Introduction:
The majority of financial data are stored in an organized format, referred to as a relational database. Essentially, what a relational database comprises, is a listing of an item of interest (a firm for example) and the characteristics of that item of interest. So, for instance, a primary key might be a firm-identifier: Company name, company ticker symbol, CUSIP, or, perhaps even a CIK code. In order to work with relational databases, we need to use a method for the extraction of the variables related to our item of interest. One way to do that is through something called the Structured Query Language (SQL). SQL is a pretty cool thing to know if you want to be a finance practitioner. Now, there are a couple of different platforms one can use to code in SQL. Visual Studio is one. In fact, if you are very attentive, you would have noticed that the T-Shirt I was wearing on Thursday (9/15/2016) had this logo. Visual Studio is useful, because of its flexibility: One can program in C++, Visual Basic, and even R. The problem with Visual Studio is that it is intended to be multipurpose (engineers use it, video game programmers use it, and people who build trading platforms do, too). The downside of that is one has to build all codes and routines from scratch.

The Power to Know:
SAS is the industry leader in business analytics. It is a useful skill to have on your resume, and will likely set you apart from your peers if you become proficient in it before you graduate. Furthermore, it is a slightly easier way to get acquainted with SQL; as in easier than trying to learn SQL through Visual Studio.

Programming:
Computers do not "think." They operate by sending and receiving a sequence of zeroes and ones. Programming languages provide the interface between us and the machine. Therefore, as you will discover through this brief tutorial, we have to be very specific about what we want the machine to do. Let's try to build a program that tracks the web browsing habits of Tulane students.

Relational Databases:
As mentioned earlier, relational databases are a way of describing items of interest. They are arranged in a series of rows (item of interest) and columns (characteristics of that item of interest). The columns contain variables that can take on a variety of attributes. For our purposes, the most important kinds of variables will be strings (a series of characters, like a company name, for instance), numeric (exactly what it sounds like) and dates (exactly what that sounds like). Ok, so in this example, a student named Gigi VanDenburgh is interesting. To keep track of Gigi, we are going to assign her a numeric ID, because, for example, her name might change, but for our purposes, she is still the same item of interest.

```
data table_1;
	input CustomerID FirstName $ LastName $;
	datalines;
	1 Gigi VanDenburgh
;
```

SAS creates a database called table_1.
It is then told that the columns are GiGi's ID, her first name and her last name.
By default, SAS assumes every entry is numeric, so we put the $ sign after the variables *FirstName* and *LastName* to let it know that these are strings.
Go to your work folder and open up table_1.

We kind of got what we wanted, but if you look at the *LastName* column, her name has been truncated. By default, SAS assumes strings to be 8 characters in length. This is a throwback feature to do with something called bits. Fun fact: Back in the day, there was a game console called the Nintendo 64 that was so cool, because its processor was 64 bits; but I digress.

```
data table_1;
       input @5 CustomerID @7 FirstName $ @12 LastName $11.;
       datalines;
       1 Gigi VanDenburgh
;
```

There are two key differences with this snippet of code:

1. First of all, we told SAS where the different columns start.
   Look at the text in yellow, and move your cursor to the *CustomerID* column.
    - Now look at the bottom right corner of your SAS editor screen. The "Col 5" means we started at Column 5.
    - Now, move your cursor over to Gigi and look at the bottom right corner. Her first name starts at column 7.
    - Now, move your cursor to VanDenburgh, and it should say "Col 12."

2. By putting 11. next to the string ($), SAS now knows that strings (a person's last name) can be as long as 11 characters. Make sure you include the period after the number 11.

We have identified Gigi, but we need a date for whatever interesting thing she was doing, as well. SAS does not count time the way that we do. It counts time based on an algorithm. So, every SAS date is a day relative to January 1st, 1960. Just like with string variables, we have to tell SAS that we are now entering a date variable. We also have to tell it in which column the date starts. There are a variety of ways of entering dates, but 2-digit months, 2-digit days and 2-digit years seem reasonable to me. So, what was Gigi doing on October 4th, 2016 (10/04/16)?

```
data table_1;
       informat CalendarDate mmddyy.;
       input @5 CustomerID @7 FirstName $ @12 LastName $11. @24 CalendarDate;
       datalines;
       1 Gigi VanDenburgh 100416
;
```

The ID and name columns look fine, but the date looks strange. Take your calculator and do this: 1960 + (20731/365). That answer should look a lot more like what we were aiming for. We need to tell SAS that not only is the data in mm-dd-yy format, we also would like to see it that way.

```
data table_1;
       informat CalendarDate mmddyy.;
       input @5 CustomerID @7 FirstName $ @12 LastName $11. @24 CalendarDate;
       format CalendarDate mmddyy.;
       datalines;
       1 Gigi VanDenburgh 100416
;
```

Success!

The concept of time of day is similar to the date concept. We specify the informat first, the start of the column and then the format we want to see it in. In finance, the 24-hour clock makes sense, because it avoids confusion between AM and PM.

```
data table_1;
       informat CalendarDate mmddyy.;
       informat LogInTime time.;
       input @5 CustomerID @7 FirstName $ @12 LastName $11. @24 CalendarDate @31
LogInTime;
       format CalendarDate mmddyy.;
       format LogInTime time.;
       datalines;
       1 Gigi VanDenburgh 100416 08:15
;
```

So, we have 8:15 AM and :00 seconds. Perhaps later in the semester, we will work with transactional data that requires even more accurate time stamps than that. For now, this is great.

Next up, we want to know what Gigi was doing at 8:15 in the morning.

```
data table_1;
       informat CalendarDate mmddyy.;
       informat LogInTime time.;
       input @5 CustomerID @7 FirstName $ @12 LastName $11. @24 CalendarDate @31
LogInTime @37 HTTP $;
       format CalendarDate mmddyy.;
       format LogInTime time.;
       datalines;
       1 Gigi VanDenburgh 100416 08:15 www.lululemon.com
;
```

Whoops! Think to yourself why the table does not look right.

```
data table_1;
       informat CalendarDate mmddyy.;
       informat LogInTime time.;
       input @5 CustomerID @7 FirstName $ @12 LastName $11. @24 CalendarDate @31
LogInTime @37 HTTP $22.;
       format CalendarDate mmddyy.;
       format LogInTime time.;
       datalines;
       1 Gigi VanDenburgh 100416 08:15 www.lululemon.com
;
```

Let us now track what Gigi was doing throughout the day:

```
data table_1;
       informat CalendarDate mmddyy.;
       informat LogInTime time.;
       input @5 CustomerID @7 FirstName $ @12 LastName $11. @24 CalendarDate @31
LogInTime @37 HTTP $22.;
       format CalendarDate mmddyy.;
       format LogInTime time.;
       datalines;
       1 Gigi VanDenburgh 100416 08:15 www.lululemon.com
       1 Gigi VanDenburgh 100416 08:20 www.facebook.com
       1 Gigi VanDenburgh 100416 10:17 www.taylorswift.com
       1 Gigi VanDenburgh 100416 12:30 www.lululemon.com
       1 Gigi VanDenburgh 100416 15:45 www.tulane.gibson.edu
;
```

Ok, that is very interesting, but it would also be useful to know how long she was on each website. The tab stops (@) are at different places in the example. Tab stops, indents and comments (the green text that I put in each program) are all a matter of personal preference. Generally, good coding practice is to make your program as legible as possible, so that if someone else has to modify it, they can clearly see what is going on.

```
data table_1;
       informat CalendarDate mmddyy.;
       informat LogInTime time.;
       input @5 CustomerID @9 FirstName $ @17 LastName $11. @33 CalendarDate @41
LogInTime @49 HTTP $22. @73 MinutesSpent ;
       format CalendarDate mmddyy.;
       format LogInTime time.;
       datalines;
       1     Gigi   VanDenburgh  100416        08:15 www.lululemon.com         3
       1     Gigi   VanDenburgh  100416        08:20 www.facebook.com          27
       1     Gigi   VanDenburgh  100416        10:17 www.taylorswift.com       22
       1     Gigi   VanDenburgh  100416        12:30 www.lululemon.com         75
       1     Gigi   VanDenburgh  100416        15:45 www.tulane.gibson.edu     3
;
```

How about the web browsing habits of a couple of other people?

```
data table_1;
       informat CalendarDate mmddyy.;
       informat LogInTime time.;
       input @5 CustomerID @9 FirstName $ @17 LastName $11. @33 CalendarDate @41
LogInTime @49 HTTP $22. @73 MinutesSpent ;
       format CalendarDate mmddyy.;
       format LogInTime time.;
       datalines;
       1      Gigi   VanDenburgh   100416 08:15  www.lululemon.com        3
       1      Gigi   VanDenburgh   100416  08:20 www.facebook.com         27
       1      Gigi   VanDenburgh   100416  10:17 www.taylorswift.com      22
       1      Gigi   VanDenburgh   100416  12:30 www.lululemon.com        75
       1      Gigi   VanDenburgh   100416  15:45 www.tulane.gibson.edu    3
       2      Emma   Siegel        100416 08:17  www.harrahs.com          12
       2      Emma   Siegel        100416 09:35  www.cnn.com              7
       2      Emma   Siegel        100416 12:45  www.facebook.com         49
       2      Emma   Siegel        100416 16:12  www.facebook.com         51
       2      Emma   Siegel        100416 20:07  www.tulane.gibson.edu    35
       3      Gabriella Rizack     100416 07:17  www.tulane.gibson.edu    18
;
```

What's the problem here?

So, since there are first names longer than Gigi and Emma, let's be on the safe side and go one character longer than Gabriella's name.

```
data table_1;
       informat CalendarDate mmddyy.;
       informat LogInTime time.;
       input @5 CustomerID @9 FirstName $10. @21 LastName $11. @33 CalendarDate @41
LogInTime @49 HTTP $22. @73 MinutesSpent ;
       format CalendarDate mmddyy.;
       format LogInTime time.;
       datalines;
       1      Gigi         VanDenburgh   100416 08:15  www.lululemon.com         3
       1      Gigi         VanDenburgh   100416  08:20 www.facebook.com          27
       1      Gigi         VanDenburgh   100416  10:17 www.taylorswift.com       22
       1      Gigi         VanDenburgh   100416  12:30 www.lululemon.com         75
       1      Gigi         VanDenburgh   100416  15:45 www.tulane.gibson.edu     3
       2      Emma         Siegel        100416 08:17  www.harrahs.com           12
       2      Emma         Siegel        100416 09:35  www.cnn.com               7
       2      Emma         Siegel        100416 12:45  www.facebook.com          49
       2      Emma         Siegel        100416 16:12  www.facebook.com          51
       2      Emma         Siegel        100416 20:07  www.tulane.gibson.edu     35
       3      Gabriella    Rizack        100416 07:17  www.tulane.gibson.edu     18
       3      Gabriella    Rizack        100416 07:58  www.facebook.com          45
       3      Gabriella    Rizack        100416 10:30  www.reuters.com           24
       3      Gabriella    Rizack        100416 12:30  www.tulane.gibson.edu     9
       3      Gabriella    Rizack        100416 17:11  www.nytimes.com           53
;
```

In a marketing context, demographic data can be exceedingly valuable. That is why pretty much every transaction you engage in winds up in a relational database somewhere. So, let's see what a different group of students has been getting up to:

```
data table_2;
       informat CalendarDate mmddyy.;
       informat LogInTime time.;
       input @5 CustomerID @9 FirstName $10. @21 LastName $11. @33 CalendarDate @41 LogInTime
@49 HTTP $22. @73 MinutesSpent ;
       format CalendarDate mmddyy.;
       format LogInTime time.;
       datalines;
       4       Jacob          Kaplan         100416  08:45   www.codeacademy.com    33
       4       Jacob          Kaplan         100416  09:45   www.pro.benzinga.com   30
       4       Jacob          Kaplan         100416  10:22   www.atlantabraves.com  10
       4       Jacob          Kaplan         100416  12:30   www.tulane.gibson.edu  7
       4       Jacob          Kaplan         100416  18:45   www.facebook.com       10
       5       Jared          Darvin         100416  08:17   www.espn.com           12
       5       Jared          Darvin         100416  09:30   www.adidas.com         5
       5       Jared          Darvin         100416  09:45   www.facebook.com       5
       5       Jared          Darvin         100416  12:12   www.facebook.com       3
       5       Jared          Darvin         100416  20:07   www.tulane.gibson.edu  15
       6       Alex           Barroukh       100416  07:00   www.tmz.com            25
       6       Alex           Barroukh       100416  07:26   www.taolasvegas.com    8
       6       Alex           Barroukh       100416  07:35   www.pro.benzinga.com   20
       6       Alex           Barroukh       100416  07:56   www.tulane.gibson.edu  10
       6       Alex           Barroukh       100416  08:07   www.latimes.com        15
;
```

Interesting…

SQL:

Now, in real life, we would not have entered all this data by hand. We could then use SQL to organize what was automatically collected into a column format that makes sense to us. For example, if you open table_1, it put the date and time-stamp first. I might think Gigi's identity is the most important thing about her, and want that to be the first column in my database. It might also make more sense to have the website first, then the time she logged into it, and then the number of minutes she spent surfing.

A SQL script typically has the following structure:

Create a new table.
Select the variables of interest, in the order that makes sense.
Perform operations on those variables.
Quit.

```
proc sql;
       create table Ladies as select
              CustomerID,
              CalendarDate,
              FirstName,
              LastName,
              HTTP,
              LogInTime,
              MinutesSpent
       from table_1;
quit;
```

Table_1 is a generic name, so we will name our SQL table, "Ladies."

Our variable names are also pretty generic. So, it is a good idea to assign labels, so that other people can understand what information it is that they are looking at. Also, maybe the person this database is intended for does not like mm-dd-yy. Maybe she, or he prefers dates written out in full. SQL gives you the flexibility to perform myriad manipulations on what you would like the final output to be.

```
proc sql;
      create table Ladies as select
            CustomerID label = 'Customer Identification Number',
            CalendarDate format worddate20. label = 'Calendar Date',
            FirstName label = 'First Name',
            LastName label = 'Last Name',
            HTTP label = 'Web Address',
            LogInTime label = 'Time of Login',
            MinutesSpent label = 'Minutes Browsing Web Address'
      from table_1;
quit;
```

We will now create a similar-looking table for our male demographic:

```
proc sql;
      create table Gents as select
            CustomerID label = 'Customer Identification Number',
            CalendarDate format worddate20. label = 'Calendar Date',
            FirstName label = 'First Name',
            LastName label = 'Last Name',
            HTTP label = 'Web Address',
            LogInTime label = 'Time of Login',
            MinutesSpent label = 'Minutes Browsing Web Address'
      from table_2;
quit;
```

If you look at the Ladies and Gents tables, you will notice that they are arranged according to the identity of the customer. This does not necessarily have to be the case, though. We could use the SQL command order by to pick a different variable of importance. Instead of the identities, we could sort our table in alphabetical order:

```
proc sql;
      create table Ladies as select
            CustomerID label = 'Customer Identification Number',
            CalendarDate format worddate20. label = 'Calendar Date',
            FirstName label = 'First Name',
            LastName label = 'Last Name',
            HTTP label = 'Web Address',
            LogInTime label = 'Time of Login',
            MinutesSpent label = 'Minutes Browsing Web Address'
      from table_1
      order by LastName;
quit;
```

Concatenation, in a database context, is the process of stacking two tables that have the same variables on top of each other:

```
proc append
      base = Ladies
      data = Gents;
run;
```

If you open up the Ladies table, we have information about everyone in our sample, but the name Ladies no longer makes sense. Let's create a new table called Customers. We will also use a SQL wildcard. Wildcards are a convenient way to not have to type out everything all over again, when we are selecting variables. They have other uses, too; for instance, when you don't know the exact name of something, but you have a rough idea and need to perform a search.

```
proc sql;
	create table Customers as select
		*
	from Ladies
	order by CustomerID;
quit;
```

There are several tables we have that we no longer need. All that matters is the full set of customers. So, we can now use the SQL drop command to delete the unnecessary tables.

```
proc sql;
	drop table Table_1, Table_2, Ladies, Gents;
quit;
```

SQL aggregate functions allow us to calculate many interesting things, like, for instance; what is each student's most visited website? The SQL group by statement is extremely important. It tells SQL at what level we need to aggregate the data. In this case the grouping for the sum function is who the student is (Gigi, for example) and the total minutes she spent on each website (HTTP).

```
proc sql;
	create table BrowsingHabits as select
		*,
		sum(MinutesSpent) as Total label = 'Total Minutes on Web Address'
	from Customers
	group by CustomerID, HTTP
	order by CustomerID;
quit;
```

As you can see from the output, Gigi spent 3 minutes on www.lululemon.com when she woke up, and an astonishing 1 hour and 15 minutes on that same website from 12:30 onwards on Tuesday, October 4th, 2016 (lecture time). However, there is a lot of duplicate data. So, we use the SQL distinct switch to cut it down to each customer's favorite website on October 4th, 2016. The SQL having clause just lets the program know that I want the maximum value for the aggregate variable Total.

```
proc sql;
	create table Targeted_Advertising as select distinct
		CustomerID,
		CalendarDate,
		FirstName,
		LastName,
		HTTP,
		Total
	from BrowsingHabits
	group by CustomerID
	having max(Total) = Total;
quit;
```

Fin!