

Assignment 05:

Implementing Dimensional Models

Part 1: Overview

This assignment will introduce you to the process of data warehouse development, specifically DDS models from the Kimball technical architecture. You will implement the physical model (database tables, keys and constraints) for your **Details Dimensional-Modeling workbook**, and then test that model by performing an initial data load using SQL Queries, a fairly common practice in dimensional model development.

Goals

The goals of this assignment are to:

- Demonstrate general table, index, and key management in the SQL Query language, as a review of your SQL experience
- Teach you how to translate a detailed dimensional design specification into a ROLAP star-schema implementation, including table creation, primary and foreign keys, as well as populate your schema with sample data
- Reinforce that the nature of development is iterative, so you must be able to re-create your structures and repopulate them with data at will

Effort

This assignment can be done individually or with a partner. If you work with a partner, do not simply divide up the work. Collaborate with each other throughout the exercise as if you were working on the same data warehousing team.

Technical Requirements

To complete this assignment you will need the following:

- Access to the course **ist-cs-dw1.ad.syr.edu** SQL Server, and specifically the Northwind Traders database. You should connect to this server before starting the assignment.
- For **Part 2**: The completed dimensional modeling Excel workbook, titled **COMPLETED-Northwind-Detailed-Dimensional-Modeling-Workbook.xlsm**, available in the same place you got this lab.
- For **Part 3**: Your own completed Detailed Dimensional Modeling workbook from the previous assignment.
- Microsoft Excel 2007 or higher for viewing and editing the worksheets.

A Really Quick SQL Refresher Course

In this first part, we'll revisit some essential SQL commands you'll need to complete this assignment. Some of you may find this information redundant, so feel free to skip over this part if you feel your SQL ability is already on par with what this assignment requires. While you relearn SQL, you'll build a single script to create a DDS star schema.

SQL Is Required! SQL Is Not Required?

Many people argue that there's no need to learn how to create database objects with SQL since the tooling of the vendor's DBMS products is more than adequate for these types of tasks. My counterpoint to that argument is that using SQL gives you these advantages:

- 1) **A means to automate the process.** You can construct the entire dimensional model simply by executing a script.
- 2) **Replay ability.** You can quickly reproduce a dimensional model in your test, development, and production environments.
- 3) **Source code control.** SQL is code; code can be tracked using a software configuration management (SCM) tool like Git, Subversion, or SVN.

Considering the advantages above, coupled with how easy it is to learn SQL and to create objects in it, I strongly recommend using SQL for database design projects. I will **require** its use for this course!

SQL Data Definition Language 101

At the heart of the SQL language are the data definition language (DDL) commands. These commands allow you to create, edit, and delete tables and their complementary structures:

DDL Command	Purpose	Example
CREATE	Creates a new database object	CREATE VIEW vw_demo ...
ALTER	Changes an existing database object	ALTER TABLE demo ...
DROP	Deletes an existing database object	DROP INDEX ix_demo ...

In addition to these commands are the types of **database objects** you can manipulate with them. Here are some of the objects we'll use in this class:

Database Object	Purpose
TABLE	Data storage mechanism consisting of rows and columns
SYNONYM	An alias for an existing table
INDEX	A structure to improve the retrieval of data from a table
VIEW	A named representation for a SQL SELECT Query

So you can combine any combination of DDL command + database object to produce the appropriate command you need. For example: CREATE VIEW, DROP INDEX, ALTER TABLE, etc. At this point all that's left are the details. ☺

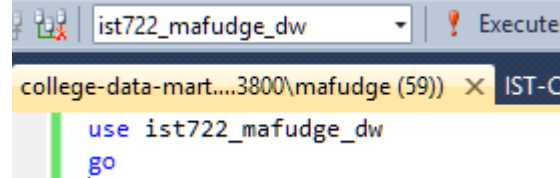
Create Table

To make a new table, use the CREATE TABLE statement. The syntax is:

```
CREATE TABLE tableName (  
    column1 datatype null | not null,  
    column2 datatype null | not null,  
    ...  
    columnN datatype null | not null,  
    CONSTRAINT pkTableNameColumn PRIMARY KEY (column1)  
);
```

Okay, let's create some tables that might be used in a simple data mart.

DO THIS: From your **ist722_yournetid_dw** database, open a new query window (Press **Ctrl+N**) and type in the SQL as it appears in the screenshot (using your NetID, of course). →

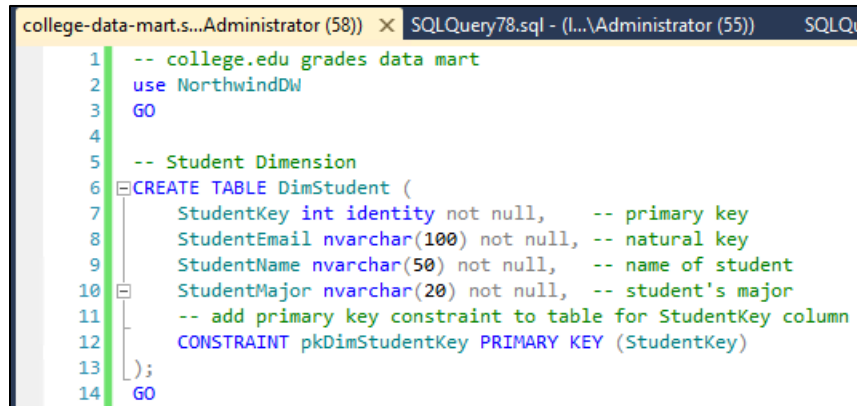


Then press **Ctrl+S** to save your SQL script as **college-data-mart.sql**. When you're done, it should have the name of the SQL code file in your tab (see screen shot above).

What do these commands do? The first line in the script is which database to use. The "GO" command tells SQL Server to batch everything before this statement, guaranteeing it is executed before advancing to the next command in the script.

Next, let's create the **DimStudents** table:

DO THIS: Type in the following code, starting with line 5:

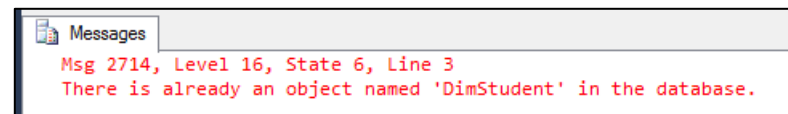


This table, called **DimStudent** has four columns and one constraint. The columns and their data types are listed first, separated by a comma. None of the columns permit nulls. In line 12 of the statement is a primary key constraint rule over the **StudentKey** column. The type **int identity** in line 7 generates a surrogate key.

When you're ready to execute your script, press the **[F5]** key. This will run your script and create your table. If it works, you should see Command(s) completed successfully. in the **Messages** area below.

If you have an error, see if you can troubleshoot the issue by comparing the screenshot to your code on the line number in question.

If you got it to work, press **[F5]** to execute your code again; you'll notice that this time you will get an error:



This error means exactly what it says. You've created the **DimStudent** table already, and you cannot create it again.

If you need to re-create it, you'll have to **execute a command to get rid of it first**.

Drop Table

The DROP TABLE command is used to remove a table (and all of the data therein) from the database. DROP TABLE should be used with caution, and in general is only useful when building out a database design.

Let's modify our SQL script so that we can reexecute it by adding a DROP TABLE before the CREATE TABLE.

DO THIS: Inside your SQL script, insert the code as it appears on lines 4-7. Your updated code will now first drop the table **DimStudent** as a command batch and then create the table **DimStudent** in the subsequent batch.

Press **[F5]** to execute your query. It should run without error each time!

```
1  -- college.edu grades data mart
2  use NorthwindDW
3  GO
4
5  DROP TABLE DimStudent;
6  GO
7
8  -- Student Dimension
9  CREATE TABLE DimStudent (
10     StudentKey int identity not null,
```

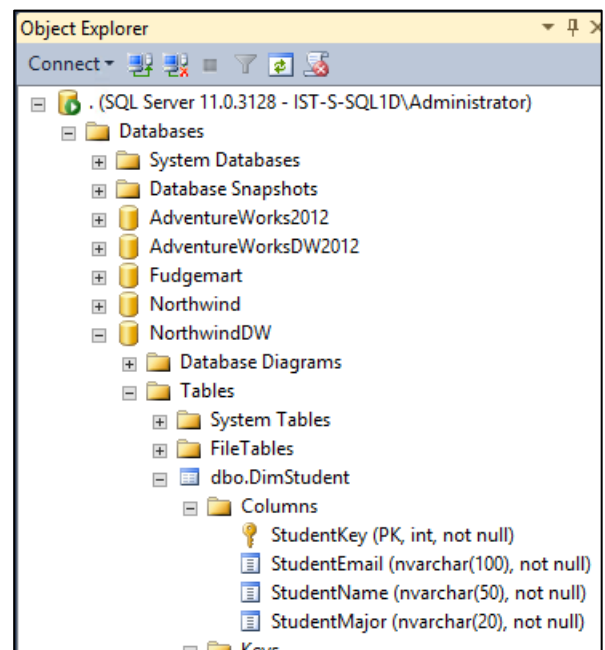
NOTE: If your DimStudent has a red line under it, it's probably because you didn't create the table, or you may simply need to refresh your Intellisense cache. Try pressing **Ctrl+Shift+R** to see if that corrects the problem.

Where Is My Table?

The really cool thing about SQL script is that it creates real database objects that you can view through the GUI of the database management system. Let's see if we can find and open our **DimStudent** table.

DO THIS: In **Object Explorer**, **double-click** on the **ist722_yournetid_dw** database, and then **double-click** on the **Tables** folder. You should see **dbo.DimStudent**. If you do not, from the **Menu** choose **View → Refresh** to refresh your view.

If you **double-click** on the **Columns** folder, you should see the columns and data types in our table.



The Rest

Here's the entire script with two dimension tables and one fact table. You will notice that I create my fact table last, but my DROP TABLE for my fact table comes first. This is because of the fact table's foreign key dependencies. To preserve referential integrity, the DBMS prevents you from dropping any table that is referenced by a foreign key (and believe me, this is a good thing). The fact table is full of foreign keys, and so it must be dropped before the dimension tables.

DO THIS: Complete this script you see to your right, → → → → → and then press [F5].

NOTE: If you encounter errors, match up the line number of the error in your code to the screen shot on your right to see if you can reconcile the error.

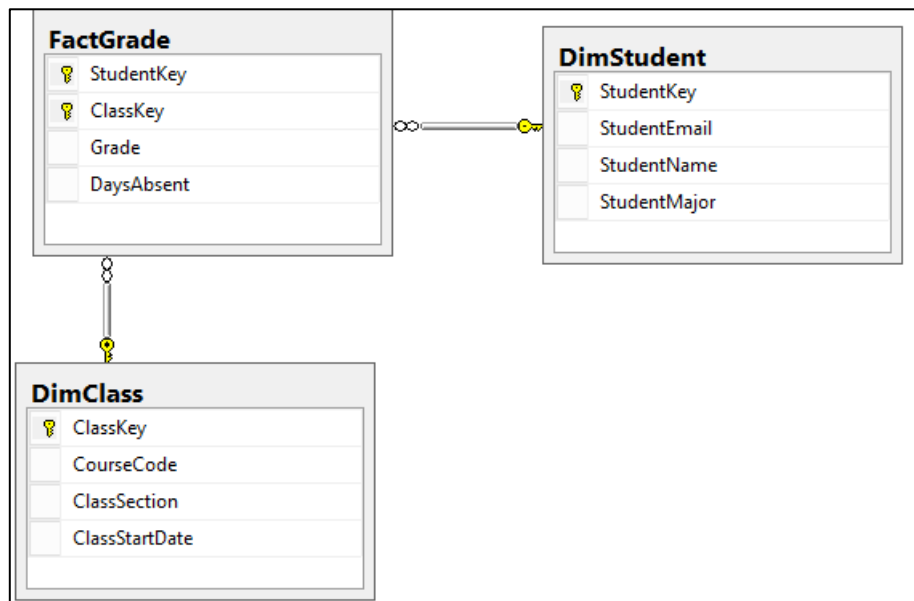
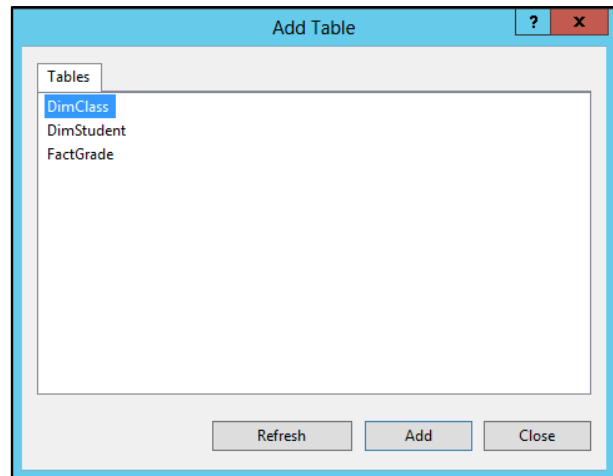
```
college-data-mart.s...Administrator (58)) X SQLQuery78.sql - (L...Administrator (55)) SQLQuery77
1  -- college.edu grades data mart
2
3
4
5  DROP TABLE FactGrade;
6  DROP TABLE DimClass;
7  DROP TABLE DimStudent;
8  GO
9
10 -- Student Dimension
11 CREATE TABLE DimStudent (
12     StudentKey int identity not null, -- primary key
13     StudentEmail nvarchar(100) not null, -- natural key
14     StudentName nvarchar(50) not null, -- name of student
15     StudentMajor nvarchar(20) not null, -- student's major
16     -- add primary key constraint to table for StudentKey column
17     CONSTRAINT pkDimStudentKey PRIMARY KEY (StudentKey)
18 );
19 GO
20
21 -- Class Dimension
22 CREATE TABLE DimClass (
23     ClassKey int identity not null, -- primary key
24     CourseCode nvarchar(6) not null, -- course code IST722
25     ClassSection nvarchar(4) not null, -- section M001
26     ClassStartDate datetime not null, -- class start date
27     CONSTRAINT pkDimClassKey PRIMARY KEY (ClassKey)
28 );
29 GO
30
31 -- Grade Fact
32 CREATE TABLE FactGrade
33 (
34     StudentKey int not null, -- StudentKey
35     ClassKey int not null, -- ClassKey
36     Grade decimal(2,1) not null, -- Student grade ex. 3.5
37     DaysAbsent int default(0) not null, -- days absent w/default=0
38     -- both StudentKey + ClassKey are this table's PK
39     CONSTRAINT pkFactGradeKey PRIMARY KEY (StudentKey, ClassKey),
40     -- add foreign key constraint for StudentKey
41     CONSTRAINT fkFactGradeStudentKey FOREIGN KEY (StudentKey)
42         REFERENCES DimStudent(StudentKey),
43     -- add foreign key constraint for StudentKey
44     CONSTRAINT fkFactGradesClassKey FOREIGN KEY (ClassKey)
45         REFERENCES DimClass (ClassKey)
46 );
```

Building a Database Diagram

They say a picture is worth 1,000 words, and frankly, I could not agree more. One of the most useful things you can do upon completion of your data mart is create a **database diagram**. The database diagram shows you the table definitions, column definitions, and foreign key relationships, giving you a clear picture of what is in your table and how they are connected to one another. Building a database diagram is easy.

DO THIS: Under **Object Explorer** for the **ist722_yournetid_dw** database, **right-click** on **Database Diagrams**, and then choose **New Database Diagram** from the context menu. (If you are asked to install database support, select **Yes**). Next you will see a dialog prompting you to add tables to your diagram (on your right). →

Click on each **table**, then click **Add**. After you've added all three tables, click **Close**. You should see a picture of your dimensional model on the screen:



Close your diagram window, and when it asks you to save, click **Yes**.

Save the diagram as **CollegeDataMart** and click **OK**.

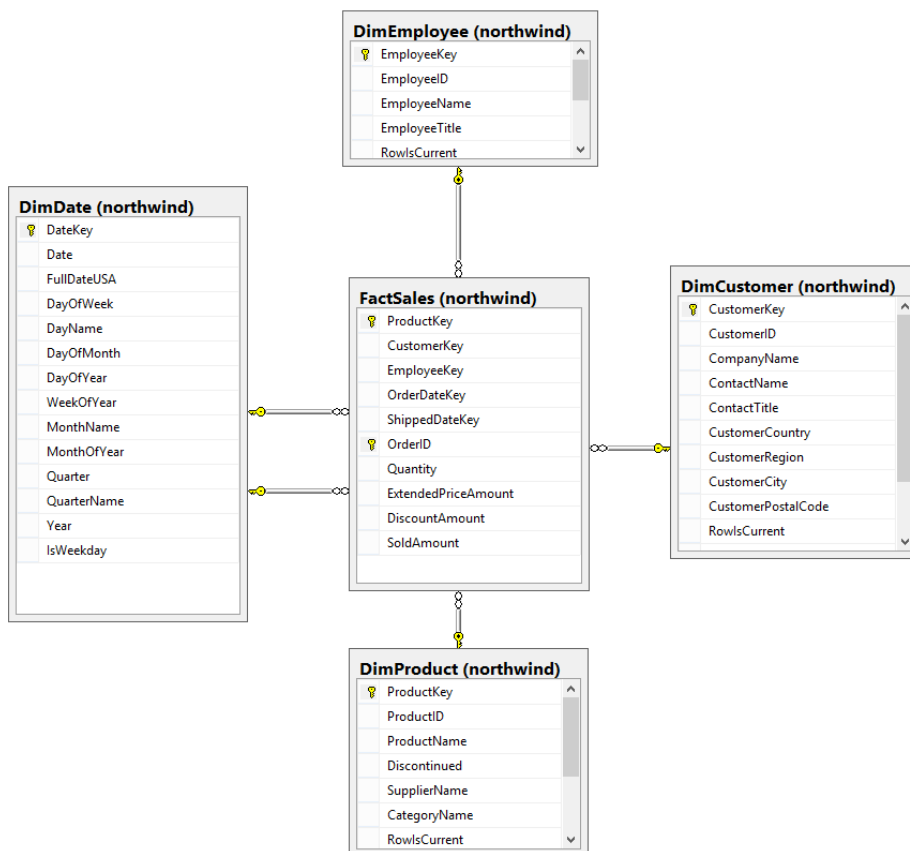
NOTE: You can view or edit this database diagram anytime by double-clicking on it in the Database Diagrams folder.

Part 2: Walk-Through

In this part we will implement, populate, and test a DDS star schema for the Northwind Sales Reporting business process. Here's the excerpt from the **Bus Matrix**:

Business Process Name	Fact Table	Fact Grain Type	Granularity	Facts	Product	Customer	Employee	Order Date	Shipped Date
Sales Reporting	FactSales	Transaction	one row per order detail / line item.	Quantity, Unit Price , Discount Amount, Sold Amount, Freight Amount	X	X	X	X	X






Our plan is to turn the business process you see above into the following dimensional data store (DDS) star schema:



We will accomplish this by generating SQL code from the **Northwind Detailed Dimensional-Modeling workbook**, which contains the detailed specification for the star schema. On to the steps!

Part 2a: Create the Star Schema

We'll start by implementing the star schema in SQL. Your goal is to create one SQL script, **part2a-northwind-sales-star-schema.sql**, which, when executed, will create the following schema and tables:

-  northwind.DimCustomer
-  northwind.DimDate
-  northwind.DimEmployee
-  northwind.DimProduct
-  northwind.FactSales

Your steps, at a high level:

- 1) Create a new SQL Query.
- 2) Switch to your data warehouse (dw) database: (your name will vary, of course)
`1 use ist722_mafudge_dw`
- 3) Open the completed dimensional modeling Excel workbook titled **COMPLETED-Northwind-Detailed-Dimensional-Modeling-Workbook-KimballU.xlsm** and use it to start making your SQL. You can hand code the SQL from the design or try to use the macro to get started.
- 4) There should be a DROP TABLE commands before the CREATE TABLE commands so that it can be rerun without error. This is critical to the iterative nature of development, as you might need to rerun this script after changes.
- 5) Remember to drop the fact table first but create it last to take care of the foreign key dependencies. Either do that, or drop the FKs.
- 6) Be sure to add the tables to a **fudgemart** schema to avoid conflicts with other project tables.
- 7) Remember to execute your script against your **dw** (data warehouse) database!
- 8) When you're done, create a **Database Diagram**. It should be identical to the screenshot on the previous page!

Part 2b: Initial Stage of Data

Now that the ROLAP start schema has been created, it's time to perform the initial ETL load from the source system. The goal of this process does not replace actual ETL tooling since we have no means to automate, audit success or failure, track changes to data, or document this process. Instead our goals are simply to:

1. Understand how to source the data required for our implementation,
2. Verify that our model actually solves our business problem,
3. Remove our dependency on the external world by staging our data, and
4. Complete these tasks in a manner in which we can re-create our data, when required.

Let's start staging the source data. Open a query window, and save the query as **part2b-northwind-sales-data-stage.sql**. Switch to your stage database (name will vary from the screenshot, of course):

```
use [ist722_mafudge_stage]
```

Staging Northwind Customers

Here's the basic structure of the command to stage data from source. This query stages Customers from Northwind.

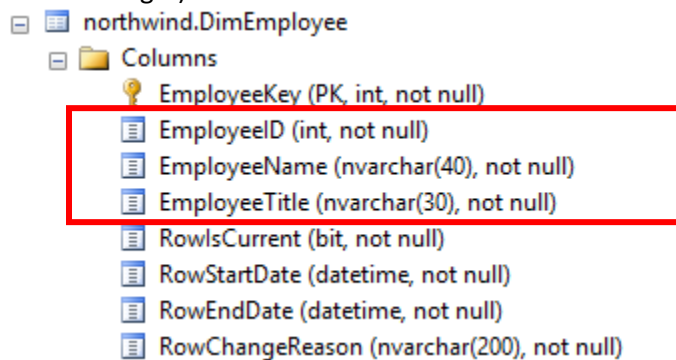

```
1  -- stage Customers
2  SELECT [CustomerID]
3         , [CompanyName]
4         , [ContactName]
5         , [ContactTitle]
6         , [Address]
7         , [City]
8         , [Region]
9         , [PostalCode]
10        , [Country]
11  INTO [dbo].[stgNorthwindCustomers]
12  FROM [Northwind].[dbo].[Customers]
```

Type it in and execute it to create the stage table, and then populate it with data from the source. We want to save all the stage queries into this one file.

Staging Northwind Employees

This time let's focus on the process.

1. What data do we need? For the answer to this question, consult the **DimEmployee** table (the eventual target)



It looks like we need EmployeeID, EmployeeName, and EmployeeTitle. When in doubt, refer to the detailed design worksheet where you specified the source to target map.

2. Next write an SQL Select statement to acquire the data. Execute this:

```
1  SELECT [EmployeeID]
2         , [FirstName]
3         , [LastName]
4         , [Title]
5  FROM [Northwind].[dbo].[Employees]
```

Take a look at the output and make sure it's the data you need.

	EmployeeID	FirstName	LastName	Title
1	1	Nancy	Davolio	Sales Representative
2	2	Andrew	Fuller	Vice President, Sales
3	3	Janet	Leverling	Sales Representative
4	4	Margaret	Peacock	Sales Representative
5	5	Steven	Buchanan	Sales Manager
6	6	Michael	Suyama	Sales Representative
7	7	Robert	King	Sales Representative
8	8	Laura	Callahan	Inside Sales Coordinator
9	9	Anne	Dodsworth	Sales Representative

NOTE: You might be tempted to combine first name and last name, as required by the target.

DO NOT DO THIS. Always stage data exactly as it appears from the source. Our goal is to have an exact version of the source pipeline without being dependent on the availability of the actual source. This allows us to design and implement the transformation logic over several iterations (which you will probably need) without taxing the source.

- Finally, when the data is what you need, it's time to sock it away into a stage table, adding it to the stage script including the **INTO** clause:

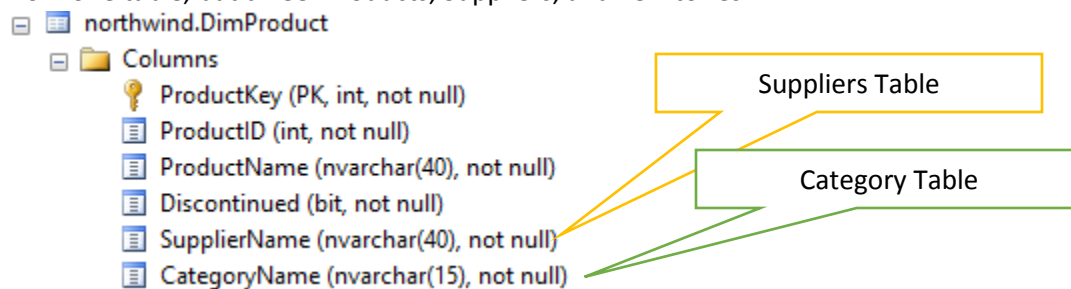
```
14  -- stage Employees
15  SELECT [EmployeeID]
16         , [FirstName]
17         , [LastName]
18         , [Title]
19  INTO [dbo].[stgNorthwindEmployees]
20  FROM [Northwind].[dbo].[Employees]
```

NOTE: The INTO clause of the SELECT statement creates the table and populates it with data. If for some reason you “mess this up,” you will need to drop the table before you can execute this statement again.

Staging Northwind Products

This next example is part of the *great data staging debate*. What does that mean? Let's find out:

If we take a peek at the Product dimension, you'll see that the source of this dimension does not come from one table, but three: Products, Suppliers, and Territories.



Should we stage all three tables? Or stage the query output of the join? The answer is, “It depends.”

- Will Supplier or Category be used as a dimension in another dimensional model? If so, stage the tables independently.

- It will accommodate future scenarios to stage independently.

It is more convenient to stage the query output, and as this is just an academic exercise, we will go that route:

```
1  SELECT [ProductID]
2         , [ProductName]
3         , [Discontinued]
4         , [CompanyName]
5         , [CategoryName]
6  FROM [Northwind].[dbo].[Products] p
7       join [Northwind].[dbo].[Suppliers] s
8         on p.[SupplierID] = s.[SupplierID]
9       join [Northwind].[dbo].[Categories] c
10      on c.[CategoryID] = p.[CategoryID]
```

After you execute the query, add the INTO clause to stage the data, adding to our [part2b-northwind-sales-data-stage.sql](#) script:

```
22  -- stage Products
23  SELECT [ProductID]
24         , [ProductName]
25         , [Discontinued]
26         , [CompanyName]
27         , [CategoryName]
28  INTO [dbo].[stgNorthwindProducts]
29  FROM [Northwind].[dbo].[Products] p
30       join [Northwind].[dbo].[Suppliers] s
31         on p.[SupplierID] = s.[SupplierID]
32       join [Northwind].[dbo].[Categories] c
33      on c.[CategoryID] = p.[CategoryID]
```

Staging the Date Dimension

The date and time dimensions are the ultimate conformed dimensions. They are reused everywhere in the data warehouse. There should be no reason to stage a date or time dimension as your DW implementation will already have this dimension present and populated with data.

While we could stage the entire date dimension, we'll use this example to demonstrate how to only stage the data we need.

NOTE: The following is an academic exercise. Normally you would not stage a date dimension, let alone in this manner.

How many dates do we need? To answer this question, let's query the Orders table for the min and max Order and Ship dates:

```
2 SELECT min(OrderDate)
3         , max(OrderDate)
4         , min(ShippedDate)
5         , max(ShippedDate)
6 FROM [Northwind].[dbo].[Orders]
```

	(No column name)	(No column name)	(No column name)	(No column name)
1	1996-07-04 00:00:00.000	1998-05-06 00:00:00.000	1996-07-10 00:00:00.000	1998-05-06 00:00:00.000

It looks like we're in good shape grabbing the years 1996 through 1998.

Here's the SQL you should add to the script to stage the dates we require:

```
41 -- stage date
42 SELECT *
43 INTO [dbo].[stgNorthwindDates]
44 FROM [ExternalSources].[dbo].[v_date_dimension]
45 WHERE Year between 1996 and 1998
```

Staging the Fact Table

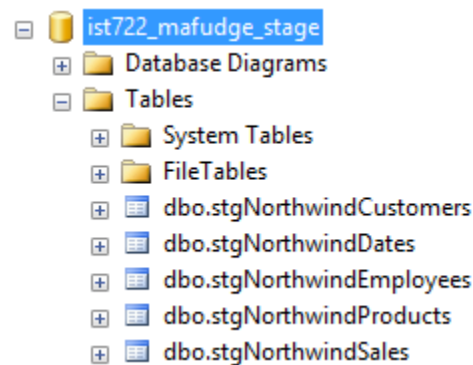
And finally, we stage the fact table:

northwind.FactSales
Columns
ProductKey (PK, FK, int, not null)
CustomerKey (FK, int, not null)
EmployeeKey (FK, int, not null)
OrderDateKey (FK, int, not null)
ShippedDateKey (FK, int, not null)
OrderID (PK, int, not null)
Quantity (smallint, not null)
UnitPrice (money, not null)
UnitDiscountAmount (money, not null)
SoldAmount (money, not null)

But, the dimension keys like **CustomerKey** and **ProductKey** are not in the staged data! What do we do? Instead, we use the natural keys from the data sources so that later in the pipeline we can “lookup” the dimension keys. This is called the *surrogate key pipeline*, and it is fairly common when loading a fact table. Here's the SQL to complete the stage:

```
48  --stage fact
49  SELECT [ProductID]
50         ,d.[OrderID]
51         , [CustomerID]
52         , [EmployeeID]
53         , [OrderDate]
54         , [ShippedDate]
55         , [UnitPrice]
56         , [Quantity]
57         , [Discount]
58  INTO [dbo].[stgNorthwindSales]
59  FROM [Northwind].[dbo].[Order Details] d
60       join [Northwind].[dbo].[Orders] o
61       on o.[OrderID] = d.[OrderID]
```

At this point, you should have 5 stage tables:



...and **one script** that will stage the data on demand by re-creating the tables. Now that the extraction is done, it's time to write a script to load staged data into our star schema!

Part 2c: Load From Stage Into the Data Warehouse

In this next step we will load from the stage database into our data warehouse (dw database), bringing our star schema to life with actual data!

Important Tip: Document your findings in this step, and this will help you plan out the actual ETL process later on. For example, if you need to combine two columns into one or replace NULL with a value, then this will need to happen in your ETL tooling later on.

Since the dimensions depend on the fact table, we should start with those first.

First open a new query and save it as **part2c-northwind-sales-data-load.sql**. Execute this command to switch to your data warehouse database (again, this will be different based on your user id).

```
1  use ist722_mafudge_dw
```

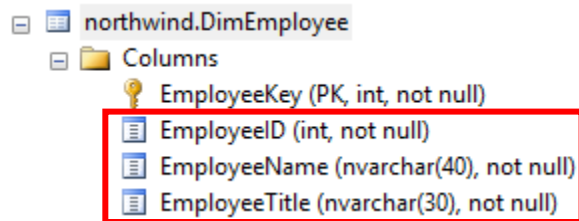
Loading DimEmployee

Since the dimension table exists already and we do not want to replace them, we cannot use the SELECT .. INTO statement. Instead we will use the INSERT INTO ... SELECT pattern to add data to the dimension.

Here's the process:

1. Identify the requirements of the dimension.
2. Write a select statement using the staged data to source the dimension.
3. Create an INSERT INTO ... SELECT statement to populate the data.

For example, DimEmployee requires:



And the stage table has:

```
17 SELECT [EmployeeID], [Firstname], [LastName], [Title]
18 FROM [ist722_mafudge_stage].[dbo].[stgNorthwindEmployees]
```

100 %

Results Messages

	EmployeeID	Firstname	LastName	Title
1	1	Nancy	Davolio	Sales Representative
2	2	Andrew	Fuller	Vice President, Sales
3	3	Janet	Leverling	Sales Representative
4	4	Margaret	Peacock	Sales Representative
5	5	Steven	Buchanan	Sales Manager
6	6	Michael	Suyama	Sales Representative
7	7	Robert	King	Sales Representative
8	8	Laura	Callahan	Inside Sales Coordinator
9	9	Anne	Dodsworth	Sales Representative

We need to combine the name into one column, like this:

```
17 SELECT EmployeeID, Firstname + ' ' + LastName as EmployeeName, [Title]
18 FROM [ist722_mafudge_stage].[dbo].[stgNorthwindEmployees]
```

100 %

Results Messages

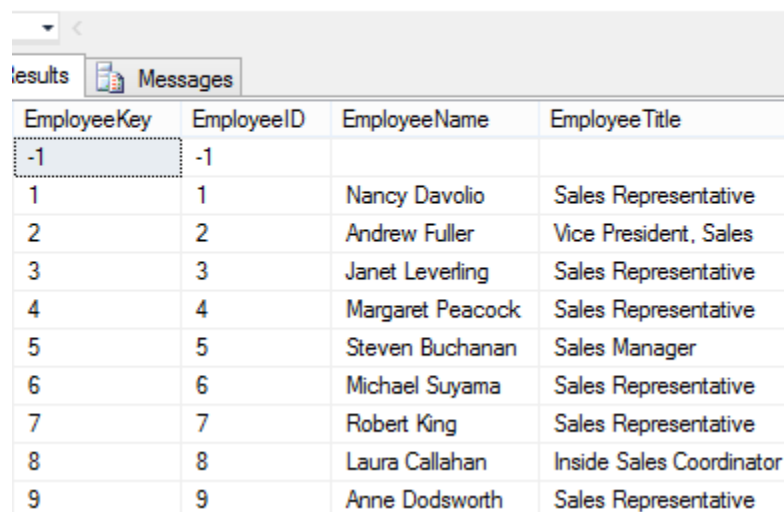
	EmployeeID	EmployeeName	Title
1	1	Nancy Davolio	Sales Representative
2	2	Andrew Fuller	Vice President, Sales
3	3	Janet Leverling	Sales Representative
4	4	Margaret Peacock	Sales Representative
5	5	Steven Buchanan	Sales Manager
6	6	Michael Suyama	Sales Representative
7	7	Robert King	Sales Representative
8	8	Laura Callahan	Inside Sales Coordinator
9	9	Anne Dodsworth	Sales Representative

That looks good. Now that the data shape of our source matches the target dimension, we can load it with this statement:

```
1  -- load DimEmployee
2  INSERT INTO [northwind].[DimEmployee]
3      ([EmployeeID] ,[EmployeeName] ,[EmployeeTitle])
4  SELECT EmployeeID, Firstname + ' ' + LastName as EmployeeName, [Title]
5  FROM [ist722_mafudge_stage].[dbo].[stgNorthwindEmployees]
```

Then check to see that the data was inserted:

```
17 SELECT * FROM [northwind].[DimEmployee]
```



EmployeeKey	EmployeeID	EmployeeName	EmployeeTitle
-1	-1		
1	1	Nancy Davolio	Sales Representative
2	2	Andrew Fuller	Vice President, Sales
3	3	Janet Leverling	Sales Representative
4	4	Margaret Peacock	Sales Representative
5	5	Steven Buchanan	Sales Manager
6	6	Michael Suyama	Sales Representative
7	7	Robert King	Sales Representative
8	8	Laura Callahan	Inside Sales Coordinator
9	9	Anne Dodsworth	Sales Representative

NOTE: It's mere coincidence that the values of **EmployeeKey** and **EmployeeID** match up. Do not assume this will always be the case!

Loading DimCustomer

You probably have gotten this figured out by now. Here's the SQL to load the Customer Dimension:

```
8  -- load DimCustomer
9  INSERT INTO [northwind].[DimCustomer]
10     ([CustomerID],[CompanyName],[ContactName],[ContactTitle],
11     [CustomerCountry],[CustomerRegion],[CustomerCity],[CustomerPostalCode])
12  SELECT
13     [CustomerID], [CompanyName], [ContactName], [ContactTitle],
14     [Country], [Region], [City], [PostalCode]
15  FROM [ist722_mafudge_stage].[dbo].[stgNorthwindCustomers]
```

But when you execute this, you get an error:

```
Cannot insert the value NULL into column 'CustomerRegion', table
'ist722_mafudge_dw.northwind.DimCustomer'; column does not allow nulls. INSERT fails.
```

This is a fairly common error. The source permits null, but the dimension, as a best practice, does not. As such, we need to replace NULL with a default value, such as 'N/A'. To accomplish this, I use the CASE WHEN expression. Here's the updated code, fixing this problem in CustomerRegion and CustomerPostalCode:

```

8  -- load DimCustomer
9  INSERT INTO [northwind].[DimCustomer]
10 ([CustomerID],[CompanyName],[ContactName],[ContactTitle], [CustomerCountry],
11 [CustomerRegion],[CustomerCity],[CustomerPostalCode])
12 SELECT
13 [CustomerID], [CompanyName], [ContactName], [ContactTitle], [Country],
14 case when [Region] is null then 'N/A' else [Region] end,
15 [City],
16 case when [PostalCode] is null then 'N/A' else [PostalCode] end
17 FROM [ist722_mafudge_stage].[dbo].[stgNorthwindCustomers]

```

Loading DimProduct

Try to load the product dimension on your own. Here's your approach:

1. Map each column in the dimension table DimProduct to a column in stgNorthwindProducts.
2. You'll have to use CASE WHEN to change the bit field for Discontinued into a Y / N field in the dimension.
3. If you don't know what the dimension columns mean, consult the Excel worksheet.

Loading DimDate

Next load the Date Dimension on your own, again mapping the columns from source to target, and consult the Excel worksheet for the meaning of the columns. This step should be trivial.

Loading the Fact Table

Finally, on to our biggest challenge yet – loading the fact table.

The Surrogate Key Pipeline

There's a unique challenge to building the fact table since the source data is missing the dimension foreign keys. Observe:

Stage	stgNorthwindSales	Fact	FactSales	Look up	
Table	ProductID	Table	ProductKey	The Key	DimProduct
Has	CustomerID	Needs	CustomerKey	value	DimCustomer
business	OrderDate	This	OrderDateKey	using this	DimDate
key:	ShippedDate	Foreign	ShippedDateKey	table:	DimDate
	EmployeeID	Key:	EmployeeKey		DimEmployee

What we need to do is build what is known as a surrogate key pipeline:

For each foreign key in the fact table, match the source data business key to the dimension **business key** so we can look up the dimension primary key.

For example, here's the setup for customer ID:

```

SELECT s.*, c.CustomerKey
FROM [ist722_mafudge_stage].[dbo].[stgNorthwindSales] s
join [ist722_mafudge_dw].[northwind].[DimCustomer] c
on s.CustomerID = c.CustomerID -- match on business key, not pk/fk!

```

The output:

ProductID	OrderID	CustomerID	EmployeeID	OrderDate	ShippedDate	UnitPrice	Quantity	Discount	CustomerKey
11	10248	VINET	5	1996-07-04 00:00:00.000	1996-07-16 00:00:00.000	14.00	12	0	85
42	10248	VINET	5	1996-07-04 00:00:00.000	1996-07-16 00:00:00.000	9.80	10	0	85
72	10248	VINET	5	1996-07-04 00:00:00.000	1996-07-16 00:00:00.000	34.80	5	0	85
14	10249	TOMSP	4	1996-07-10 00:00:00.000	1996-07-10 00:00:00.000	0.00	0	0	79
51	10249	TOMSP	4	1996-07-10 00:00:00.000	1996-07-10 00:00:00.000	0.00	0	0	79
41	10250	HANAR	4	1996-07-08 00:00:00.000	1996-07-12 00:00:00.000	7.70	10	0	34
51	10250	HANAR	4	1996-07-08 00:00:00.000	1996-07-12 00:00:00.000	42.40	35	0.15	34
65	10250	HANAR	4	1996-07-08 00:00:00.000	1996-07-12 00:00:00.000	16.80	15	0.15	34
22	10251	VICTE	3	1996-07-08 00:00:00.000	1996-07-15 00:00:00.000	16.80	6	0.05	84
57	10251	VICTE	3	1996-07-08 00:00:00.000	1996-07-15 00:00:00.000	15.60	15	0.05	84
65	10251	VICTE	3	1996-07-08 00:00:00.000	1996-07-15 00:00:00.000	16.80	20	0	84

If we repeat this process for the other dimensions of product and employee, we should get this:

```
SELECT s.*, c.CustomerKey, e.EmployeeKey, p.ProductKey
FROM [ist722_mafudge_stage].[dbo].[stgNorthwindSales] s
join [ist722_mafudge_dw].[northwind].DimCustomer c
    on s.CustomerID = c.CustomerID -- match on business key, not pk/fk!
join [ist722_mafudge_dw].[northwind].DimEmployee e
    on s.EmployeeID = e.EmployeeID -- match on business key, not pk/fk!
join [ist722_mafudge_dw].[northwind].DimProduct p
    on s.ProductID = p.ProductID -- match on business key, not pk/fk!
```

That gives us everything but the date dimension keys. This can be handled differently because date keys are predictable. For example, the date key for Aug-12-2015 would be 20150812. You can generate a date key with some simple formatting to YYYYMMDD. Fortunately, I've written a SQL function for you to do this. It is accessible from the **ExternalSources** database.

Here's the code completing the surrogate key pipeline, generating the Order and Shipping date keys:

```
SELECT s.*, c.CustomerKey, e.EmployeeKey, p.ProductKey,
    [ExternalSources].[dbo].[getDateKey](s.OrderDate) as OrderDateKey,
    [ExternalSources].[dbo].[getDateKey](s.ShippedDate) as ShippedDateKey
FROM [ist722_mafudge_stage].[dbo].[stgNorthwindSales] s
join [ist722_mafudge_dw].[northwind].DimCustomer c
    on s.CustomerID = c.CustomerID -- match on business key, not pk/fk!
join [ist722_mafudge_dw].[northwind].DimEmployee e
    on s.EmployeeID = e.EmployeeID -- match on business key, not pk/fk!
join [ist722_mafudge_dw].[northwind].DimProduct p
    on s.ProductID = p.ProductID -- match on business key, not pk/fk!
```

Writing Out the Facts

Now that we've addressed the foreign keys, it's time to write out the facts we require in the fact table. Most of our facts are derived from unit price, quantity, and discount rate. However, we should store the results of the calculations in our fact table, not the source values. This is contradictory to what you learned in OLTP databases, but it makes sense in the data warehouse because our data is static.

Fact	Source
Quantity	Quantity
ExtendedPriceAmount	Quantity * UnitPrice
DiscountAmount	Quantity * UnitPrice * Discount
SoldAmount	Quantity * UnitPrice * (1 - Discount)

Here's the query with just the columns we need and the required facts:

```
SELECT p.ProductKey, c.CustomerKey, e.EmployeeKey,
       [ExternalSources].[dbo].[getDateKey](s.OrderDate) as OrderDateKey,
       [ExternalSources].[dbo].[getDateKey](s.ShippedDate) as ShippedDateKey,
       s.OrderID,
       Quantity,
       Quantity * UnitPrice as ExtendedPriceAmount,
       Quantity * UnitPrice * Discount as DiscountAmount,
       Quantity * UnitPrice * (1-Discount) as SoldAmount
FROM [ist722_mafudge_stage].[dbo].[stgNorthwindSales] s
join [ist722_mafudge_dw].[northwind].DimCustomer c
    on s.CustomerID = c.CustomerID
join [ist722_mafudge_dw].[northwind].DimEmployee e
    on s.EmployeeID = e.EmployeeID
join [ist722_mafudge_dw].[northwind].DimProduct p
    on s.ProductID = p.ProductID
```

And the final query, which should be added to the SQL file, now becomes trivial:

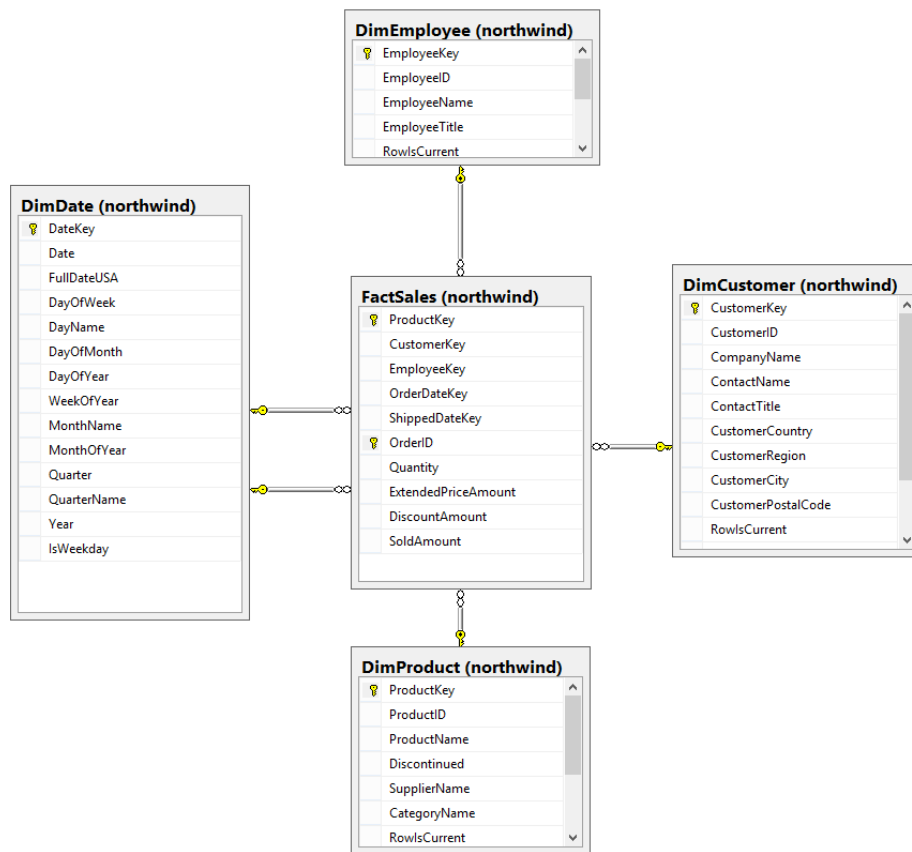
```
33 --loading FactSales
34 INSERT INTO [northwind].[FactSales]
35     ([ProductKey],[CustomerKey],[EmployeeKey]
36     ,[OrderDateKey]
37     ,[ShippedDateKey]
38     ,[OrderID]
39     ,[Quantity]
40     ,[ExtendedPriceAmount]
41     ,[DiscountAmount]
42     ,[SoldAmount])
43 SELECT p.ProductKey, c.CustomerKey, e.EmployeeKey,
44     [ExternalSources].[dbo].[getDateKey](s.OrderDate) as OrderDateKey,
45     case when [ExternalSources].[dbo].[getDateKey](s.ShippedDate) is null then -1
46     else [ExternalSources].[dbo].[getDateKey](s.ShippedDate) end as ShippedDateKey,
47     s.OrderID,
48     Quantity,
49     Quantity * UnitPrice as ExtendedPriceAmount,
50     Quantity * UnitPrice * Discount as DiscountAmount,
51     Quantity * UnitPrice * (1-Discount) as SoldAmount
52 FROM [ist722_mafudge_stage].[dbo].[stgNorthwindSales] s
53 join [ist722_mafudge_dw].[northwind].DimCustomer c
54     on s.CustomerID = c.CustomerID
55 join [ist722_mafudge_dw].[northwind].DimEmployee e
56     on s.EmployeeID = e.EmployeeID
57 join [ist722_mafudge_dw].[northwind].DimProduct p
58     on s.ProductID = p.ProductID
```

Be sure to save your file as **part2c-northwind-sales-data-load.sql**.

Part 2d: Did It Work? Verifying Your Dimensional Model

Once you get the data in the dimension and fact tables, we need to verify our dimensional model for accuracy. To do this, we create an SQL view to represent the dimensional model. This view will make it easier for you and business users to explore the dimensional model to verify its accuracy.

This view should contain all the tables in our star schema:



Creating the View

The pinnacle of your ROLAP star schema is a simple SQL view that joins the dimensions and facts together. This view should include all the rows from each dimension table and just the facts and degenerate dimensions from the fact table, which will make it easy and convenient for end-users to explore the data. Here's the query:

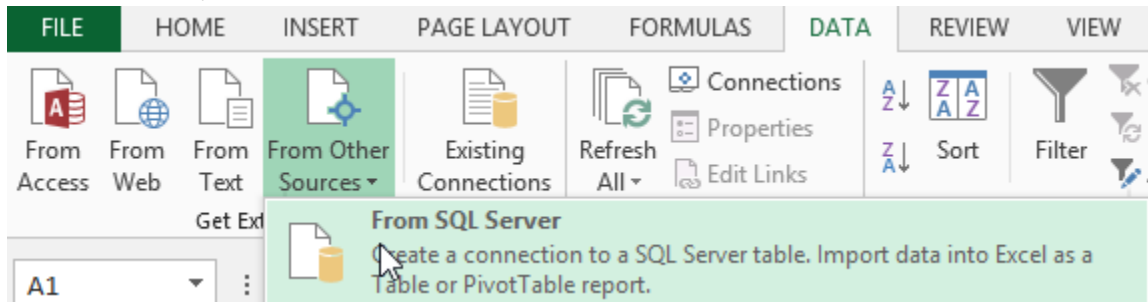
```
1 CREATE VIEW [northwind].[SalesMart]
2 AS
3 SELECT s.OrderID,s.Quantity, s.ExtendedPriceAmount, s.DiscountAmount, s.SoldAmount
4      ,c.CompanyName, c.ContactName, c.ContactTitle, c.CustomerCity
5      ,c.CustomerCountry, c.CustomerRegion, c.CustomerPostalCode
6      ,e.EmployeeName, e.EmployeeTitle
7      ,p.ProductName, p.Discontinued, p.CategoryName
8      ,od.*
9 FROM northwind.FactSales s
10  join northwind.DimCustomer c on c.CustomerKey = s.CustomerKey
11  join northwind.DimEmployee e on e.EmployeeKey = s.EmployeeKey
12  join northwind.DimProduct p on p.ProductKey = s.ProductKey
13  join northwind.DimDate od on od.DateKey = s.OrderDateKey
```

Connecting the View to Excel for Exploring the Data

What better way to explore your newly minted dimensional model than to use a BI tool? In this section we'll hook up the **SalesMart** to an Excel pivot table so we can explore the data.

First we need to connect to the source data.

1. Open Microsoft Excel. Create a blank workbook.
2. From the ribbon, choose **DATA** → **From Other Sources** → **From SQL Server**



3. Enter our class SQL Server information into the Data Connection Wizard dialog, and then click **Next>**

Data Connection Wizard

Connect to Database Server

Enter the information required to connect to the database server.

1. Server name: ist-cs-dw1.ad.syr.edu

2. Log on credentials

☒ Use Windows Authentication

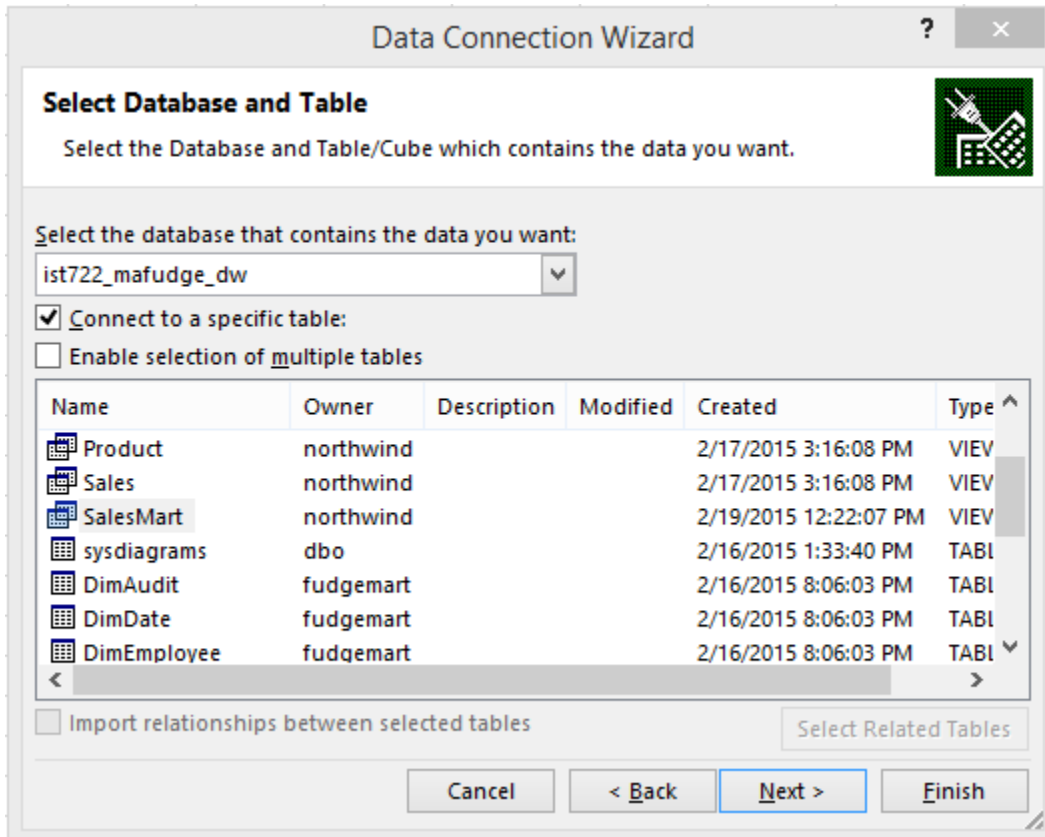
☐ Use the following User Name and Password

User Name:

Password:

Cancel < Back Next > Finish

4. Select your **ist722_netid_dw** database and select the **SalesMart** view. Then click **Next>**



Data Connection Wizard

Select Database and Table

Select the Database and Table/Cube which contains the data you want.

Select the database that contains the data you want:

ist722_mafudge_dw

☒ Connect to a specific table:

☐ Enable selection of multiple tables

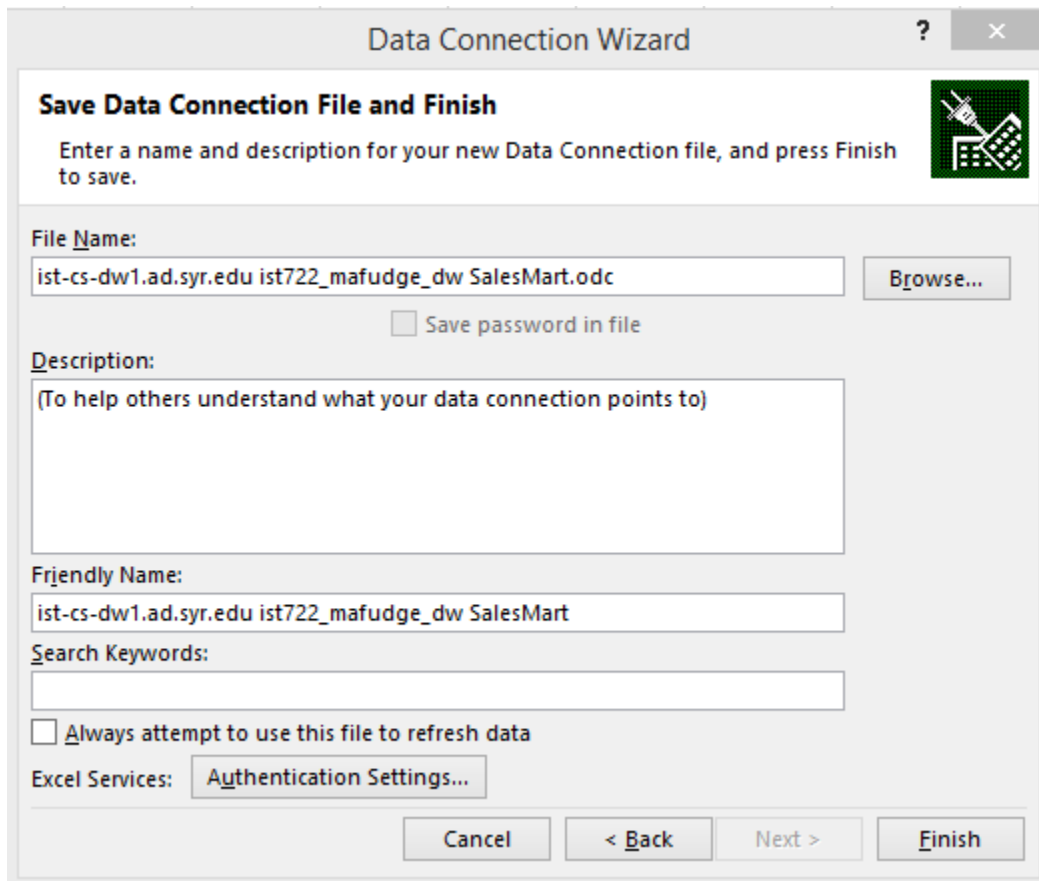
Name	Owner	Description	Modified	Created	Type
Product	northwind			2/17/2015 3:16:08 PM	VIEW
Sales	northwind			2/17/2015 3:16:08 PM	VIEW
SalesMart	northwind			2/19/2015 12:22:07 PM	VIEW
sysdiagrams	dbo			2/16/2015 1:33:40 PM	TAB
DimAudit	fudgemart			2/16/2015 8:06:03 PM	TAB
DimDate	fudgemart			2/16/2015 8:06:03 PM	TAB
DimEmployee	fudgemart			2/16/2015 8:06:03 PM	TAB

☐ Import relationships between selected tables

Select Related Tables

Cancel < Back **Next >** Finish

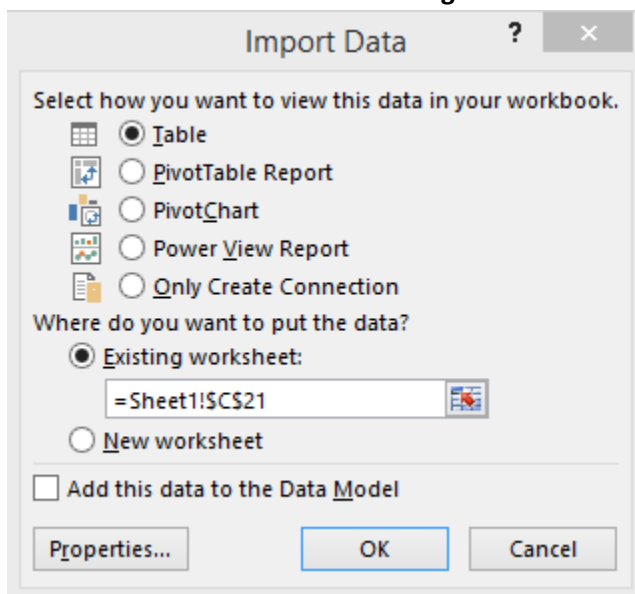
5. Just click **Finish** on this screen:



The screenshot shows the 'Data Connection Wizard' window, specifically the 'Save Data Connection File and Finish' step. The window has a title bar with a question mark and a close button. The main area contains the following fields and controls:

- File Name:** A text box containing 'ist-cs-dw1.ad.syr.edu ist722_mafudge_dw SalesMart.odc' and a 'Browse...' button to its right.
- ☐ Save password in file
- Description:** A large text area with the placeholder text '(To help others understand what your data connection points to)'.
- Friendly Name:** A text box containing 'ist-cs-dw1.ad.syr.edu ist722_mafudge_dw SalesMart'.
- Search Keywords:** An empty text box.
- ☐ Always attempt to use this file to refresh data
- Excel Services:** A button labeled 'Authentication Settings...'.
- At the bottom are four buttons: 'Cancel', '< Back', 'Next >', and 'Finish'.

6. This final screen allows you to decide what you need to do with the data. For this example, choose **Table** and then select **Existing worksheet** and click **OK**.

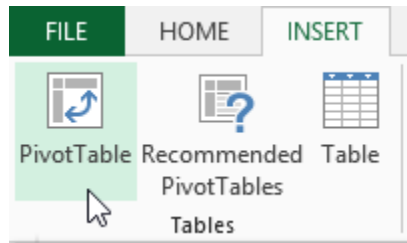


The screenshot shows the 'Import Data' dialog box. It has a title bar with a question mark and a close button. The main area contains the following options and controls:

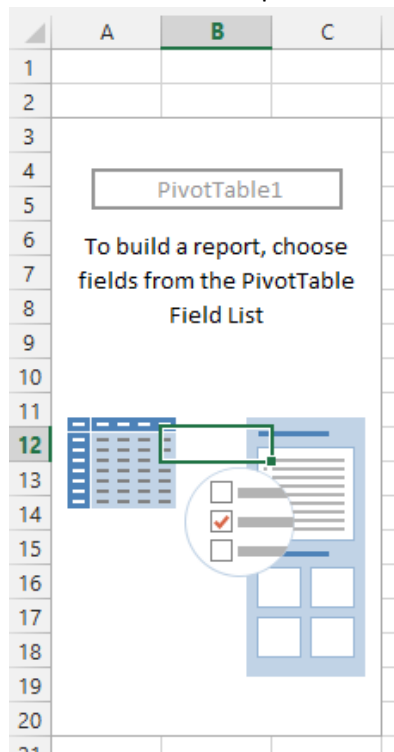
- Select how you want to view this data in your workbook.** A group box containing five radio button options:
 - ☒ Table
 - ☐ PivotTable Report
 - ☐ PivotChart
 - ☐ Power View Report
 - ☐ Only Create Connection
- Where do you want to put the data?** A group box containing two radio button options:
 - ☒ Existing worksheet:
 - A text box containing '=Sheet1!\$C\$21' and a small icon to its right.
 - ☐ New worksheet
- ☐ Add this data to the Data Model
- At the bottom are three buttons: 'Properties...', 'OK', and 'Cancel'.

Next we add the pivot table:

1. From the ribbon, choose **INSERT → PivotTable**



2. You will now see the pivot table screen:



Drag and drop the fields into the row, column, or values area of the pivot table and see if you can make the following pivot table reports:

3. Here's a setup of sales by day of the week:

Row Labels	Sum of SoldAmount
Monday	\$258,665.95
Tuesday	\$257,549.77
Wednesday	\$243,686.03
Thursday	\$236,813.68
Friday	\$269,077.62
Grand Total	\$1,265,793.04

4. And a breakdown of sales by product category and year:

Sum of SoldAmount	Column Labels			
Row Labels	1996	1997	1998	Grand Total
Beverages	\$47,919.00	\$103,924.31	\$116,024.88	\$267,868.18
Condiments	\$17,900.39	\$55,368.59	\$32,778.11	\$106,047.09
Confections	\$29,685.55	\$82,657.75	\$55,013.92	\$167,357.22
Dairy Products	\$40,980.45	\$115,387.64	\$78,139.19	\$234,507.28
Grains/Cereals	\$9,507.92	\$56,871.83	\$29,364.84	\$95,744.59
Meat/Poultry	\$28,813.66	\$80,975.11	\$53,233.59	\$163,022.36
Produce	\$13,885.78	\$54,940.77	\$31,158.03	\$99,984.58
Seafood	\$19,391.23	\$66,959.22	\$44,911.30	\$131,261.74
Grand Total	\$208,083.97	\$617,085.20	\$440,623.87	\$1,265,793.04

5. Here are the most heavily discounted products sold:

Row Labels	Sum of DiscountAmount
Côte de Blaye	\$8,587.47
Thüringer Rostbratwurst	\$7,367.73
Raclette Courdavault	\$5,140.30
Camembert Pierrot	\$3,460.52
Manjimup Dried Apples	\$2,922.95
Carnarvon Tigers	\$2,815.63
Alice Mutton	\$2,783.82
Tarte au sucre	\$2,592.93
Gnocchi di nonna Alice	\$2,528.14
Gudbrandsdalsost	\$2,364.84
Chang	\$2,203.24
Sirop d'érable	\$2,086.20

The queries are easy; the possibilities are endless, and you don't need to write a single line of SQL to get your questions answered. That, friends, is business intelligence!

Be sure to save your Excel pivot table as **part2-northwind-sales-pivot.xlsx**

Part 3: On Your Own

After you have finished up the sales reporting SQL, it's time to move on to order fulfillment!

In this part you will repeat the process outlined in Part 2, but this time you'll use **your own detailed dimensional design as opposed to the one I've provided**.

Instructions

Make sure your name and email address appear at the top of each SQL script. I should be able to execute your scripts, and they should work (create tables, import data, etc.).

1. Create the star schema /detailed dimensional model worksheet:
 - a. Follow your own detailed design from the previous lab as you create your implementation.
 - b. Conform your dimensions! Do not re-create dimensions that exist already; reuse them! If the dimension is already present from a previous step, there's no need to re-create it in SQL.
 - c. Save your SQL as **part3-northwind-order-fulfillment-rolap.sql**
2. Next stage the data.
 - a. You only need to stage the data you do not already have.
 - b. You should **always** stage the fact table, even if some of the data was staged already.
 - c. Save your SQL as **part3-northwind-order-fulfillment-stage.sql**
3. Finally, load the data.
 - a. Use your staged data as a source to populate the dimensions and the fact table.
 - b. Hopefully your technical specifications are good enough! If you encounter issues with the import, you might need to tweak your detailed dimensional model, regenerate the SQL and then retry the load.
 - c. Save your SQL as **part3-northwind-order-fulfillment-load.sql**
 - d. Create a view in your database joining all the tables together and then query your model with Excel to verify it makes sense.
 - e. Save your Excel pivot as **part3-northwind-order-fulfillment-pivot.xlsx**

Turning It In

Please turn all files from part 3. Make sure your name, NetID, and date appear somewhere on each of the files you include.

If you worked with a partner, please indicate that in your assignment by including your partner's name and NetID. You should both submit the assignment individually.