Lecture 1 [Building a Relational Database]

Gwinyai T.

May 15, 2017

1 Building a Relational Database

Financial data are typically stored in an organized container, known as a relational database. A straightforward example could be a list of publicly traded companies (items of interest) and the associated stock prices and dividends (characteristics of the items of interest). Formalizing the connection between each list member and its variables is accomplished with the use of (ideally) unique identifiers. Ticker symbols, CUSIP numbers and CIK codes are some common candidates, with each providing a different level of specificity.

The work of an investor, analyst, or academic may involve the augmentation of extant stores of structured data with information gleaned from unstructured sources; sentiment analysis being a case in point. Nonethless, in each of the aforementioned roles, the deliverable rendered to the client invariably rests on the schema of a relational database. As such, modeling in financial economics ipso facto demands proficiency in a software platform that can transform raw data into actionable intelligence. This series of lectures uses the Structured Query Language (SQL) for that purpose.

1.1 DBMS

The core asset in finance is data. However, data without some semblance of order lack meaning. A database managament system (DBMS) is software that interpets the sequences of ones and zeros with which hardware stores content, and presents them in a sensible format to the end-user. A relational database management system (RDBMS) is a specific type of DBMS, with the distinguishing feature that each entry can be drilled down to an indivisible placeholder. Thus, an RDMS precludes the existence of many-to-one associations. Absent a primary key, the ticker symbol AA, say, would be invalid as a reference to equity offerings floated by both the Aluminum Company of America and Alcoa Inc; even though they are the same firm.

Once interpreted, data may be retrieved, updated or destroyed according to the user's needs. For a variety of reasons, the structural query language, as implemented through SAS, makes a good choice for these tasks. First, SAS/SQL is safe, meaning that permissible instructions conform to a stable, well-tested

version of the base language. As of this writing, *proc sql* (SAS's invocation of SQL), runs on the *ANSI-99* release. Second, SAS is presently the industry leader in business analytics. While a standalone SQL installation comes with a rich set of data management tools, discipline-specific routines need to be fed to an external package. SAS, on the other hand, houses data manipulation and statistical analyses in a seamless development environment. Third, knowledge of SAS is a valued commodity in the job market owing to its ubiquity in both practitioner and scholarly settings.

To avoid getting bogged down by taxonomic minutiae, it will suffice to think of a row as a single instance of an item of interest. The columns intersecting the row describe what is known about the item of interest in that instance. Columnar variables in SQL take on a handful of formats that should be familiar to a reader with prior programming experience. A brief, but non exhaustive list includes string types (alphanumeric sequences); integer, decimal and float (strictly numeric sequences, with varying degrees of mathematical precision); date (a data type that instructs the machine to treat entries within a column as references to the time dimension).

1.2 Manual Data Entry

The objective of this lecture is to create two tables. The first will store details about a select group of individuals. The second will contain market data on a handful of publicly traded US firms. In the event that the revealed preferences of the people in table one have some association with the stocks in table two, a composite table will yield valuable insights about both classes of entities. As a start, enter the program below in the SAS *Editor* and submit the statements for processing (click on the icon with the running man).

```
/*  ID, First Name, Last Name */
proc sql;
  create table _student01
    (ID int label = 'Identification Number',
      firstName char label = 'First Name',
      lastName char label = 'Last Name');
      insert into _student01
      values(1, 'Gigi', 'V')
  ;
quit;
```

Assuming everything was entered correctly, the SAS *Work* folder should have a new member. Opening it reveals that there is a person named Gigi V, with an identification number of one. The first line of the program, known as a comment, serves no operational purpose. Rather, it explains in clear terms what the fragment of code does. SAS automatically skips over text enclosed in countervailing slashes and asterixes. Nonetheless, regular commenting is considered good coding practice. Comments do not impose a cost on system resources, and as programs become more complex, they are a useful guide to the application designer's original intent.

The create command instantiates a table, while the instructions in parentheses define its layout. In this example, there are three variables and two data types. From a user standpoint, the relative importance of each variable depends on the question being asked. The DBMS designer, in contrast, assigns primacy to the unique identifier; *ID* here. In so doing, the other two (or more) columns may vary over time. Consider a name change. No conflict occurs, because the entity of interest remains the same. George Orwell instead of Eric Arthur Blair, or Alcoa instead of the Aluminum Company of America (see section 1.1), for example.

Duplicate row entries referencing disparate concepts are invalid. To wit: Gigi V of New Orleans, LA and Gigi V of Des Moines, IA cannot share an identification number if they are, in fact, different people. Gigi V, with a work address in New Orleans, and the same Gigi V with a new work address in Des Moines can both be identified by the number one, however. Likewise, a change in headquarters for Alcoa only has a material impact on the location column, provided a couple of conditions are fulfilled. The first is that the highest degree of precision necessary happens at the company level. A scenario in which Alcoa constructs its flagship facility with the issuance of new securities (that we wish to find out more about) would require a more specific key. The second is that the relocation is not the result of an existential change, like a corporate takeover. In that case, the new firm would be something entirely different, which would also mandate the creation of a new key.

Having told SAS to expect an integer for the identification number and strings for his/her first and last name, it will attempt to reproduce the values as entered. One should note that the data type, *char*, has a built-in constraint. By default, *char* limits strings to a maximum of eight characters. This is a legacy feature of the, then state of the art, 8 bit architecture IBM introduced with the release of the System/360 mainframe in 1964. With the one row thus far, this restriction is not a problem. Had SAS encountered a longer name, it would have truncated the entry. Test out the code below and make a note of what happens to Gabriella.

```
/*  8 Bits
  Newer SQL releases (Microsoft SQL Server 2008 and later) do not
    require multiple calls to the values command
*/
proc sql;
  drop  table _student01;
  create table _student01
    (ID int label = 'Identification Number',
      firstName char label = 'First Name',
      lastName char label = 'Last Name');
      insert into _student01
      values(1, 'Gigi', 'V')
      values(2, 'Gabriella', 'R')
  ;
quit;
```

Thereafter, increase the allotted memory using the data type *varchar* (variable character), with a length of 32 bits.

```
/*  32 Bits
  Gregorian Calendar
  SAS has a "birth date" of January 1st, 1960
  Populate with more individuals
*/
proc sql;
  drop  table _student01;
  create table _student01
    (ID int label = 'Identification Number',
      firstName varchar(32) label = 'First Name',
      lastName char label = 'Last Name',
      calendarDate int format yymmdd10. label = 'Calendar Date');
      insert into _student01
      values(1, 'Gigi', 'V', 20731)
      values(2, 'Gabriella', 'R', 20731)
      values(3, 'Emma', 'S', 20731)
      values(4, 'Jacob', 'K', 20731)
      values(5, 'Jared', 'D', 20731)
      values(6, 'Alex', 'B', 20731)
  ;
quit;
```

This final version of *_student01* has more people, and a curious looking date value. Time is perhaps best visualized as a count relative to an anchor point. Years in the Gregorian calendar are enumerated as belonging to either the Common Era (CE) or the Before Common Era (BCE). SAS, is of a more recent vintage, and counts time as the number of days since its "birthday"; January 1st, 1960. The reader is, of course, welcome to operate under the principle that time is an illusion. Unfortunately, SAS has not released an update to handle that yet. Therefore, it shall be agreed that this lecture was first delivered 20,731 days after SAS was born. For readability, a format of 4-digit year, 2-digit month and 2-digit day is chosen.

1.3 Retrieving Data from an Existing Source

Data are typically retrieved via subscription to a provider. Apart from financial data vendors, for whom it is their raison d'être, hand-collection is impractical for most purposes. With that said, competitive pressures, low barriers to entry and ease of duplication quickly attenuate the advantages of prepackaged information, and its attendant programming routines. Automated parsing of unstructured content is a supplementary method for uncovering patterns beyond what hard data alone can reveal. This topic will be addressed in a later lecture. For now, the discussion centers on conventional approaches.

The exercises in this course are written on the Center for Research in Securities Prices (CRSP), and Standard and Poor's Compustat databases. CRSP contains historical market quotes on U.S. equities, exchange traded funds (ETFs), real estate investment trusts (REITs) and American Depository Receipts (ADRs). Compustat provides accounting data on publicly traded U.S. firms. It is possible to arrive at all of the solutions in this text using free sources. In fact, that might even be a fun way to test one's programming skills. Nonetheless, the

assumption going forward is that the reader has paid for CRSP and Compustat access.

To recap, the financial world is made up of six people. The goal in this subsection is to extract a matching number of real companies, namely: Exxon Mobil (XOM: NYSE), The New York Times Company (NYT: NYSE), Time Warner (TWX: NYSE), C.B. Richard Ellis (CBRE: NYSE), Delta Airlines (DAL: NYSE) and Lululemon Athletica (LULU: NASDAQ). Plain language markers do not work well as primary keys. Name changes, variations in spelling, duplicates and divisibility of the represented entity all fail the uniqueness requirement. It is for these reasons that finance practitioners prefer ticker symbols when communicating about securities. The reader should be aware, though, that tickers can be reused, requiring a mapping.

The block of code below extracts rudimentary information about the six companies. At this stage, the query is just exploratory, and so a table is not created. Before executing it, change the library name *a_stock* to the location of CRSP on your machine. A successful run returns the firms' tickers (*ticker*), plain language names (*comnam*), and the periods over which the names were in use (*nameDt* to *nameEndDt*). As the semester progresses, you may find it useful to keep variable definitions close to hand. CRSP maintains a glossary of its data items here.

```
/*  Data from an existing source */
proc sql;
    select
      monotonic() as rowIndex,
      ticker, comnam,
      nameDt, nameEndDt,
      permno
    from a_stock.stocknames
    where ticker in('XOM', 'NYT', 'TWX', 'CBRE', 'DAL', 'LULU');
quit;
```

The output fulfills the request, with each observation ordered by the user-defined variable *rowIndex*. Behind the scenes, your machine generated a system message indicating a succesful execution. However, almost all the records warrant further inspection. Taken at face value, the HTML output in the SAS *Results Viewer* suggests that: Exxon Mobil came into being on December 1st, 1999; the New York Times and Nytronics Incorporated are the same company; Delta Airlines did not exist between October 12, 2005 and May 3rd, 2007. As it happens, all the data are correct, but the inferences drawn could be misleading. Refer to the CRSP glossary, search for *permno* and then enter the next code snippet.

The difference between the present and prior release is that *stocknames* is subset with *permno*. A security's *permno* is a numeric data type and must be entered as is. String variables like *ticker*, on the other hand, are only processed when enclosed in single or double quote marks.

```
/*  permno as primary key */
proc sql;
    select
      monotonic() as rowIndex,
      ticker, comnam,
      nameDt, nameEndDt,
      permno
    from a_stock.stocknames
    where permno in(92203, 47466, 77418, 90199, 11850, 91926);
quit;
```

This run generates results that are much closer to the desired companion table for *_student01*. The full date range of Standard Oil, later Exxon Mobil, is visible. The marker for the New York Times corresponds to the newspaper, and not a mechanical and electronic firm. Finally, observations of Delta Airlines will be restricted to the post-bankruptcy period.

When speaking colloquially, one might refer to shares of Exxon Mobil as just Exxon, or mistakenly call Time Warner's stock by its old name, AOL Time Warner. This does not present a problem since a listener would quickly hone in on meaning through context. Thanks to the higher granularity of *permno* over *ticker*, your machine can do that, too. The next step is to eliminate redundant information. SQL's *group by* clause allows for selective processing based on a shared characteristic. The *max()* function returns the maximum value of a variable in a column, or column subgrouping. To conclude this lesson, create a table that pulls the six unique identifying symbols, but only keeps the observation at the end of each group. The idea here is that the *max(rowIndex)* within each group will correspond to the most recent name, and likely be what users find familiar.

```
/*  Duplicates */
proc sql;
  create table _stock01 (drop = rowIndex)
    as select
      monotonic() as rowIndex,
      permno, comnam, ticker
    from a_stock.stocknames
    where permno in(92203 ,47466, 77418, 90199, 11850, 91926)
    group by permno having max(rowIndex) = rowIndex;
quit;

/*  Fin!  */
```

Verify your results, save your code and exit SAS.