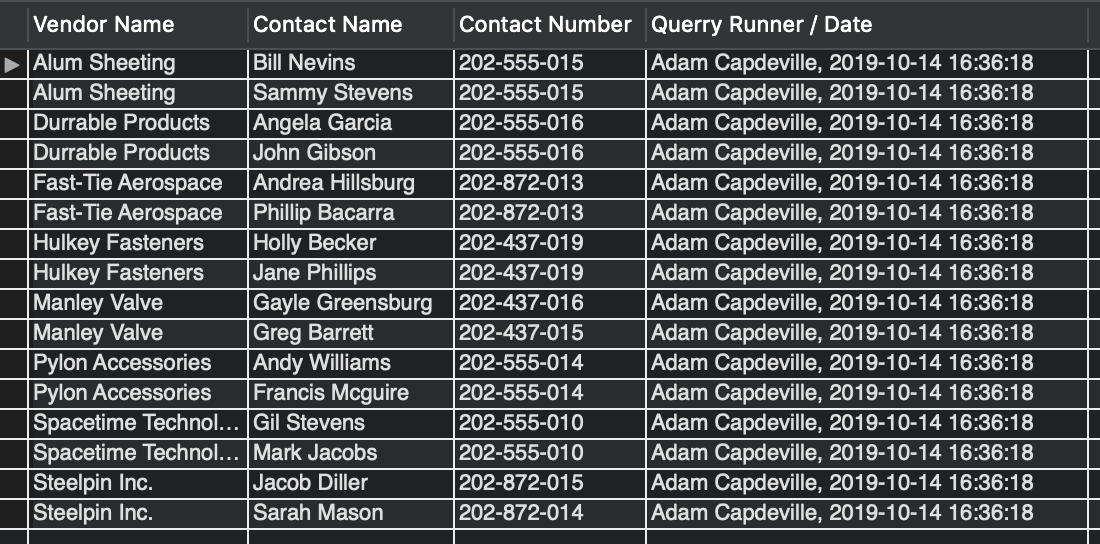


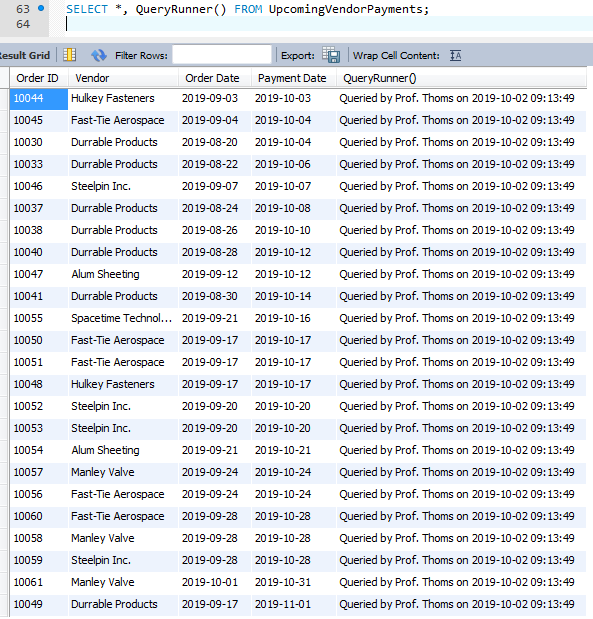
1. Create the SQL view `OpenOrders` that returns order information for orders that have not yet arrived (use estimated delivery date). Return invoice-level data including number of products ordered and the total dollar amount for the order as detailed below.

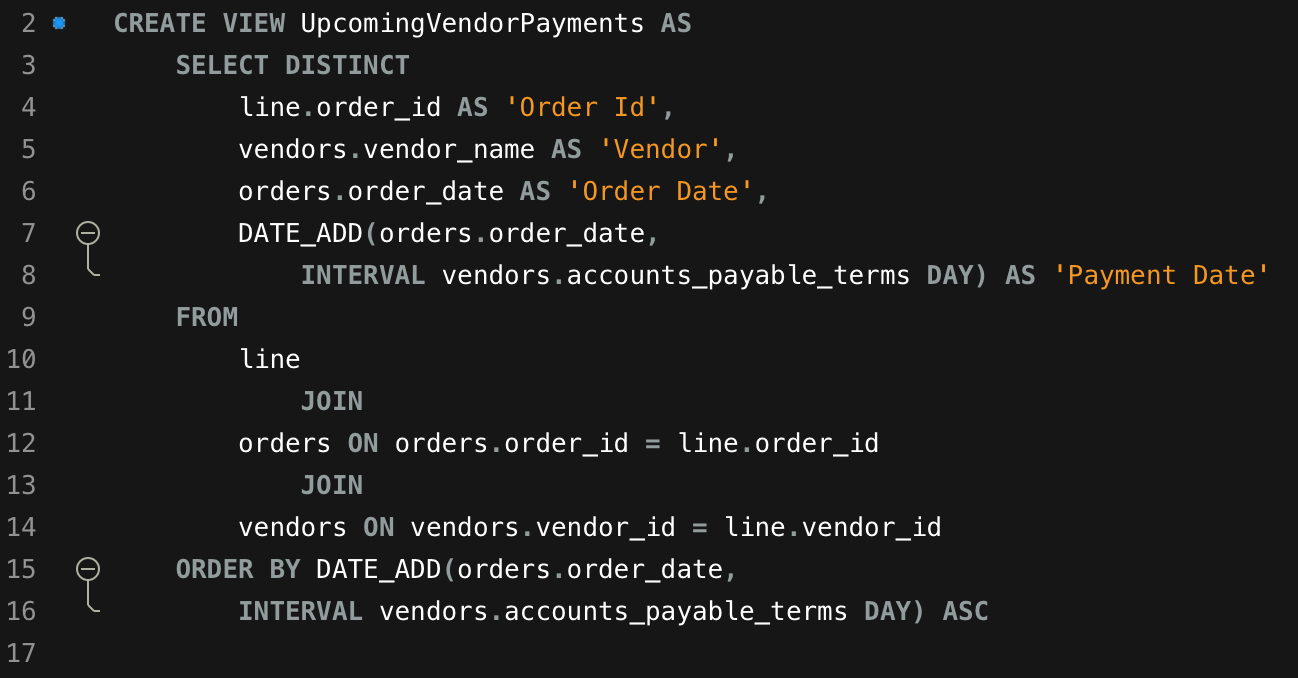


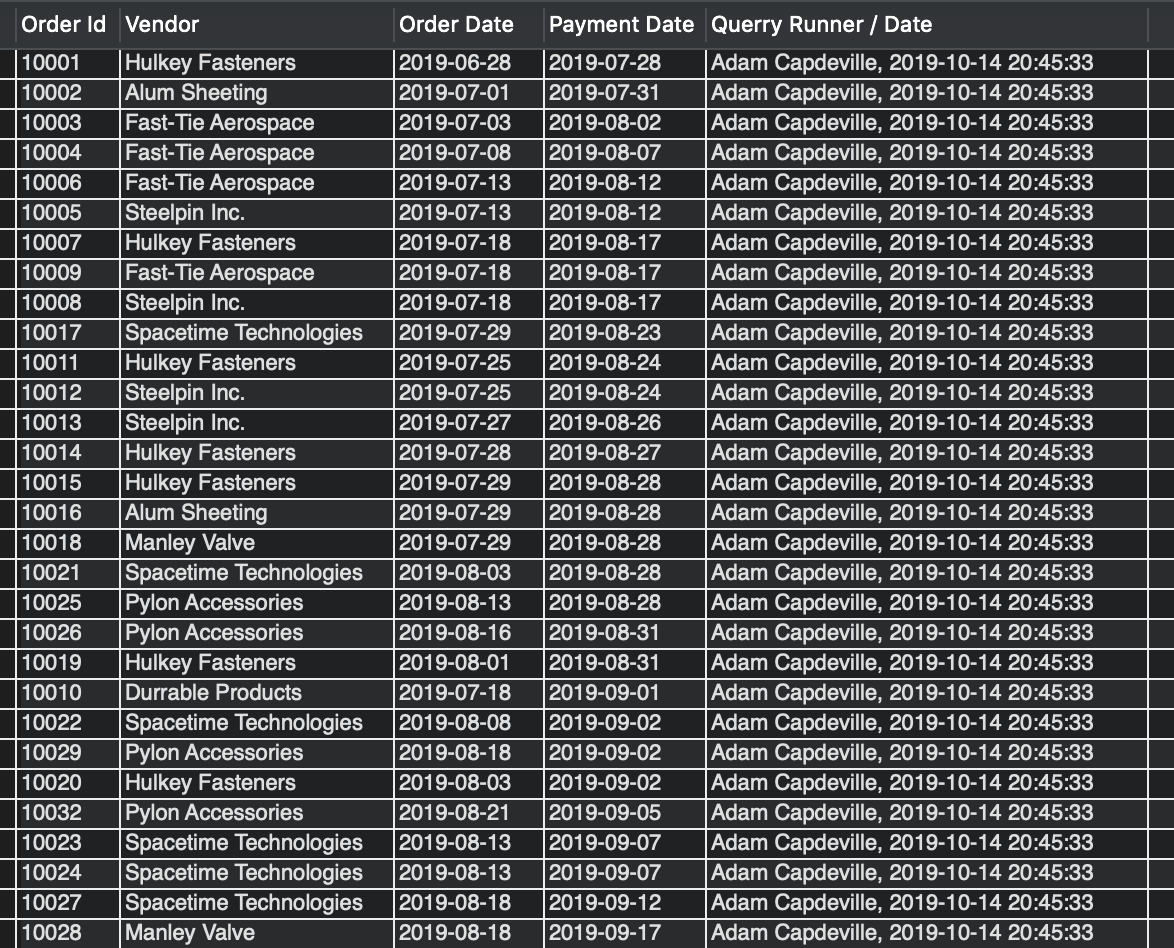


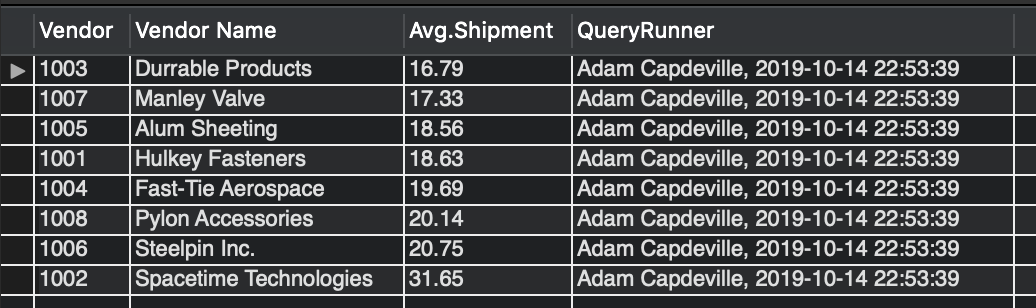
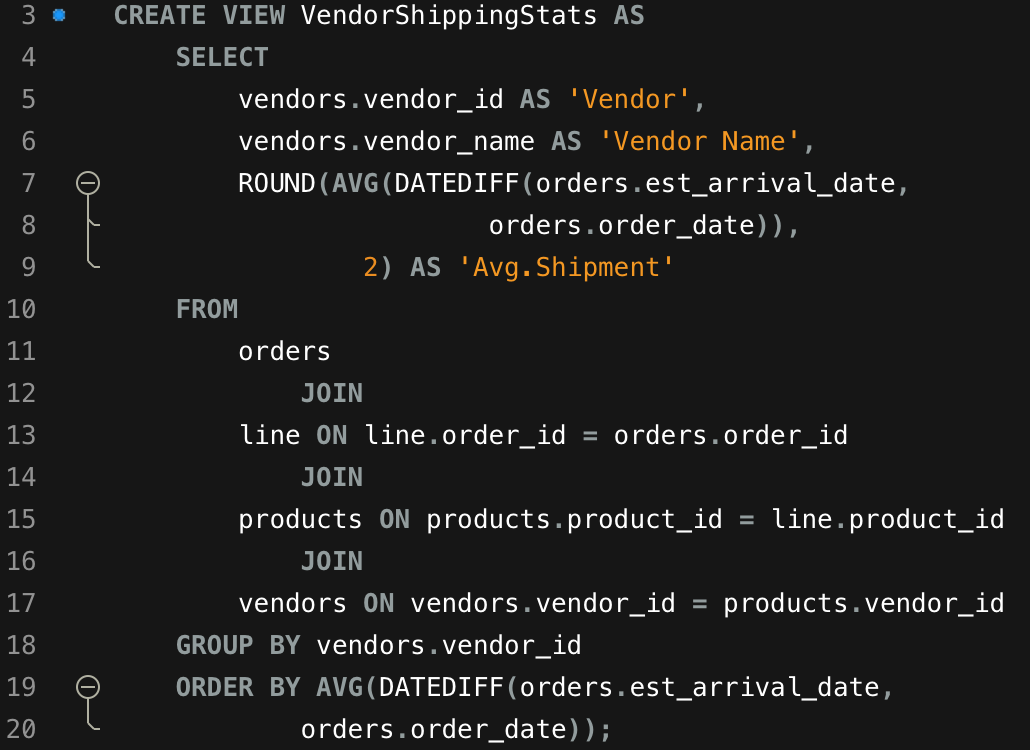
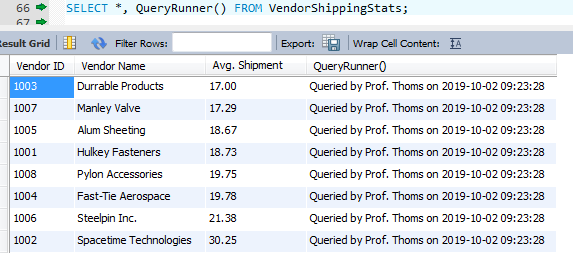


1. Create the SQL view `VendorContacts` that returns a listing of contacts for each vendor. Sort your results alphabetically by vendor, then contact last name, first name. Include fields shown below.  
     
    
2. Create the SQL view `UpcomingVendorPayments` that outputs the future dates for when payments are due for orders where payment is not yet due. Use the accounts / payable terms to determine the next date for payment. Include fields shown below.



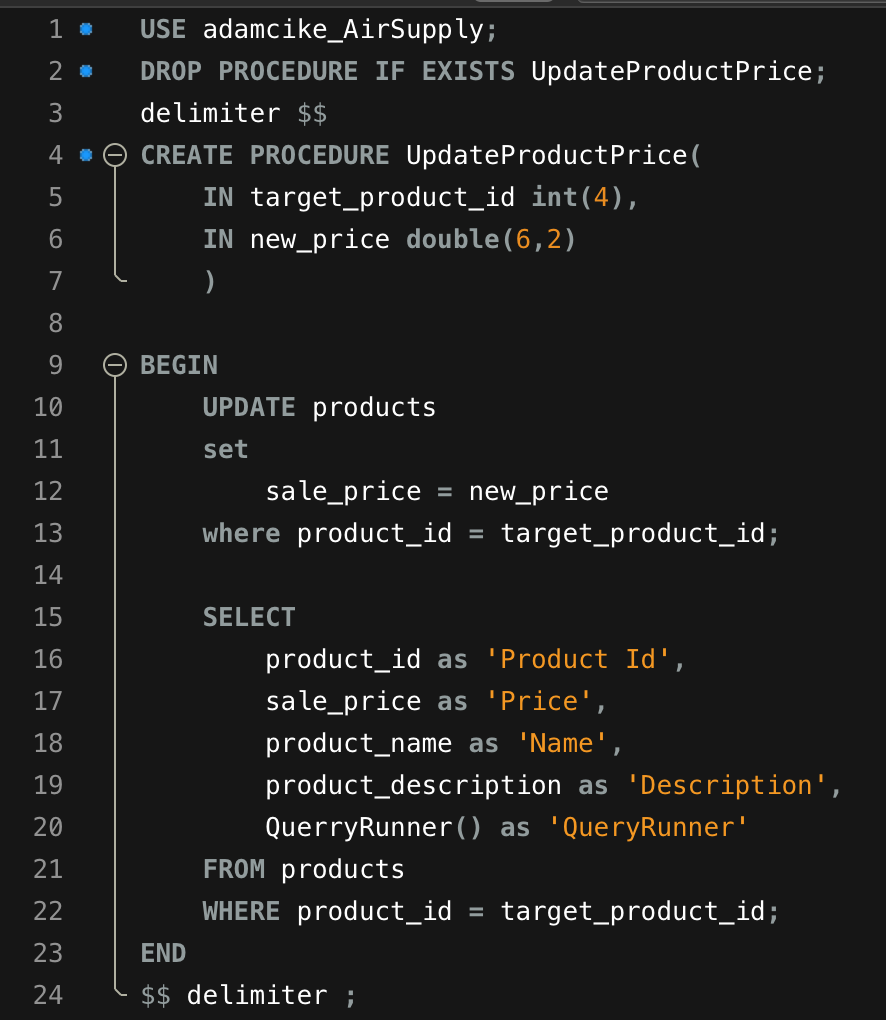


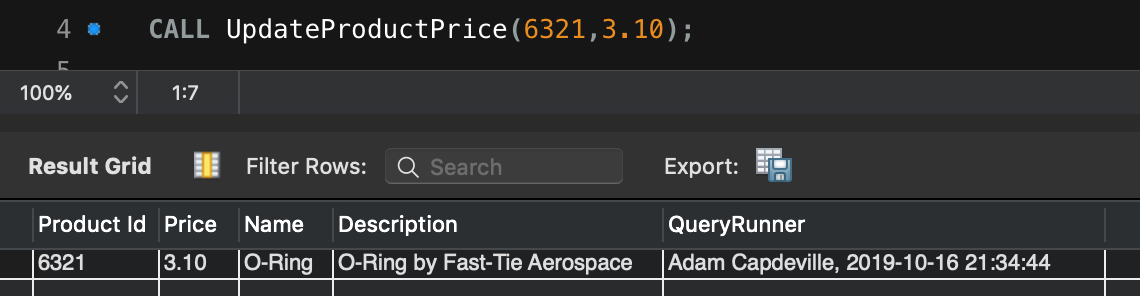


1. Create the SQL view `VendorShippingStats` that produces a listing of the average ship times for each vendor. Sort the results by shortest to longest duration.  
   

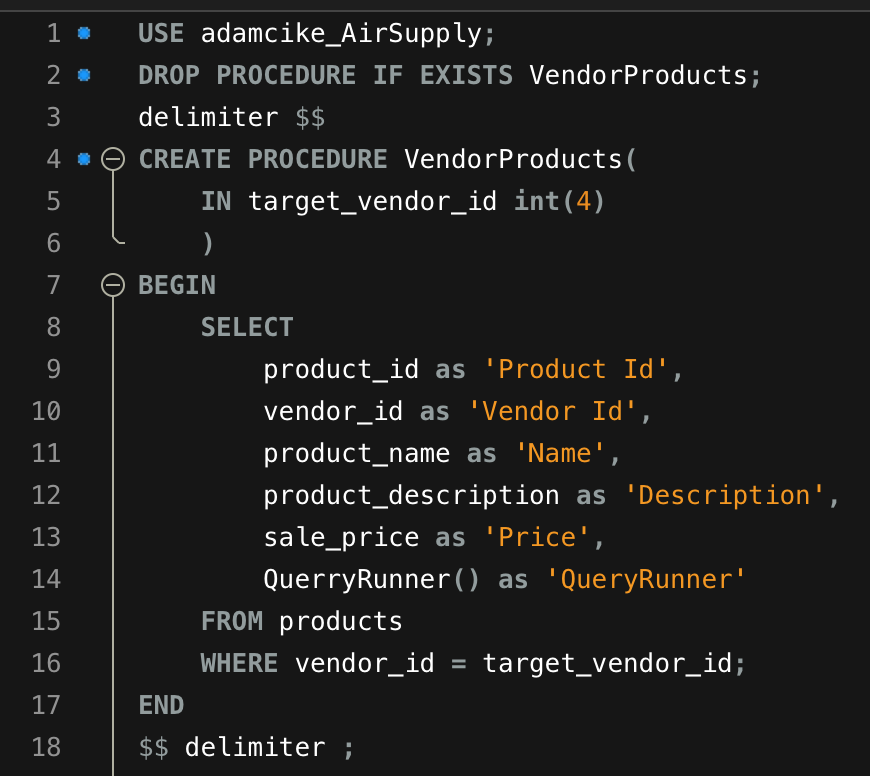
*STORED PROCEDURES (output coming soon!)*

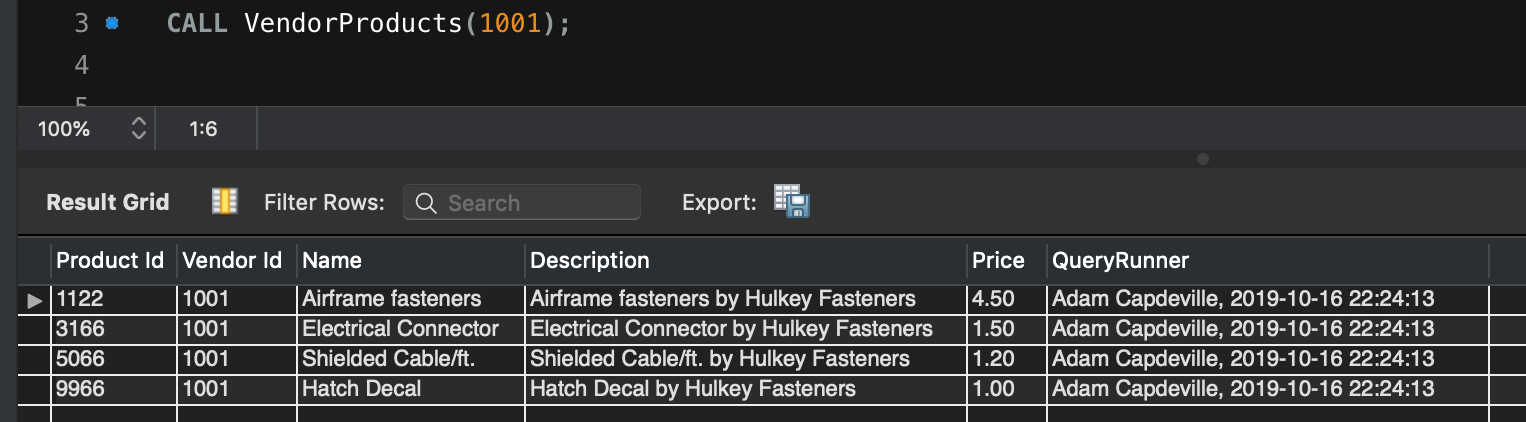
1. Create the SQL stored procedure `InsertVendorContact` that allows a user to insert new contact information for a vendor. After inserting the new data return a listing of contacts for this vendor. Sort your results alphabetically by vendor, then contact last name, first name.

1. Create the SQL stored procedure `UpdateProductPrice` that accepts vendor, product and new price data and updates a vendor’s product price accordingly. Return the new product and pricing information for the newly inserted product.  
   



1. Create the SQL stored procedure `VendorProducts` that accepts a vendor id as input and returns a listing of available vendor products. Sort the data by product name, price.





1. Create the SQL stored procedure `SearchProductByDescription` that accepts a search string and returns products matching any part of the search string. Return all relevant product information for that product, including vendor, product id, description and price. Sort the results by product name, price ascending.
2. Create the SQL stored procedure `SearchOrdersByDate` that accepts a start date and end date as parameters and returns all order, product and vendor information for orders placed on or between those dates sorted by order date. Capture all relevant information from order, item and vendor tables.
3. (Extra Credit) Assuming the current 3-month interest rate for a U.S. treasury 2.43% (as of Feb. 26, 2019), construct a view that calculates the total dollar amount the company saves by not paying until the last possible date according to the A/P terms of the vendor versus paying on the invoice date. Capture all relevant information from order, item and vendor tables.