An Introduction to Databases for Accountants

# Author Information

Vic Anand, Ph.D.

Assistant Professor of Accountancy – University of Illinois at Urbana-Champaign

R.C. Evans Data Analytics Fellow - University of Illinois-Deloitte Foundation Center for Business Analytics[[1]](#footnote-1)

# Learning Objectives

* Learn how transaction-level accounting data should be stored.
* Gain experience with the design of relational databases.
* Write basic SQL queries to create tables, retrieve data from one table, merge data from multiple tables, aggregate and filter data from one or more tables.
* Integrate database skills with Python skills by using the SQLite and Pandas packages.
* Take the perspective of a preparer of financial statements. Retrieve, aggregate, and report financial statement line items that are created from transaction-level data.

# Introduction

The purpose of this case is to introduce accounting students to databases. Practicing accountants interact with data from databases whether they realize it or not. For example, transaction-level data captured in accounting systems is stored in databases. Preparers of financial statements should therefore understand how accounting data is structured and stored in databases.

In this case, you will assume the role of a preparer of financial statements. You will create a database to store transaction-level data for a fictitious company and then compute some common accounting metrics like revenue and cost of goods sold. Within the database, you will create separate tables to store customer information and customers’ orders. You will also create a table to store information about the company’s products. Finally, you will retrieve data from these tables, merge that data, and transform it into useful numbers like revenue or cost of goods sold.

## Company Description

In this case, we borrow data from the famous Northwind Traders database, a sample database that Microsoft provides with its Microsoft Access database software. Northwind Traders (NWT) is a fictitious company that sells gourmet food products. The original Microsoft-supplied database contains tables for NWT’s customers, employees, suppliers, inventory, customer orders, supplier orders, and more. However, in this case, you will only work with a subset of that data.

## Introduction to Databases

A database is a collection of data, typically organized into tables. Each table consists of rows (called records) and columns (called fields), and is similar to an Excel worksheet. Figure 1 shows a subset of the table that stores information about NWT’s products. Notice that the table has one row per product. Also notice that the columns contain different information about each product, such as its name and price.

While a database table may appear similar to a spreadsheet, there are some key differences. Unlike a spreadsheet, each table column must contain the same type of data. For example, the Product Name column contains text data and the List Price column contains numeric data. A database strictly enforces column data types. The reason for this constraint is that it often does not make sense to store different data types in the same column. For example, it would not make sense to store text data in price or cost columns.[[2]](#footnote-2)

Another key difference between a database table and a spreadsheet is the primary key. A primary key is a field (or fields) that uniquely identifies each record. In products table shown in Figure 1, the Product Code field is the primary key. This code is different for every product and can therefore uniquely identify each product.



Figure 1: Northwind Traders' products table

You might wonder whether a product’s name can serve as a primary key. It could if each product has a unique name. However, what if the company offers the same product in different sizes? We would need to add a new record to the table for the other sizes, but the product name would be the same for all sizes. Therefore, since the product name might not always be unique, it is not a good candidate for a primary key.[[3]](#footnote-3)

### Common Brands of Databases

Some common brands of databases that you might encounter are Oracle, MySQL, Microsoft SQL Server, Microsoft Access, and PostgreSQL. In this case, we will work with SQLite, a lightweight, simple database. Everything you learn here is transferable to other common types of databases.

### Client-Server vs. Embedded Databases

Many databases, such as those used in websites, utilize a client-server model. For example, when a Capital One credit cardholder makes a purchase, the details of their card transaction are stored on a database that is hosted by Amazon Web Services. Thus, the database that Capital One uses is “remote” and their transaction server, the client, needs to communicate with a remote database server.

Embedded databases are integrated within their apps. For example, if you play a computer game on your gaming console, game data might be stored in an embedded database that resides on your console. Phone apps like Dropbox also use embedded databases to keep track of local files.

In this case, we will work with an embedded database, SQLite. Because it is embedded, it is stored in a single file on your computer, and you can easily view and interact with it.

## Prerequisites

* You should have access to Jupyter notebooks (or JupyterLab) running Python 3.
  + If you do not have access to Jupyter notebooks, I recommend you use [Microsoft Azure Notebooks](https://notebooks.azure.com/). This is a free service from Microsoft that you can run in a browser that requires no additional software on your computer.
  + If you wish to work on your own computer, I suggest you download and install the [Anaconda distribution of Python](https://www.anaconda.com/). This is a premiere distributions of Python that is designed for data science. It includes a Jupyter notebook server and JupyterLab.
* You will need access to the following Python packages, both of which are included with Microsoft Azure Notebooks and with Anaconda.
  + pandas
  + sqlite3
* Basic knowledge of SQL. If you need a tutorial or a refresher, I recommend either of:
  + <https://www.w3schools.com/sql/>
  + <https://www.sololearn.com/Course/SQL/>
* The program [DB Browser for SQLite](http://sqlitebrowser.org/).
  + This free software allows you to create and manage SQLite databases.
  + It is available for Windows and Mac.
  + While it is possible to do this entire case using only Python, DB Browser will simplify many of the tasks and will allow you to easily visualize the database tables.

# Creating a Database with Tables for Customers and Products

This section provides instructions and guidance for the first three deliverables. In sum, you will first create a blank database with an empty products table. Then you will populate that table with data from a file that accompanies the case. Finally, you will create and populate a customers table with data from a file that accompanies the case.

## Create a Blank Database

You will now create a blank database. While it is possible to do this entirely from Python, we will use a tool, DB Browser for SQLite, to simplify your task.

1. Download and install [DB Browser for SQLite](http://sqlitebrowser.org/).
2. Open DB Browser for SQLite. You will see a window similar to that shown in Figure 2.
3. Click on the New Database button, circled in red in Figure 2.
4. Navigate to the folder in which you wish to store your database and save your database file as NWT.db.

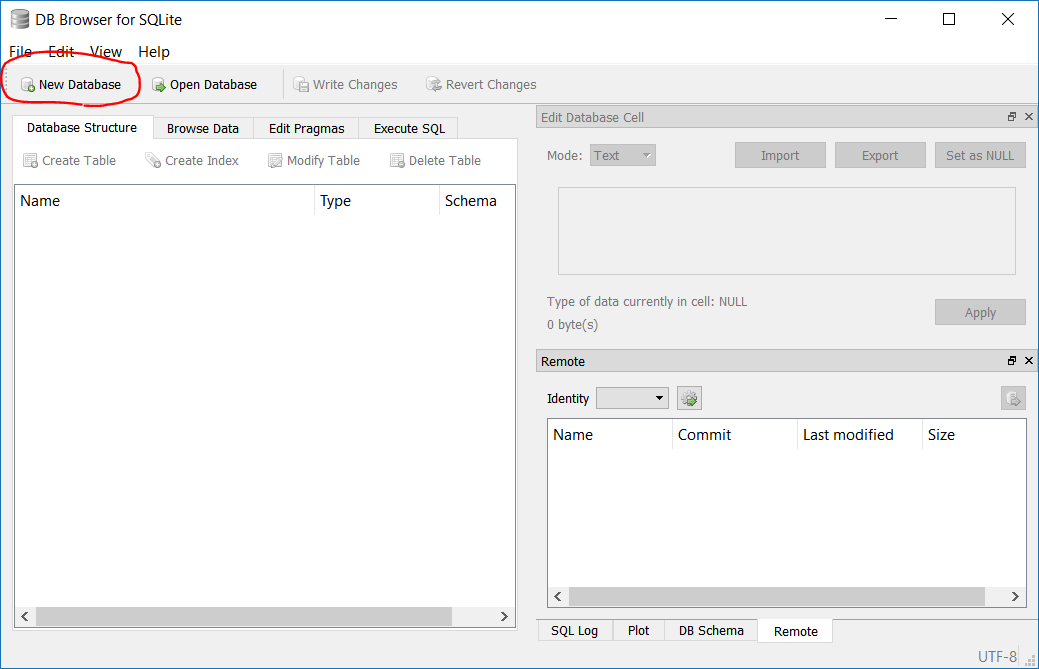


Figure 2: DB Browser for SQLite with no database loaded

## Create a New Table Named *tblProducts*

A new dialog will appear that will allow you to create a table (see Figure 3). If it does not, click the *Create Table* button. Name your new table tblProducts, then click the Add field button. Follow the steps below to create the products table.

1. Replace the text Field1 with ProductCode. Change the type to TEXT, and select the PK checkbox (this tells SQLite that ProductCode will be your primary key column).
2. Continue to click the *Add field* button and populate the dialog with all the fields shown in the right snapshot of Figure 3.
   1. Make sure the *PK* checkbox is checked for the ProductCode field (and only for that field!).
   2. I strongly recommend that you do not use spaces in the field names. While it is allowed, your life will be much easier if you do not use spaces.
3. Note that, as you add fields, the bottom panel of the dialog updates to show the SQL code that is generated. Ultimately, DB Browser will execute SQL code to create a new table. The graphical user interface automates the process of writing SQL.
4. When you have populated the dialog with all of the fields shown in Figure 3, click the OK button in the bottom right.
5. Verify that the table structure was correctly saved by comparing your DB Browser window to that shown in Figure 4.
6. In the main window for DB Browser (Figure 4), click the *Write Changes* button. This will save the new table to the database file.

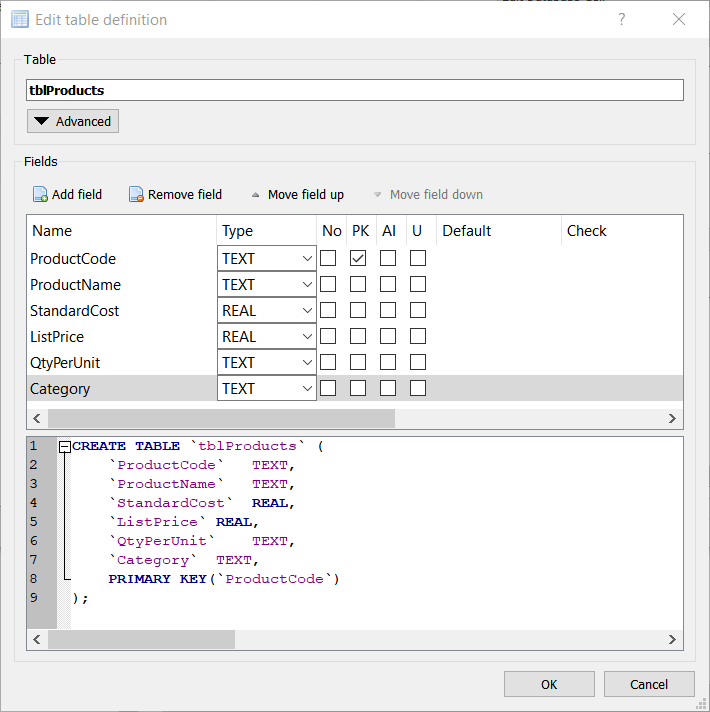
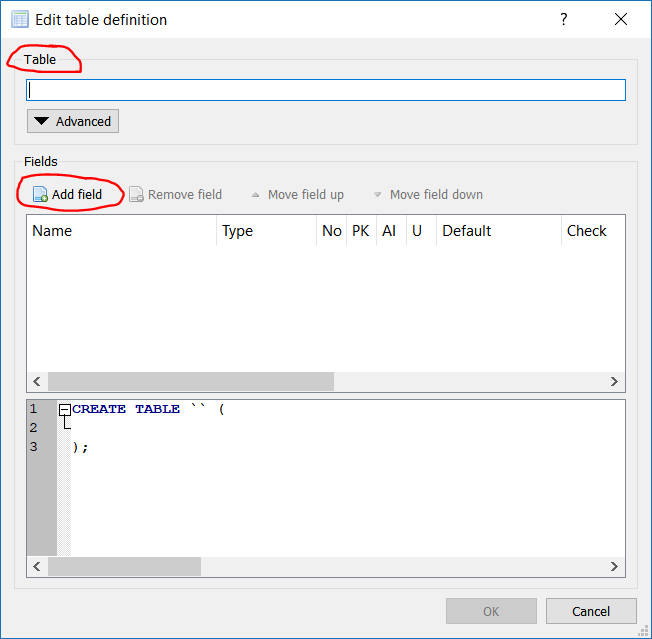


Figure 3: Dialog for creating a new database table. The left dialog appears after you first create a new database, or when you click the Create Table button. The right dialog shows the dialog populated with the fields needed for the Products table. Note that the bottom panel of the right dialog shows the SQL code to create the table.

You have completed the first deliverable, which is summarized below.

**DELIVERABLE 1: Create database file *NWT.db*. Within that file, create an empty table named *tblProducts* with the correct table structure.**

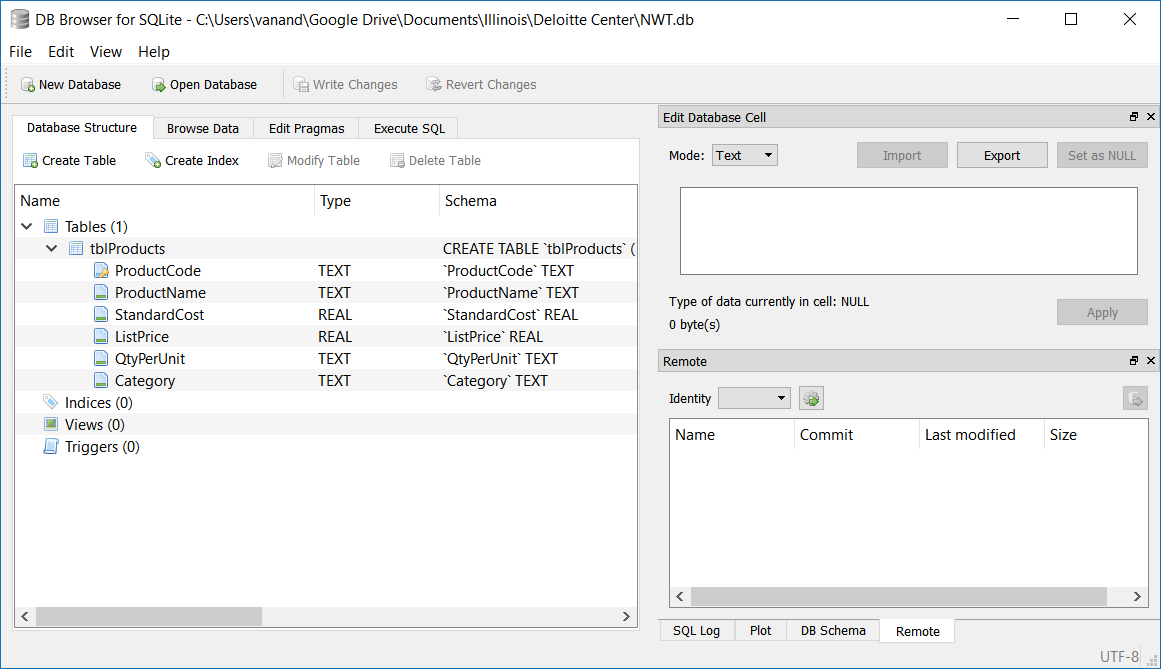


Figure 4: DB Browser window showing correctly configured products table (tblProducts)

## Adding Data to the Products Table

You have now created a database file and, within it, a table that can store information about the company’s products. However, the table is empty. It has a structure, but no records. Thus, your next task is to populate the table with data from the file Products.csv that accompanies the case. There are multiple ways to accomplish this:

1. Using the Browse tab in DB browser, click the New Record button and manually type data from the CSV file. You would need to do this for each of the 45 rows in Products.csv. I strongly discourage this approach, as it has the following consequences:
   1. It is error-prone.
   2. It is time-consuming.
   3. You will be subject to ridicule from me, your professor, and your peers.
2. Using the Execute SQL tab of DB Browser, write an SQL [INSERT INTO](https://www.w3schools.com/sql/sql_insert.asp) statement that inserts multiple rows. You could open the *Products.csv* file in Excel and use some Excel formulas to create this statement.
3. (Recommended approach) Use the Pandas and sqlite3 packages in Python. Students who are using a remote Jupyter server will need to upload their *NWT.db* and *Products.csv* files to that server. The following tips should help:
   1. Make a backup copy of your NWT.db file.
   2. Consider reading [this tutorial on working with SQLite databases using Pandas](https://www.dataquest.io/blog/python-pandas-databases/).
   3. Open a connection to the database in Python. Do so using the connect function of the sqlite3 library, e.g. conn = sqlite3.connect('NWT.db').
   4. Load the data from *Products.csv* into a Pandas DataFrame.
   5. Transfer the data from your Pandas DataFrame into a table by using the [*to\_sql*](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_sql.html) method, e.g. df.to\_sql(…).
      1. You will need to specify the arguments to the *to\_sql* function.
      2. I strongly recommend you use the *name*, *con*, *if\_exists*, and *index* arguments.
   6. Close the connection to your database with the Python command conn.close(). That will close the connection to the database and commit your changes.
   7. You can see the results in DB Browser.
   8. If you wish to delete all the rows from *tblProducts* and try again, you can easily do so in DB Browser. On the *Execute SQL* tab, type DELETE FROM tblProducts; and press the execute button. After the query executes, click the *Write Changes* button.

Upon successful completion of these steps, you will have satisfied deliverable 2:[[4]](#footnote-4)

**DELIVERABLE 2: Populate table *tblProducts* with data from the file *Products.csv*.**

## Creating and Populating a Customers Table

Using the file *Customers.csv* that accompanies this case, create and populate a new table in the database, *tblCustomers*. Use your judgment in choosing data types for the columns and in choosing the primary key column(s).

**DELIVERABLE 3: Create and populate *tblCustomers* with data from the file *Customers.csv*.**

# Tracking Customer Orders

This section provides instructions and guidance for the next four deliverables. In sum, you will create and populate an orders table, *tblOrders1*, then answer questions about why this table is poorly designed. Then you will create and populate another orders table, *tblOrders2*, whose design rectifies the flaws in *tblOrders1*. Then you will write an SQL query to retrieve and merge data from the orders, products, and customer tables. Finally, you will create and populate two tables, *tblOrders3* and *tblOrderDetails*, that constitute the final design of our order tracking system.

For the remainder of this case, assume that all customers pay for their orders immediately in cash (i.e. no receivables).

## Customer Orders: First Try

What information should an orders table in a database store? For now, assume you want to store the order date, the customer name, the product ordered, quantity ordered, unit price paid, and the shipping date.

### Create a New Table named *tblOrders1*

Create a new table in your database named *tblOrders1*. Within that table, create the columns shown in Table 1. Remember: do not use spaces in column names.

|  |  |  |
| --- | --- | --- |
| Field | Data Type | Description |
| OrderDate | Text | Date on which the order was placed |
| CustomerName | Text | Customer name |
| Product | Text | The product ordered |
| Quantity | Integer | The number of units of the product ordered (assume no fractional units) |
| Price | Real | The unit price the customer agreed to pay |
| ShipDate | Text | The date on which the product shipped |

Table 1: Structure for table *tblOrders1*

### Refining tblOrders1

Does this table structure work? No! Are we missing something? Yes! With the current table structure, we cannot uniquely identify an order. One possible solution is to set the primary key to two existing columns, the order date and the customer name. However, that would only work if customers never place two orders on the same day.

Let’s follow the best practice of creating a separate column called *OrderID* with data type integer. Mark the column as the primary key, and also add the attribute “auto increment”. If you use DB Browser, you can add these attributes by checking the PK and AI boxes. The “auto increment” attribute tells the database to automatically generate a new order ID every time a new row is added. Neat, huh?

When you have created all the columns (the six columns in the bulleted list above, plus the *OrderID*), click the *Write Changes* button in DB Browser to save your changes.

### Populating tblOrders1 with Sample Data

Populate *tblOrders1* with the three orders shown in Table 2 below. I suggest you add these three rows manually using DB Browser. After you add the rows, don’t forget to click the *Write Changes* button.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| OrderID | Order Date | Customer Name | Product | Quantity | Price | Ship Date |
| 1 | 2018-10-10 | Company AA | Northwind Traders Chai | 100 | $18.00 | 2018-10-10 |
| 2 | 2018-10-11 | Company AA | Northwind Traders Beer | 500 | $14.00 | 2018-10-11 |
| 3 | 2018-10-11 | Company H | Northwind Traders Olive Oil | 200 | $21.35 | 2018-10-11 |

Table 2: Sample data to be inserted into *tblOrders1*

**DELIVERABLE 4:**

**a. Create table *tblOrders1* using the structure shown in Table 1, and with primary key column *OrderID*.**

**b. Populate table *tblOrders1* with the data shown in Table 2.**

**c. Explain why storing the product and customer names as text strings is inefficient, unwise, and downright stupid. What can go wrong with this design?**

**d. Propose a better design for incorporating the product and customer information into the orders table.**

## Customer Orders: Second Try

Let’s try a different design for our orders table. However, before we do, I need you to check a setting in DB Browser. Click on the tab titled *Edit Pragmas*, and make sure that *Foreign Keys* is checked.

You will now create a new database table, *tblOrders2*. To save you the grunt work of entering in the details of each column, do the following. In DB Browser, click on the *Execute SQL* tab. Paste the SQL code from Figure 5 into the appropriate box in DB Browser, and then execute it.

**CREATE** **TABLE** `tblOrders2` **(**

`OrderID` **INTEGER PRIMARY KEY AUTOINCREMENT,**

`OrderDate` TEXT**,**

`Customer` **INTEGER,**

`Product` TEXT**,**

`Quantity` **INTEGER,**

`Price` **REAL,**

`ShipDate` TEXT**,**

**FOREIGN** **KEY(**`Customer`**)** **REFERENCES** `tblCustomers`**(**`CustomerID`**),**

**FOREIGN** **KEY(**`Product`**)** **REFERENCES** `tblProducts`**(**`ProductCode`**)**

**);**

Figure 5: SQL code to create tblOrders2

Your next tasks:

* Try to add one of the rows one of the rows from Table 2 into *tblOrders2*. What happens?
* Add all the rows from Table 2 above into *tblOrders2*.
  + Instead of using the product names, use their product codes from *tblProducts*. Thus, enter Northwind Traders Chai as NWTB-1, Northwind Traders Beer as NWTB-34, and so on.
  + Instead of using customer names, use customer ID’s from *tblCustomers*.

**DELIVERABLE 5:**

**a. Create *tblOrders2* using the SQL from above.**

**b. Populate *tblOrders2* with the three rows from Table 2, but substitute product codes for the product names and customer ID’s for customer names.**

**c. What error did you receive when you tried to enter a product name instead of a product code into the Product field of *tblOrders2*? Why did you receive this error? What does it mean?**

**d. What is a foreign key? You might need to Google this.**

**e. What is a foreign key constraint?**

**f. Why is the design of *tblOrders2* superior to that of *tblOrders1*? Hint: read the section titled “*What is good database design?”* on** [**this web page**](https://support.office.com/en-us/article/Database-design-basics-EB2159CF-1E30-401A-8084-BD4F9C9CA1F5#bmgood)**.**

## Retrieving Data from Multiple Tables

Your database now has tables that stores product, customer, and order information. However, if you view the orders table, it shows product codes and customer ID’s. That’s not very helpful if you are preparing a report for your manager. In many situations, you will want to see all the rows in the orders table, but with product and customer names substituted for codes and ID’s. To make those substitutions, you will need to retrieve and merge data from all three tables. The process of retrieving and merging data from one or more tables is called a query.

**DELIVERABLE 6:**

**a. Write an SQL query that retrieves data from *tblOrders2*, *tblProducts*, and *tblCustomers* and returns a result identical to what is shown in Table 2.**

**- Hint1: use the SQL** [**SELECT**](https://www.w3schools.com/sql/sql_select.asp) **statement and** [**INNER JOIN**](https://www.w3schools.com/sql/sql_join_inner.asp) **keyword.**

**- Hint2: in your SELECT statement, qualify the field names with their table name, e.g. *tblOrders2.OrderDate*. If you used spaces in your field names, enclose the field name in square brackets, e.g. *tblOrders2.[Order Date]*.**

**b. Import *tblOrders2,* *tblCustomers*, and *tblProducts* into Python data frames. Then use the Pandas merge method to generate the same information as in part a.**

## Customer Orders: Third and Final Try

Our second design of the orders table was, unfortunately, not sufficient. It cannot handle a situation in which a customer orders multiple products in one order. This is common in retailing. For example, think about your past orders from Amazon. Did you ever order more than one product?

How can we redesign our orders table to accommodate multiple products? The naïve solution is to add multiple product columns. But this is a poor design because it imposes a maximum order size. Even if you add 20 product columns, a customer might order 25 items.

Best practice is to create two tables to store orders. The first table, *tblOrders3*, will have one record per order. The second table, *tblOrderDetails*, will have one row per product per order. Thus, there will be a one-to-many relationship between *tblOrders3* and *tblOrderDetails*.

**Deliverable 7:**

**a. Create a new table *tblOrders3* using the structure implied by the file *Orders.csv* that accompanies this case. Make sure to set a primary key column. Also set a foreign key constraint for customers. Hint: use the SQL code in Figure 5 as a template, or use the foreign key field in the DB Browser *Add Field* dialog.**

**b. Populate *tblOrders3* with data in the file *Orders.csv*.**

**c. Create a new table *tblOrderDetails* using the structure implied by the file *Order Details.csv* that accompanies this case. Also set foreign key constraints for orders *and* product codes. Hint: use the SQL code in Figure 5 as a template, or use the foreign key field in the DB Browser *Add Field* dialog.**

**d. Populate *tblOrderDetails* with data in the file *Order Details.csv*.**

# Queries: Extracting Information from the Database

Companies store data at the transaction level. When it is time to prepare financial statements, they retrieve transaction-level data from their accounting systems and aggregate it into line items, such as revenue. In this final part of the case, you will do this. Hopefully, you will see why the database structure we created aids in preparing financial statements. If you think about it, you will also see why the database structure also lends itself to auditing.

The final deliverables are all queries. You can perform the queries in SQL or in Python/Pandas – moo your choice. If you wish to use Pandas, I recommend the *read\_sql\_query* method to retrieve data from your tables. Also, [this webpage](https://www.dataquest.io/blog/python-pandas-databases/) provides a nice tutorial on interacting with SQLite databases with Python and Pandas.

**DELIVERABLES 8 – 11:**

**8. What is the company’s total revenue from orders in the database?**

**9. Compute revenue in each quarter of 2017.**

**10. Compute revenue and profit by product in 2017. Assume the company had no variances, i.e. the actual cost equaled the standard cost for all products. Which product was most profitable?**

**11. Compute revenue and profit by customer in 2017. Which customer generated the most revenue?**

1. I thank The Deloitte Foundation for their generous financial support in preparation of this case. [↑](#footnote-ref-1)
2. Many modern databases store unstructured data, such as speech, video, and text. For simplicity, and for didactic purposes, this case assumes you are working with structured data. [↑](#footnote-ref-2)
3. Manufacturing companies issue part numbers and retailers issue SKU (stock keeping unit) numbers for their products. These numbers can be used as primary keys in their databases. [↑](#footnote-ref-3)
4. You may be wondering whether it was possible to create a blank database and an empty *tblProducts* using Python. The answer is yes, and the tutorial referenced above shows you how to do so. However, I chose to have you work with DB Browser so you can more easily visualize what is happening. [↑](#footnote-ref-4)