Suggestion 1: If you right-click in any of the SSIS or SSAS designers there is a feature called "Add Annotation". This feature is a good way to list tasks and communicate current status with team mates.
Suggestion 2: There are several ways to insert the required fictitious Internet Store and Salesperson rows.
(1) This approach only works in the situation where your dimension primary key is **not** an IDENTITY column. Recall that SELECT is capable of echoing string and numeric literals to a results set.
For example: SELECT 'Hello World!' AS Hello will create a result set with 1 row and 1 column where the column header is Hello and the row contains the text Hello World! To echo a UNICODE string you use a UNICODE string literal like so.
SELECT N'Hello World!' AS Hello Prepending any string with a capital **N** is how the powers that be decided one crafts UNICODE string literals in SQL. You might use the *Union All* SSIS transformation to join such a souce data flow to another data flow.
(2) If your dimension primary key is **is** an IDENTITY column then you must consider using 2 data flow tasks to build these tables. One of the data flow tasks will simply move the data from OLTP sources to the DW destination. The other data flow task will create the fictitious row as in (1); however, in this case the data flow destination will be configured to "Keep Identity". See the *Keep Identity* checkbox in the **OLE DB Destination Editor**. Selecting this checkbox will allow you to insert rows with identity value column values included.

Suggestion 3: As is normal, you will find developing your SSIS solution requires iteration. It is very helpful to create packages that do things like delete all rows from a table so that you can re-load the table data without having to DROP and re-CREATE the table. Of course, if you need to make changes to the design of your tables you will be forced to DROP and re-CREATE them. See the *Execute SQL Task* we used in the SSIS lab.

Suggestion 4: If you right-click any existing table in SS Mgt Studio you can "view" the DDL statements used to create it. You can then use this DDL as a starting point to any other table you might want to create - say dimension or fact tables. To do so select Script Table As > CREATE To > New Query Editor Window (or which ever option you prefer).

Suggestion 5: The primary key to DimDate (DateKey) is a 4 byte signed integer. That means in order to join DimDate and the FactTable to perform queries, the corresponding foreign key in the FactTable (OrderDate) must be a 4 byte signed integer - which it is not. In the OLTP SalesOrder table, OrderDate is a DateTime data type. You will need to rectify this by converting the OrderDate column to an integer compatible with the primary key (DateKey) in DimDate. This can be accomplished in the SELECT statement that you are using as the OLTP source for your FactTable. Here's an example SELECT statement that you can try in SQL Server Mgt Studio.

```
SELECT CAST(CONVERT(CHAR(8), SalesOrder.OrderDate, 112) AS INT)  AS OrderDate
FROM SalesOrder ...
```

This SELECT will result in values for OrderDate that are compatible with the DateKey column in DimDate. You can then use this idea in your project. Don't like the looks of this? You can also accomplish the same thing using a data flow transformation.

Suggestion 6: While there are many ways to add the *SalesReasons* information into the customer table, the most efficient and best way is to use the **MergeJoin** transformation. This transformation works similar to a FROM statement that joins two tables. Make sure to use a **LEFT OUTER** join (the default is **INNER JOIN**) as all customers do not appear in the SalesReasons table and you need to retain all of the customers in the data flow to that point. You can also use the **Lookup** transformation.

Suggestion 7: If you create your **DimDate** table using **nvarchar** data types you are creating a table that uses a **Unicode** encoding for its contents. First, you should consistently either use (or not use) the **nvarchar** type within a single table, If you do use **nvarchar**, then you will need to use Excel's **Save As... Unicode** option to save your date worksheet to a **Unicode** text file. This will work just as well in SSIS as a CSV file only the delimiter will be a TAB instead of a comma. SSIS is smart enough to figure all of this out when you use the **Unicode** file as input to the **Flat File Source** data flow source. Note that you will also (most likely) need to set the **Text Qualifier** in the **Flat File Connection Manager** to be one double quote character (i.e., "). You can use the **Notepad** program to examine the contents of the text file saved by Excel.

Suggestion 8: "I'm using an Excel data source and I can design my package; however it won't run - at all?". A little background to start. SQL Server Data Services (SSDS) is a 32-bit application. When you're designing your package, you're using 32-bit facilities - and have no choice in the matter. When you execute your package it is actually sent to another SQL Server process for execution. On a 64-bit computer this process may use 64-bit mode. However, some commonly used objects in SSIS (like Excel connections) don't have 64-bit counterparts, and will therefore cause your packages to fail in odd ways. Lately, the package seems to do nothing at all and just appears to hang.
Anyway, when running your package inside SSDS, the setup is simple (unless you're using the Execute Package Task or Execute Process Task to run child packages). The package you currently have open will (by default) run in 64 bit mode on a 64-bit installation. The setting that controls this is a property on the project called Run64BitRuntime. To access this property, right-click on the Integration Services project in your solution explorer and select Properties. Then select the Debugging node in the editor. The default here is "true", which means all the packages in this project will run in 64-bit mode. If you change this to "false", all the packages will be run in 32-bit mode.