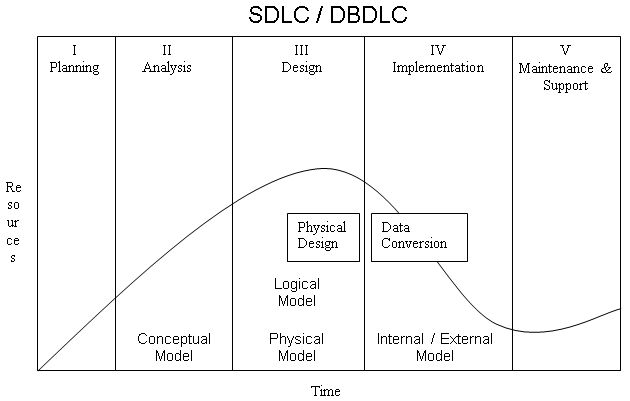
# Class Exercise, Unit 11: Data Migration

This week we’ll look at how to move data from existing designs into our internal data model. If you look at our methodology you can see that we are still in the design phase, and that data conversion falls in the implementation phase.   
  


You are here

## Part 1: Common Data Migration Techniques

In this part we will discuss various strategies for converting and migrating data from different data sources. We will take a look at some approaches for using SQL to convert data and move it into normalized tables.

### What is data Migration?

**Data migration** represents the process of transferring data from one data source to another. This can be a one-time process, as in the case of improving database design through normalization, or a routine, scheduled process such as a feed from a transactional database into a data warehouse. The data sources are usually heterogeneous (not the same system or DBMS). Database **middleware** such as ODBC, OLEDB, and JDBC greatly aid the process of data migration as they make the process of reading and writing the data a little more trivial.   
  
For any given data migration from a data source to a target, there are five phases:

1. **Extraction.** In the extraction phase, data is pulled out of the data source. Depending on the type data source, this can be a trivial or arduous process. For example is the source the same or another DBMS, middleware can be used to easily perform the extraction. If the data source is free-form text, custom programming will be required to accurately extract structured data.
2. **Cleansing.** Data cleansing involves cleaning up the extracted data source so that it will be accepted more readily by the target system. If your target system uses check constraints or defaults but they aren’t in the source data, you’re need to manipulate the source data so that is will be acceptable to the target system.
3. **Transformation.** Data transformation involves breaking up the source data structure into a form that will be acceptable to the target data structure. For example if the source data is full name such as “Michael Fudge”, but the target data has two columns for this: First Name and Last Name then we need to split up the names in the data transformation process. During data transformation we build lookup tables, and copy data from one source table to multiple target tables as the result of data normalization.
4. **Loading.** During data loading, we import our extracted, cleansed and transformed data neatly into our target tables (or other data stores). When using SQL commands for data migration, you can typically transform, cleanse data, and load data all in one step. If the cleansing and transformation occurs outside the target DBMS, we use database middleware to perform the loading.
5. **Verification.** Possibly the most important step in the process is data verification. Data verification involves checking whether the data was migrated completely and correctly. On targets that are relational DBMS we can use foreign keys and unique constraints along with row counts to perform data verification. With automated, scheduled data migrations such as data warehouse feeds, the entire migration is nested in a database transaction so that a failure in data verification allows us to rollback any changes made to the target DBMS.

### A General Data Migration Approach for DBMS targets

When performing a data migration from an external data source into a target DBMS, such as Oracle or SQL server, we can take advantage of the features of the DBMS to help with the data cleansing, transformation, loading and verification phases. The general approach for a one-time data migration is as follows:

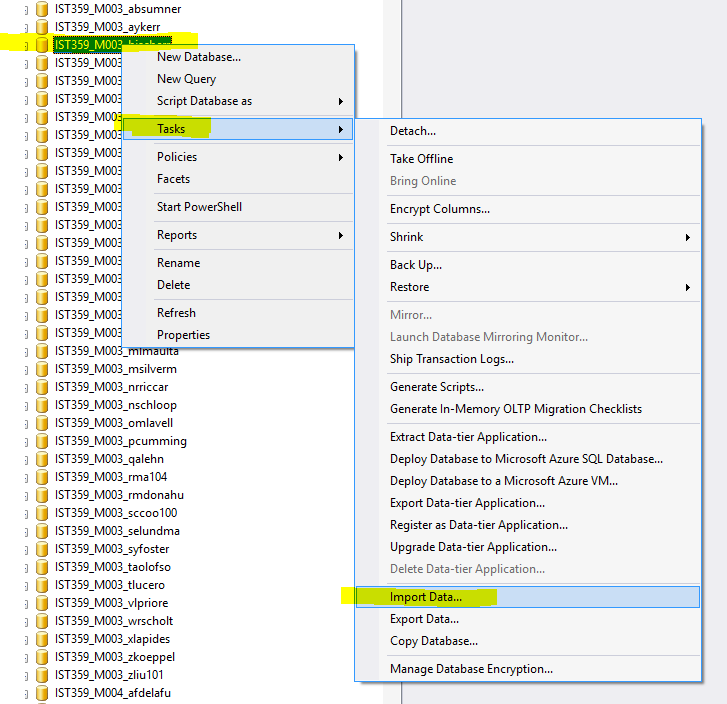
1. Normalize the table(s) you start with.
2. Extract and bulk-load the **data source** into the target DBMS **as-is**. These are now referred to as the **SOURCE TABLE(S).**
3. Next, cleanse the **data source tables** using SQL **UPDATE** statements.
4. Transform and load source data into target tables using **INSERT INTO table SELECT ...** statements
5. Verify the data is correct using row counts and query joins **SELECT COUNT(\*) ...**

### Data Migration Walk-Thru: Simulation we will run in class

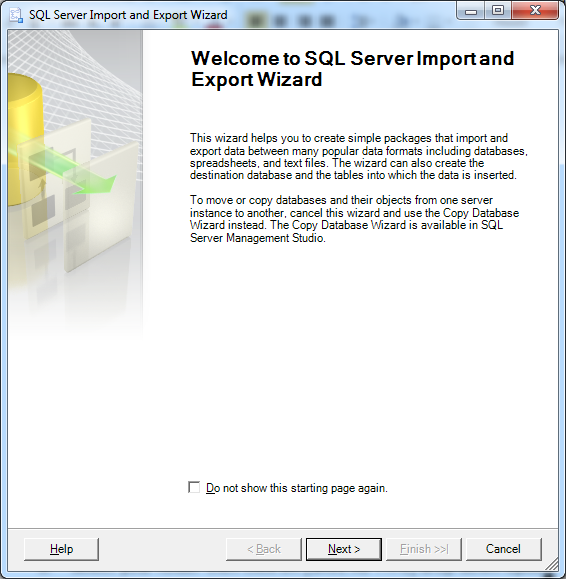
Step one: Import Source Data into SQL Server

Your first task will be to import the given spreadsheet as-is into your SQL server database. To accomplish this, you will use the Import and Export Data wizard in SQL Server 2016 to connect to the data source.

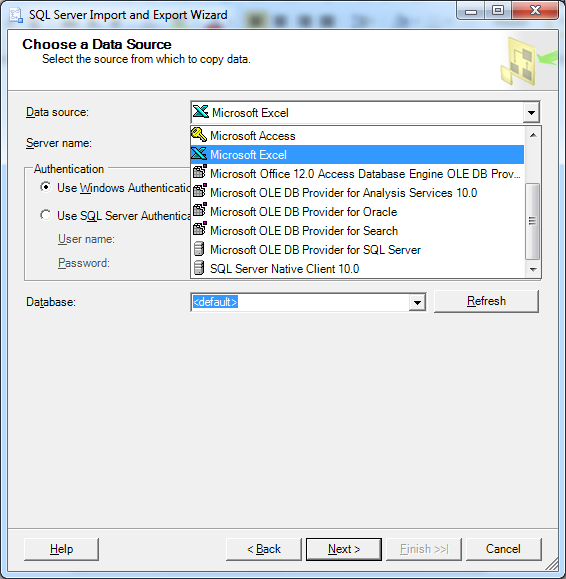
1. Open SQL Server Management Studio and connect to your SQL server instance. Expand the Databases tree and scroll down to your personal database. Right click on your database and select Tasks -> Import data… as shown:



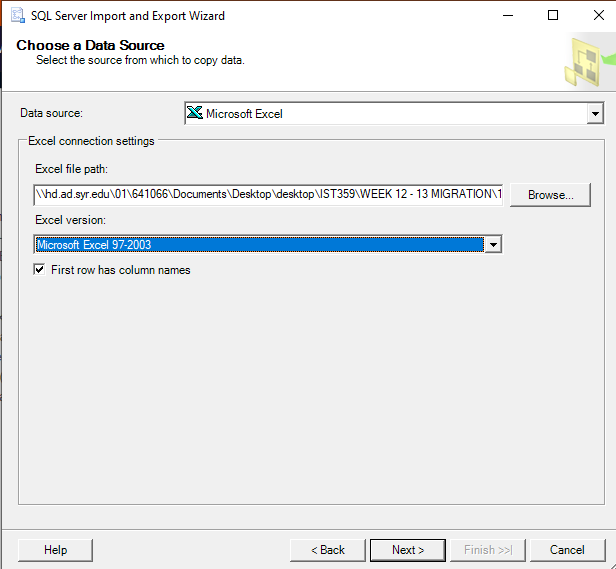
1. You will be prompted with the following screen, click next.



1. You are now prompted to choose the source of your data. Since you are importing data **from** an Excel spreadsheet, select Microsoft Excel from the drop down menu.

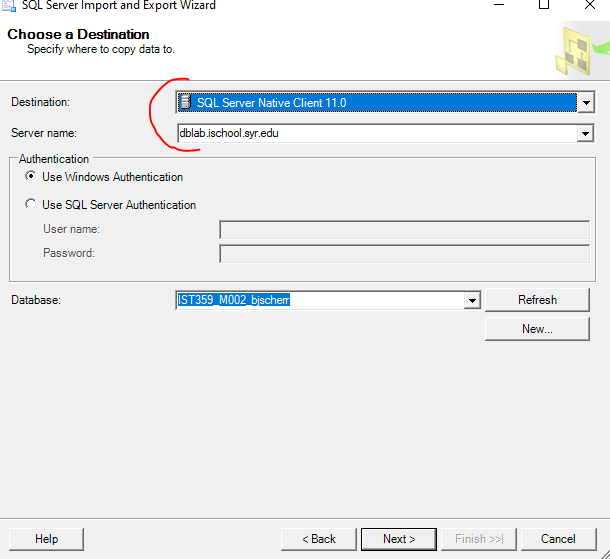


1. Next, browse to the location on your computer where you have saved your lab data files and select the Drivers Excel spreadsheet. **UPDATE**: There is a compatibility issue (known) with the iSchool dblab server, so do the following:
   1. In the Excel file you want to bring in, be sure to do a ‘Save As’ and save it as a .xls file (that’s a much older version of Excel)
   2. When you bring in the .xls file for your import (below screenshot) make sure you’ve selected ‘Microsoft Excel 97 – 2003).
   3. iSchool ITS is aware of this issue, and *this is the temporary workaround*.



Notice that “First row has column names” is checked. Check that since our data set has column names. *Click next.*

1. You will be prompted to choose a destination for the data.

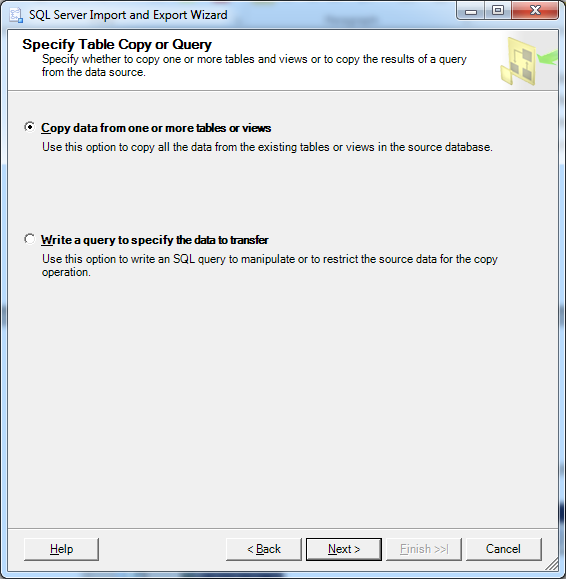


Since you are importing data **into** SQL Server, set the destination as **SQL Server Native Client 11**. Make sure the server name is set to dblab.ischool.syr.edu.

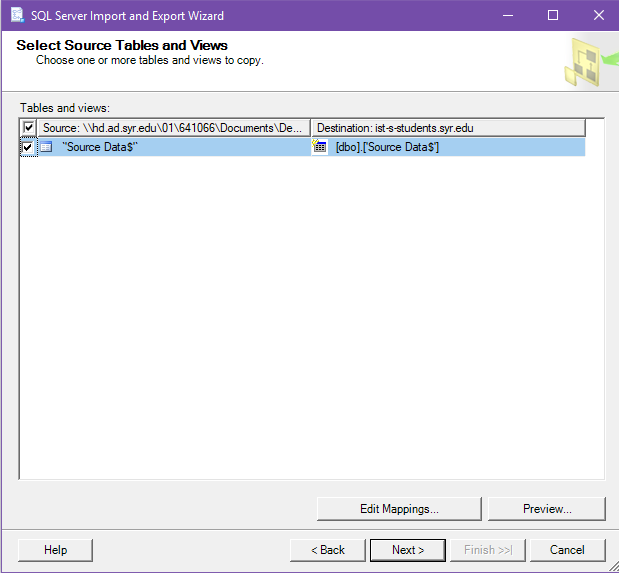
Leave the “Use Windows Authentication” box checked. The wizard should default to your database.

Click next.

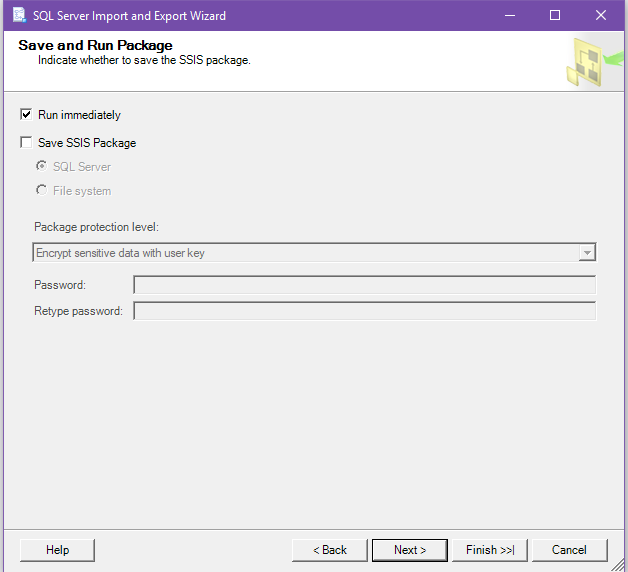
1. Make sure that Copy data from one or more tables is selected. Click next.

  
Click Next.

1. Note the source and destination names below**.** After a successful import, the table **dbo.’Source Data$’** should appear in your database. Note: SQL Server will append a $ to the end of the imported table name to indicate that the source of the data was a Microsoft Excel spreadsheet. You may also note the **EDIT MAPPINGS** option. That is a very useful but powerful option that allows you to change field names, data types, and whether or not NULLS are allowed. Some of this can be handled on the Excel end, some here, and always via an ALTER COMMAND in SQL. In this lab we will NOT MAKE any changes, but you should play around w/ this feature. Click next.

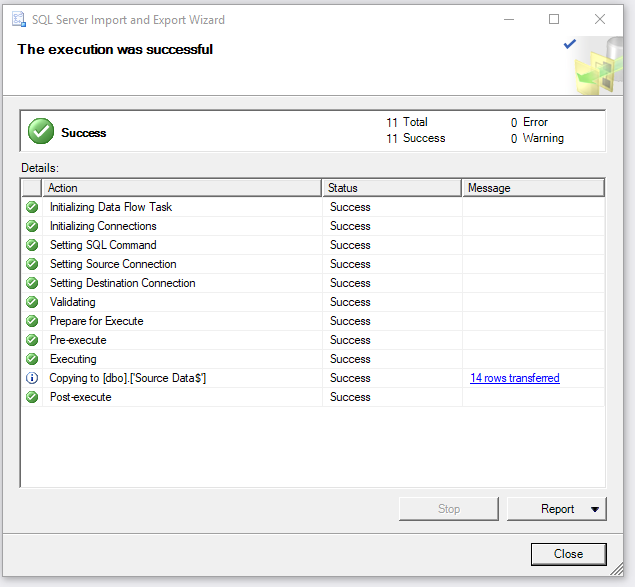


1. Make sure Run immediately is selected. **Click next**.



At this point, you will be presented with a final confirmation screen, verify your information is correct and then click **finish.**

If the data migration has completed successfully, your screen should look like this and you should note that it transferred the same number of rows that were contained in the original Excel file. Be sure to verify this!

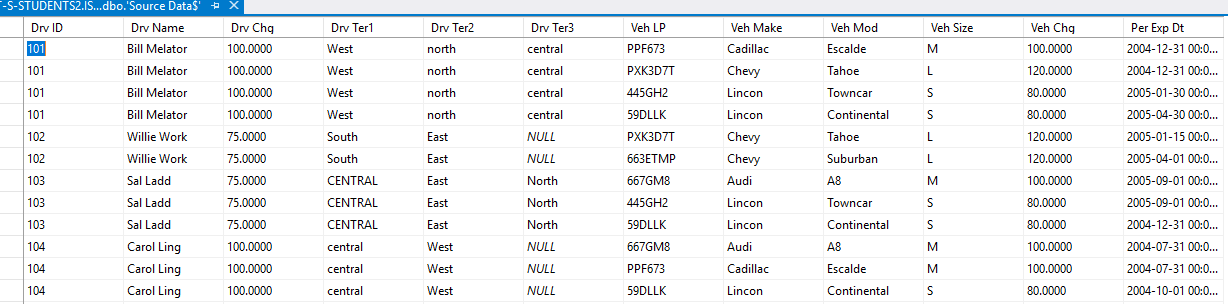


1. Refresh your database tables and verify that the **… $’** table exists. It will end in a $ symbol. It should look like this in your object browser depending on what you named it:

Graphical user interface, text

Description automatically generated

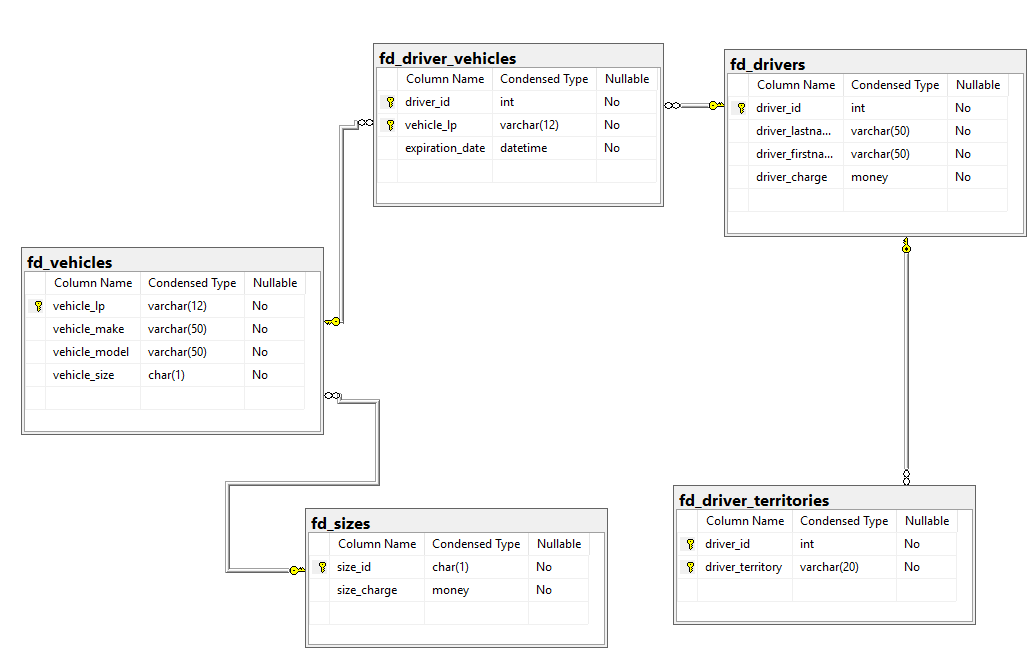
The data should look like this:



If it does, congratulations! You have successfully imported the spreadsheet into SQL Server and are now ready to proceed with the rest of the exercise.

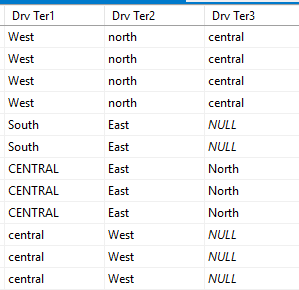
Step two: Create target database schema

In this next step you will use **SQL Server Management Studio** to create the target database schema (tables, keys and constrains). It should be noted that this is not a typical step in the data migration process, since usually the target schema already exists.

1. Keep **SQL Server Management Studio** open and connected to your SQL server instance.
2. Open the DDL Script provided with this exercise in Blackboard: CreateFudgeDriversSchema.sql and run it in it’s entirety to create the fd\_ tables. Refresh your table objects.
3. Create a database diagram by adding all the tables that begin with **fd\_** this should create a diagram like this:  
   
4. Save the diagram as **FudgeDrivers**.

Step three: Data cleansing

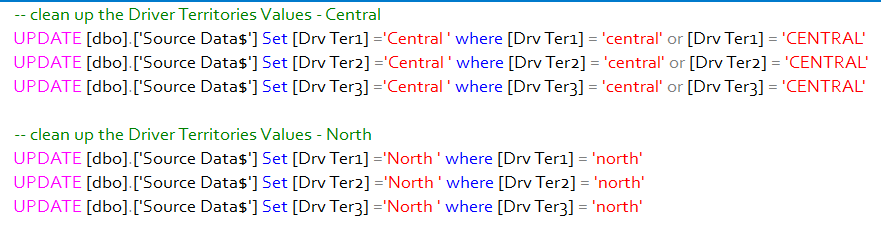
The next step in the migration process is data cleansing. In this step you “clean up” the source data so that things will be nice and neat when we do the data transformations. What do we need to clean up? Take a look at the following:



The data here represents the same thing, but it is represented differently ‘central vs. ‘CENTRAL’ and ‘north’ vs. ‘North’ this needs to be corrected before we transform the data.

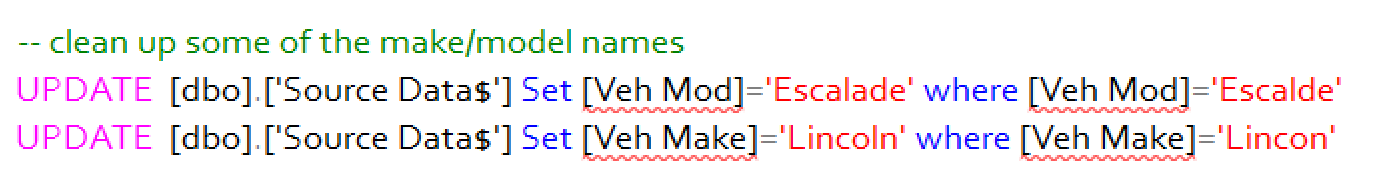
To clean the data up we run SQL update statements on the **source** data. **Open a new query window** in **SQL Management Studio** and start a migration script.

First type and run the following:



You can open your data source and verify that the fields were corrected.

I also notice that these users typed “Lincoln” and “Escalade” wrong so let’s fix that too. Using the model above, write the cleansing code to update the right fields with the right spelling.



Now that the data is clean, we’re ready to transform!

Step four: Transform the source data and load into the target tables

This next step is the critical step. Here’s where we move data from the source to the destination tables. In looking at our target schema, **we must start our import into the parent tables without foreign keys**. Therefore we first migrate data into the tables in the following order:

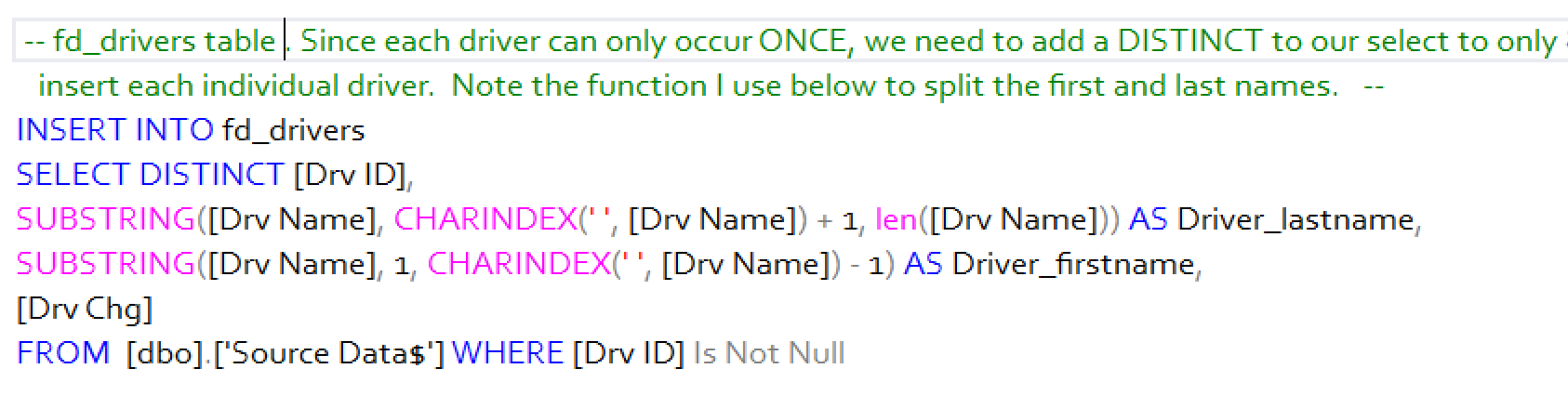
1. fd\_drivers *(no fk)*
2. fd\_driver\_territories *(fk looks up to drivers which you just inserted)*
3. fd\_sizes *(no fk)*
4. fd\_vehicles *(fk looks up to sizes which you just inserted)*
5. fd\_driver\_vehicles *(fks look up to tables whose data you just inserted)*

Regardless of the table we’re migrating, the approach is the same.

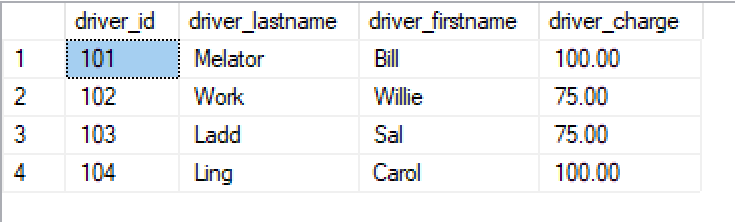
First, create the SQL SELECT statement(s) which output the data ***as you need it to appear in the target table***. Keep in mind these SQL techniques as you attempt to accomplish the task:

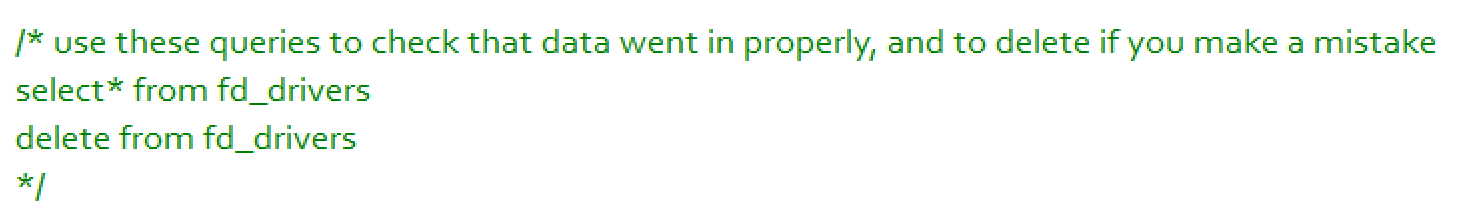
* The **DISTINCT** keyword outputs unique rows of data. This is important for building lookup tables, and tables on the “one” side of the 1-M relationship.
* The **UNION** keyword allows you to combine like SQL statements into one output. This is useful for addressing multi-valued attributes and repeating groups of data.
* No need to specify the target table’s column names here, just put the source fields in the same order as the target table fields and it will insert into the target table smoothly.
* To combine columns of data use the + operator.   
  eg.[Last Name] + ‘, ‘ + [First Name] as [Full Name]
* To extract one column into two columns use the CHARINDEX function.   
  eg. left([Full Name],charindex([Full Name],’,’)) as [Last Name]

First we load up the **fd\_drivers** table. The hardest part here is breaking up the first and last names. We gave you the function to do that here. Note that there should be a single space between those string quotes.

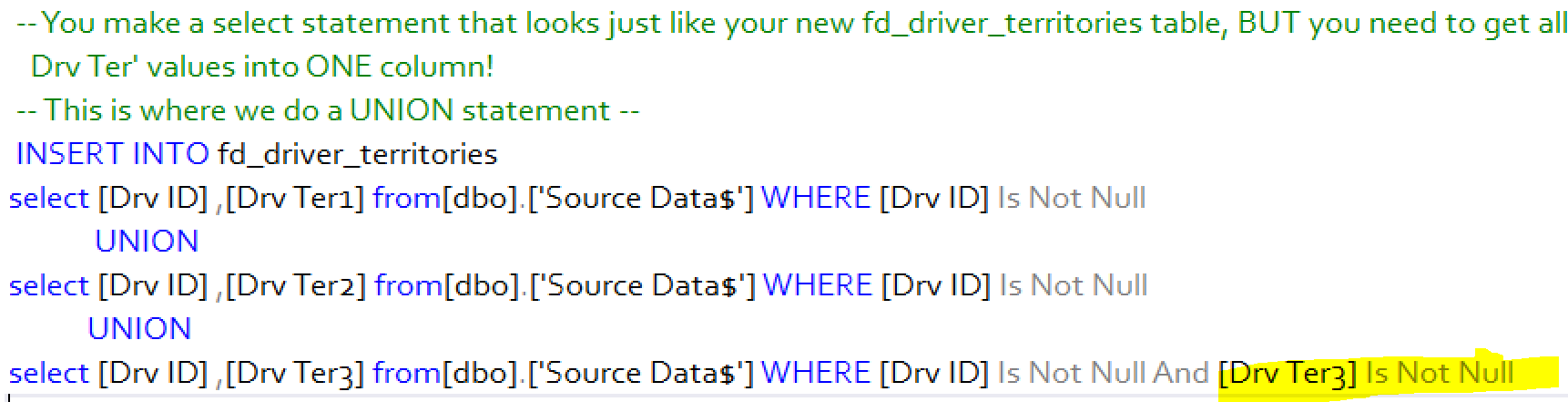


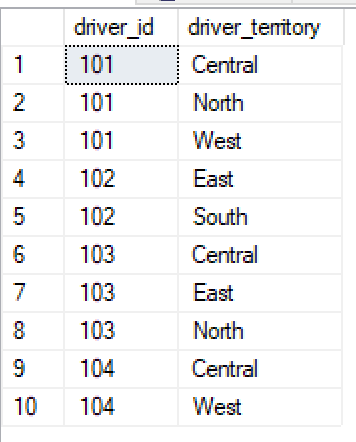
Then the table data should look like this.



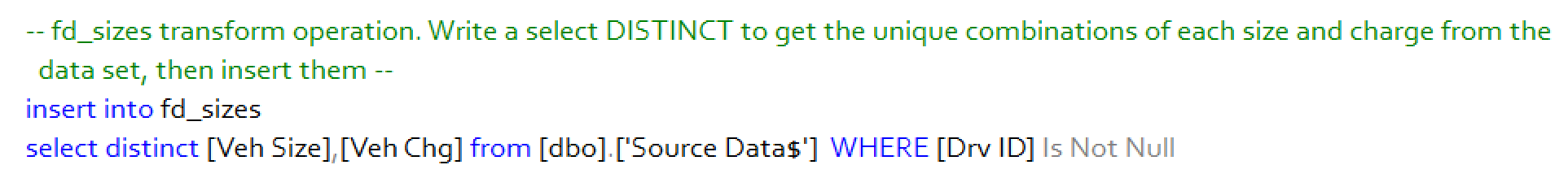


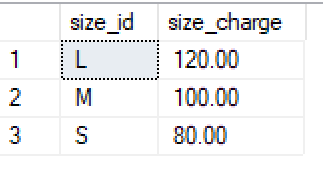
Next we load the **fd\_driver\_territories** table. Look at the portion in yellow. You can see that in the 3rd territory column we have some NULL values. Those can’t come in so we only want to insert values that are NOT null. I can exclude nulls from any column.

And the table data looks like:

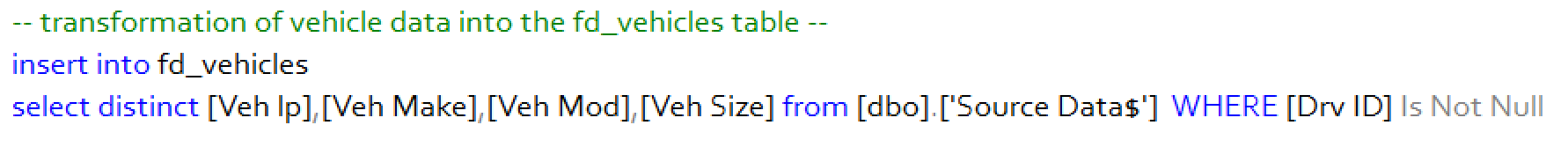


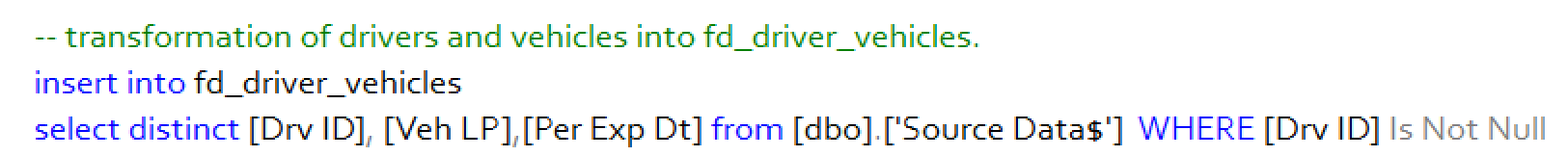
Now we can move onto our other ‘no fk’ table: fd\_sizes *(no fk)*

All I need to do is insert unique combinations of size/charge from my source data, as below:And the table data looks like:

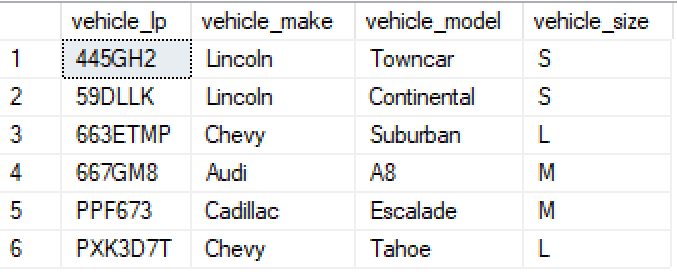


Next, finish up by transforming the source data into the **fd\_vehicles table, and the fd\_driver\_vehicles** table. Remember to look at your database diagram for help in ordering the columns!

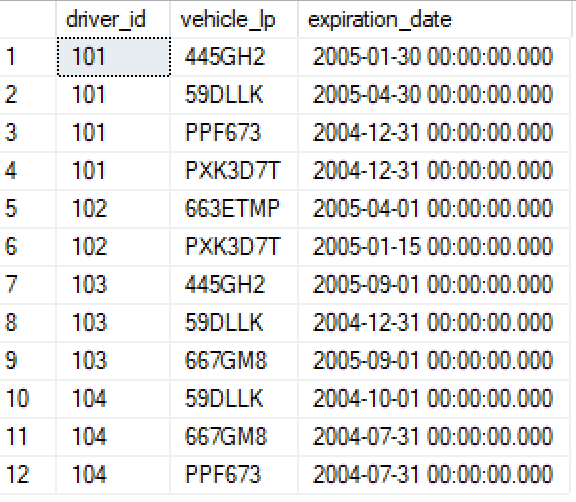




Data in the fd\_drivers table:



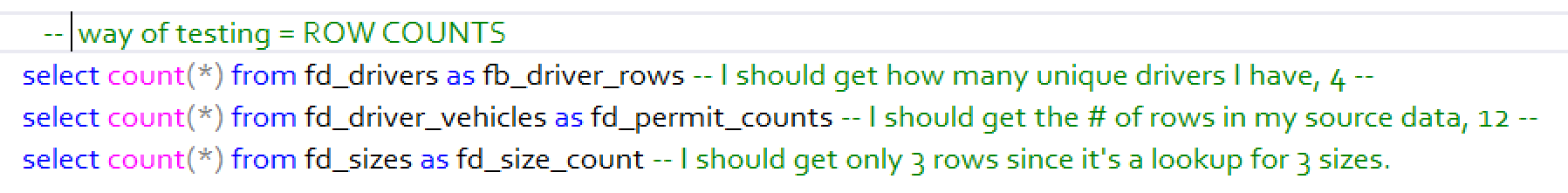
Data in the fd\_drivers\_vehicles table:



Step five: Data Verification

The final step is to perform data verification on the target data to make sure it does indeed have the same data as the source. The easiest (but farthest from complete) means of accomplishing this is to check row counts.

The following query compares the total row counts of the original data source and the corresponding table in the target data source.



Row counts work as long as you’re also checking to make sure the SQL commands are not generating errors. Data that errors out during transformation / loading well definitely affect row counts.

The final verification query joins all the tables back together to give you a view that resembles the original data.

