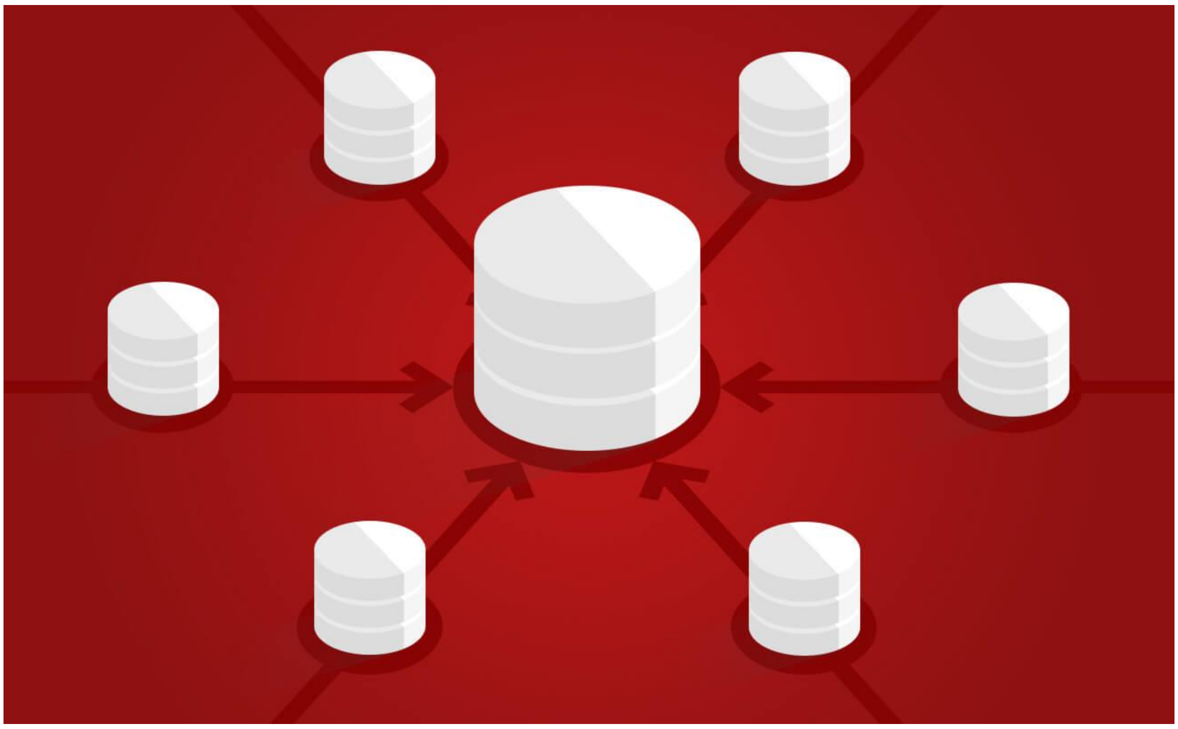
Referential integrity and its role in data warehousing: part one



23/06/2015

In this pair of tech blog posts, Rob Hawthorne, our head of Microsoft Solutions and Architecture, walks us through the steps needed to implement referential integrity in our data warehouse and keep our DBAs happy. This blog is targeted at business intelligence developers or those with a keen interest in learning new ways of managing metadata in SQL Server.

As a business intelligence developer, I have heard many, many times from DBAs that data warehouses should have referential integrity (RI). However, as any BI dev knows, this is not that easy. Why? Well, generally speaking, when we are loading facts and dimensions, we tend to dump all the data and do a full re-load of these tables. This makes it simple to do joins (for key lookups) and ensures that the keys line up properly. So far so good. However, if you have ever tried to truncate a table with RI enabled, you would see messages like the following:

```
TRUNCATE TABLE [dimension].[Date]
GO
Msg 4712, Level 16, State 1, Line 1
Cannot truncate table 'dimension.Date' because it is being referenced by a FOREIGN KEY
constraint.
```

While it is fine to truncate the fact table, you can't truncate the dimension. So most BI devs (and most solutions I have seen) do not implement RI, as it is just a bit of a pain at the end of the day.

So how do we keep DBAs happy, but also make it easy for ourselves to manage RI? Thankfully SQL Server comes to the rescue.

Internally, SQL Server holds a lot of metadata about tables, indexes, relationships, column types, etc. This means that all that information is queryable through the system catalog views (as long as you have the appropriate permissions of course).

In this article we are going to look at how to use that information to build our RI, and tear it down and re-establish when we have completed our ETL. There are code examples, so you do need to be comfortable with SSIS and basic T-SQL.

This article is built using the AdventureWorksDW2014 database, which you can download from CodePlex.

Download the source code.

Note: Use of this Referential Integrity Manager is entirely at your own risk. Theta makes no representation or warranty regarding its use.

Getting the Information from SQL

The first step is to get the RI information out of SQL. We use the system views to get this. We will use the following views:

- **sys.foreign_key_columns** – Provides information about the columns (fields) used in a foreign key within a database - https://msdn.microsoft.com/en-us/library/ms186306.aspx
- **sys.foreign_keys** – Provides information about the foreign keys in a database, i.e. name, referenced object, etc. - https://msdn.microsoft.com/en-us/library/ms189807.aspx
- **sys.tables** – Provides information about the tables in a database, i.e. name, object_id, type, schema it belongs too, etc. - https://msdn.microsoft.com/en-us/library/ms187406.aspx
- **sys.schemas** – Provides information about the schemas in a database, i.e. name, object_id, owner. - https://msdn.microsoft.com/en-us/library/ms176011.aspx
- **sys.columns** – Provides information about the columns within a table (or view), i.e. name, data type, size, table it belongs too, etc. - https://msdn.microsoft.com/en-us/library/ms176106.aspx

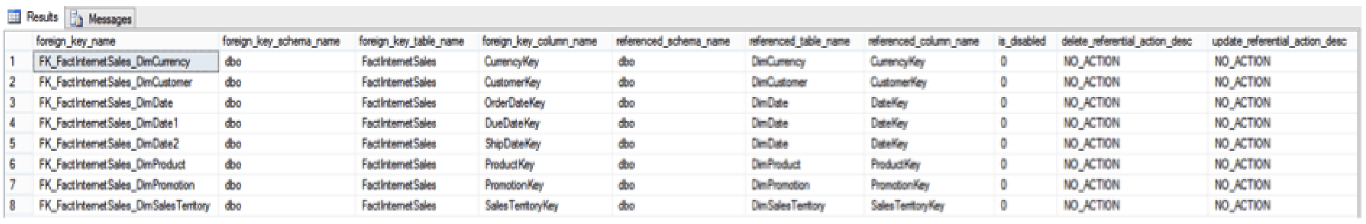
As you can see there is some pretty rich metadata about the objects within a database.

The following query will retrieve all of the information we need about the RI in our database:

NOTE: This is limited to a single schema (dbo) and a single fact table (FactInternetSales).

```
SELECT
      fk.name AS foreign_key_name,
      sp.name AS foreign_key_schema_name,
      p.name AS foreign_key_table_name,
      pc.name AS foreign_key_column_name,
      sr.name AS referenced_schema_name,
      c.name AS referenced_table_name,
      rc.name AS referenced_column_name,
      fk.is_disabled,
      fk.delete_referential_action_desc,
      fk.update_referential_action_desc
FROM sys.foreign_key_columns fkc
INNER JOIN sys.foreign_keys fk ON fk.object_id = fkc.constraint_object_id
INNER JOIN sys.tables p ON p.object_id = fkc.parent_object_id
INNER JOIN sys.schemas sp ON sp.schema_id = p.schema_id
INNER JOIN sys.tables c ON c.object_id = fkc.referenced_object_id
INNER JOIN sys.schemas sr ON sr.schema_id = c.schema_id
INNER JOIN sys.columns pc ON pc.object_id = p.object_id AND pc.column_id =
fkc.parent_column_id
INNER JOIN sys.columns rc ON rc.object_id = c.object_id AND rc.column_id =
fkc.referenced_column_id
WHERE sp.name = 'dbo'
AND p.name = 'FactInternetSales'
```

This brings back a result set like this, giving us the name of the foreign key, the foreign key schema, foreign key table name, the referenced table name, schema and column.



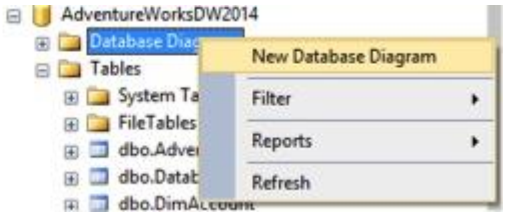
Results | Messages

	foreign_key_name	foreign_key_schema_name	foreign_key_table_name	foreign_key_column_name	referenced_schema_name	referenced_table_name	referenced_column_name	is_disabled	delete_referential_action_desc	update_referential_action_desc
1	FK_FactInternetSales_DimCurrency	dbo	FactInternetSales	CurrencyKey	dbo	DimCurrency	CurrencyKey	0	NO_ACTION	NO_ACTION
2	FK_FactInternetSales_DimCustomer	dbo	FactInternetSales	CustomerKey	dbo	DimCustomer	CustomerKey	0	NO_ACTION	NO_ACTION
3	FK_FactInternetSales_DimDate	dbo	FactInternetSales	OrderDateKey	dbo	DimDate	DateKey	0	NO_ACTION	NO_ACTION
4	FK_FactInternetSales_DimDate1	dbo	FactInternetSales	DueDateKey	dbo	DimDate	DateKey	0	NO_ACTION	NO_ACTION
5	FK_FactInternetSales_DimDate2	dbo	FactInternetSales	ShipDateKey	dbo	DimDate	DateKey	0	NO_ACTION	NO_ACTION
6	FK_FactInternetSales_DimProduct	dbo	FactInternetSales	ProductKey	dbo	DimProduct	ProductKey	0	NO_ACTION	NO_ACTION
7	FK_FactInternetSales_DimPromotion	dbo	FactInternetSales	PromotionKey	dbo	DimPromotion	PromotionKey	0	NO_ACTION	NO_ACTION
8	FK_FactInternetSales_DimSalesTerritory	dbo	FactInternetSales	SalesTerritoryKey	dbo	DimSalesTerritory	SalesTerritoryKey	0	NO_ACTION	NO_ACTION

It also extracts the actions that the key takes when actions like delete and updates are executed.

For more information on **Cascading Referential Integrity Constraints** see https://technet.microsoft.com/en-us/library/ms186973%28v=sql.105%29.aspx?f=255&MSPPError=-2147217396.

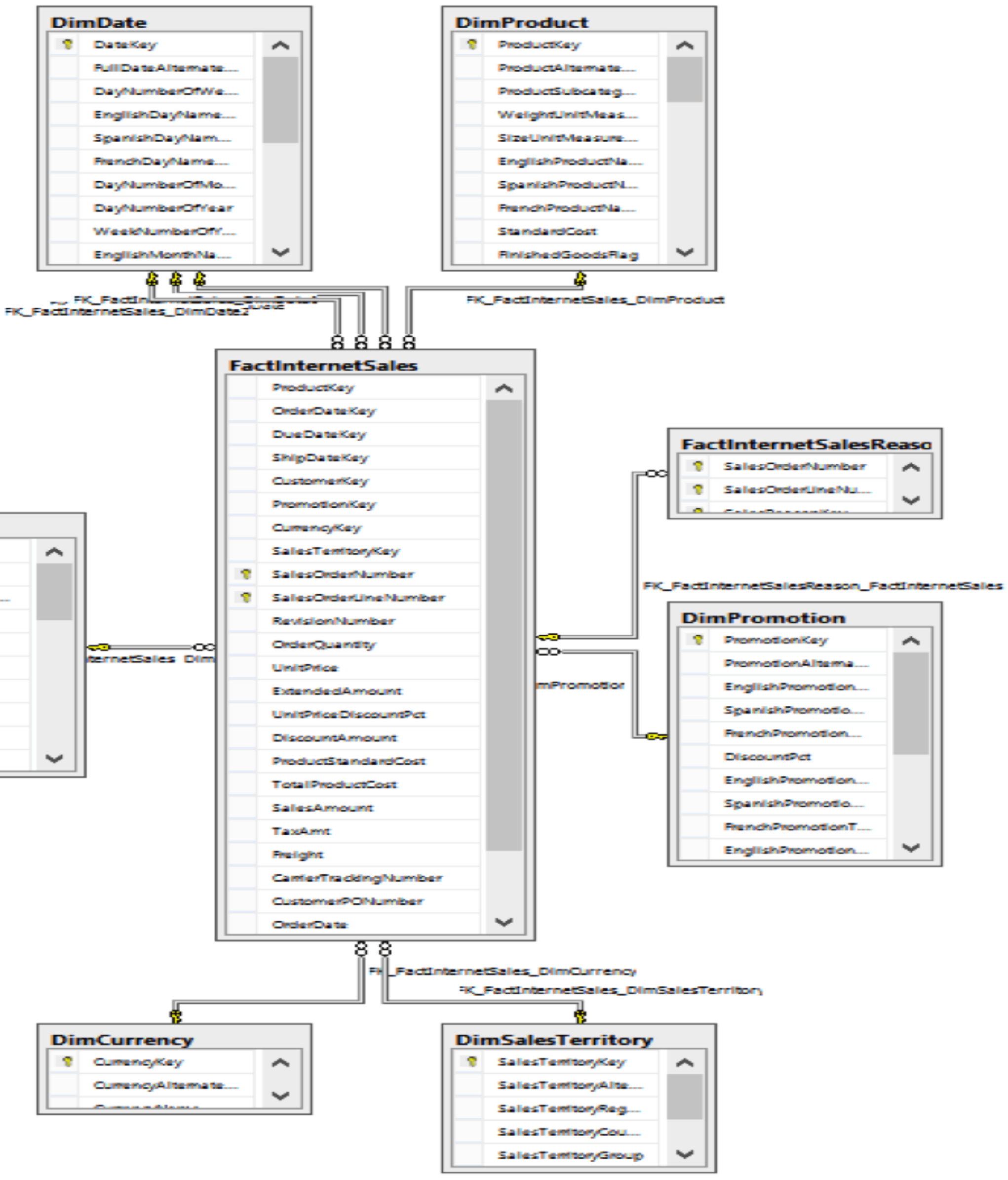
Right, so at this stage we have the information, but you may find that your database doesn't have any RI constraints. The quickest way to create this is via the database diagraming tool within SQL Server.



Go ahead and build your database diagram. This will serve 2 purposes:

1. You can carry on with this article!
2. You can validate that your data will support RI being enabled

Since the Adventure Works DW is based on Kimball’s methodology we have a star schema, which resembles this:



This is a typical scenario, lots of relationships and also multiple keys referencing the same table (i.e. three constraints referencing the DimDate dimension).

We now have the RI enabled for our star schema. Let's look at how we can use this in our ETL.

Structuring our ETL for RI

The first thing to do is to extract the RI and store it somewhere. When I first started on this path, I put the RI configuration into memory, but if you stop the ETL (especially when developing) you lose the configuration and have to start all over again! So lesson learnt, we will dump the configuration out into a table for storage.

Just add the following line to the SELECT statement:

```
INTO dbo.ForeignKeyReferences
```

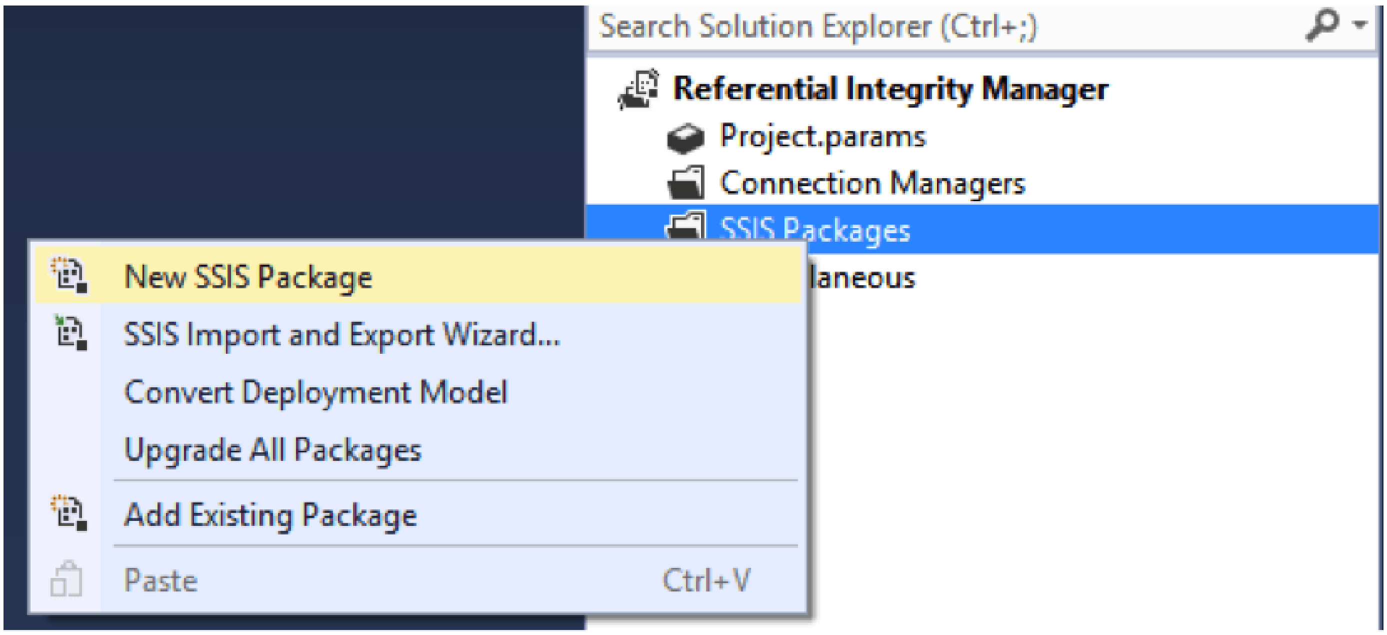
For example:

```
SELECT
      fk.name AS foreign_key_name,
      sp.name AS foreign_key_schema_name,
      p.name AS foreign_key_table_name,
      pc.name AS foreign_key_column_name,
      sr.name AS referenced_schema_name,
      c.name AS referenced_table_name,
      rc.name AS referenced_column_name,
      fk.is_disabled,
      fk.delete_referential_action_desc,
      fk.update_referential_action_desc
INTO dbo.ForeignKeyReferences
FROM sys.foreign_key_columns fkc
INNER JOIN sys.foreign_keys fk ON fk.object_id = fkc.constraint_object_id
INNER JOIN sys.tables p ON p.object_id = fkc.parent_object_id
INNER JOIN sys.schemas sp ON sp.schema_id = p.schema_id
INNER JOIN sys.tables c ON c.object_id = fkc.referenced_object_id
INNER JOIN sys.schemas sr ON sr.schema_id = c.schema_id
INNER JOIN sys.columns pc ON pc.object_id = p.object_id AND pc.column_id =
fkc.parent_column_id
INNER JOIN sys.columns rc ON rc.object_id = c.object_id AND rc.column_id =
fkc.referenced_column_id
WHERE sp.name = 'dbo'
AND p.name = 'FactInternetSales'
```

This will allow us to store the configuration of our Foreign Keys in a table within our database.

Here's a tip: make another copy of the data into a secondary table as well. Sometimes when developing you have to empty the **dbo.ForeignKeyReferences** table and as such you won't want to rebuild the entire data set again. Create a second table either within the same database or a configuration database. It will come in handy. I called my second table **dbo.ForeignKeyReferences_Backup**. You can name it anything that makes sense to you.

Now the fun part! We will create a new SSIS task to deal with the RI. In your Visual Studio project, add a new SSIS package.



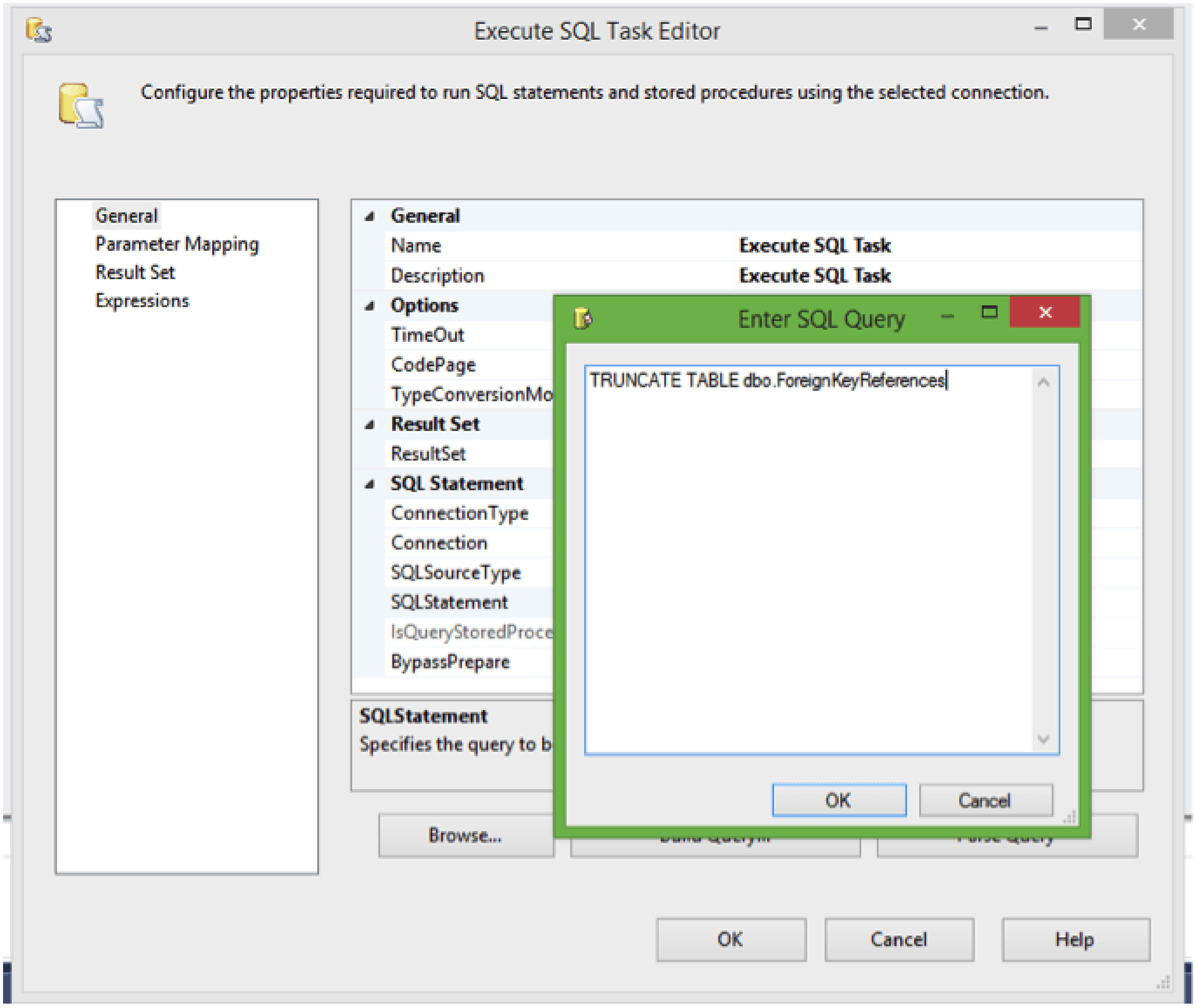
I call this new package **000 - Preliminary Tasks.dtsx,** as I want it to be clear it executes before any of my ETL starts. Again you can name it whatever you like.

Building the SSIS task:

The first step is to retrieve the information we have and store it within the **dbo.ForeignKeyReferences** table. You want this to be refreshed every time the package runs (i.e. in case the RI changes), so our very first step is to add a TRUNCATE TABLE to the package (see now why we have a second table?).

In the new package add an Execute SQL Task. If you don't have a data connection setup (and you probably won't as this is a new package) go ahead an add a connection to your SQL instance and database that contains the **dbo. ForeignKeyReferences** table.

In the Execute SQL Query task editor, add the line TRUNCATE TABLE dbo.ForeignKeyReferences, as shown below:



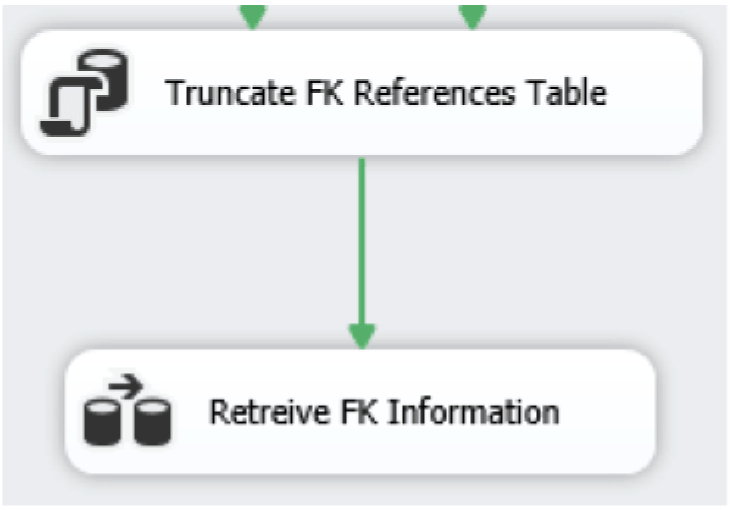
Name: Execute SQL Task
Description: Execute SQL Task

```
TRUNCATE TABLE dbo.ForeignKeyReferences
```

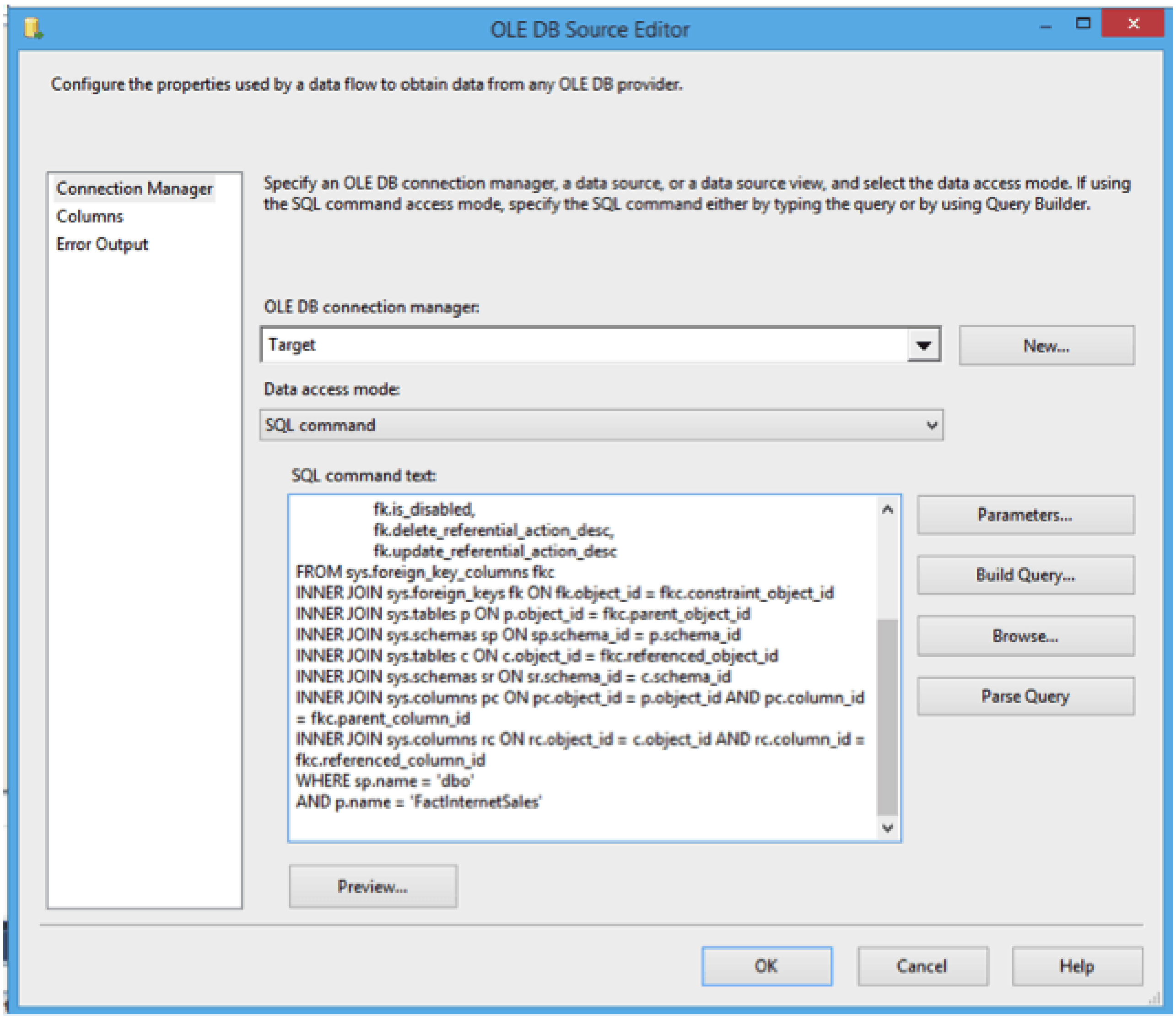
You should also name the Execute SQL Task appropriately. I've called mine **Truncate FK References Table.**

Note: If you want to build more robust code, you should put in a check for the existence of this table first and either error out if it doesn't exist or choose a different path and create the table. I have added this second option into the sample code with this article, but for sake of brevity I have excluded it from the article.

Now add a Data Flow Task to retrieve the information from the sys catalogs (I've called mine **Retrieve FK Information**):



In the Data Flow Task, use the Source Assistant (it will create an OLE DB task based on the connection type you created earlier). Set the Data access mode to SQL command, and paste the query (2) we wrote earlier.



Configure the properties used by a data flow to obtain data from any OLE DB provider.

Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder.

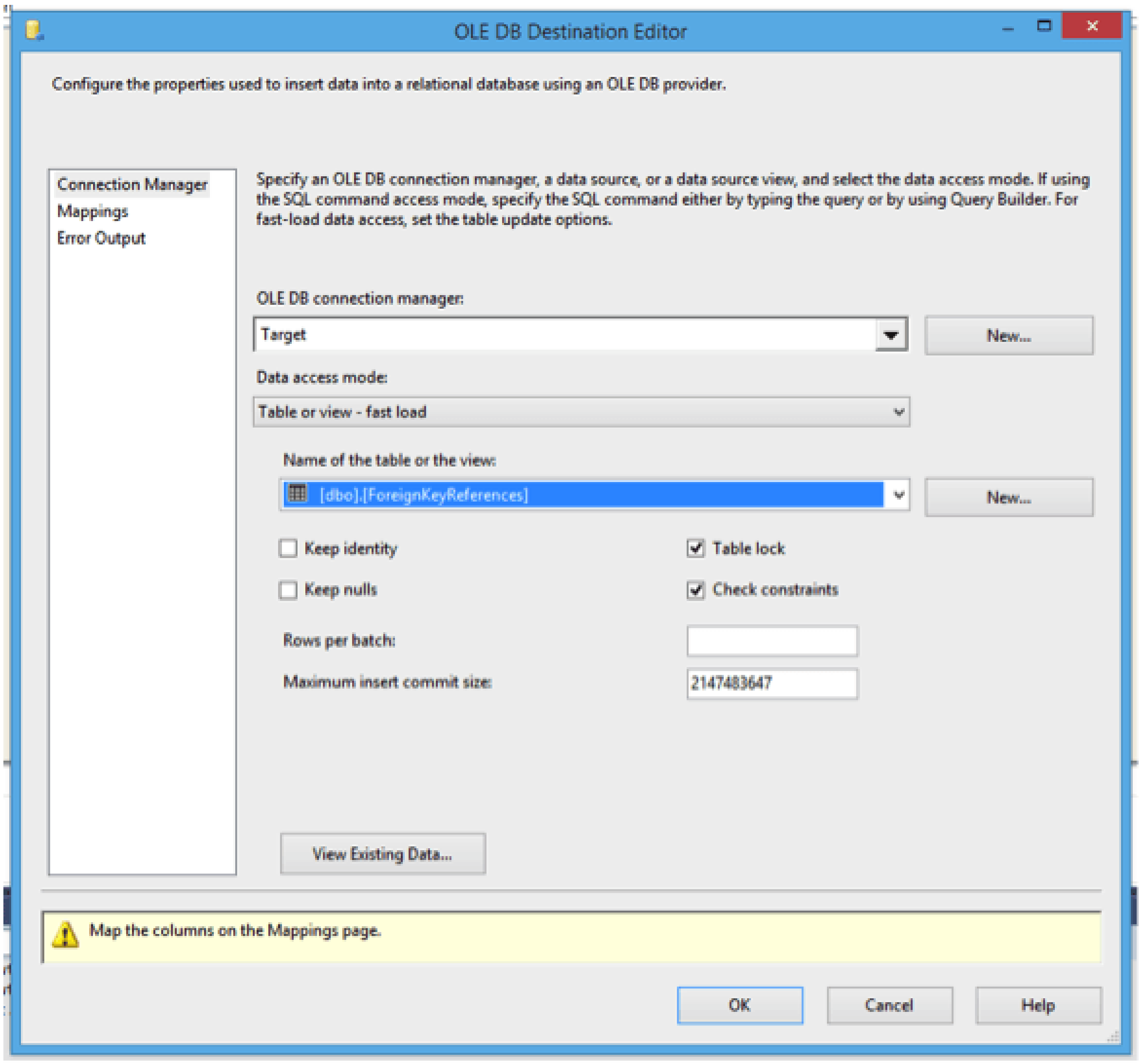
OLE DB connection manager: Target

Data access mode: SQL command

SQL command text:

```
            fk.is_disabled,
            fk.delete_referential_action_desc,
            fk.update_referential_action_desc
FROM sys.foreign_key_columns fkc
INNER JOIN sys.foreign_keys fk ON fk.object_id = fkc.constraint_object_id
INNER JOIN sys.tables p ON p.object_id = fkc.parent_object_id
INNER JOIN sys.schemas sp ON sp.schema_id = p.schema_id
INNER JOIN sys.tables c ON c.object_id = fkc.referenced_object_id
INNER JOIN sys.schemas sr ON sr.schema_id = c.schema_id
INNER JOIN sys.columns pc ON pc.object_id = p.object_id AND pc.column_id
= fkc.parent_column_id
INNER JOIN sys.columns rc ON rc.object_id = c.object_id AND rc.column_id =
fkc.referenced_column_id
WHERE sp.name = 'dbo'
AND p.name = 'FactInternetSales'
```

Now we need to write this information to our table. Use the Destination Assistant wizard to create a target. Open the destination and set the table to the **ForeignKeyReferences** table.



OLE DB Destination Editor

Configure the properties used to insert data into a relational database using an OLE DB provider.

Connection Manager
Mappings
Error Output

Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder. For fast-load data access, set the table update options.

OLE DB connection manager: Target

Data access mode: Table or view - fast load

Name of the table or the view: [dbo].[ForeignKeyReferences]

Keep identity [ ] Table lock [✓]
Keep nulls [ ] Check constraints [✓]

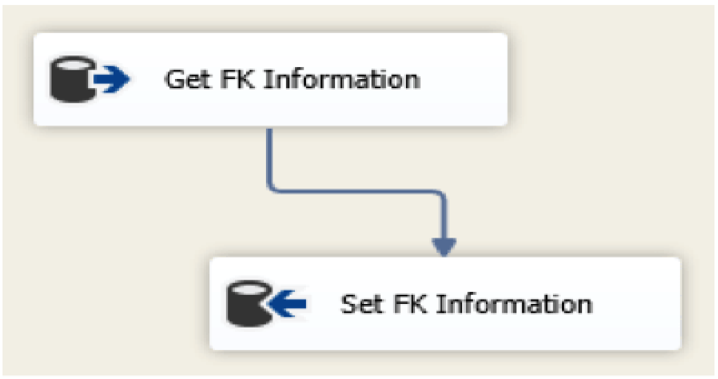
Rows per batch:

Maximum insert commit size: 2147483647

Map the columns on the Mappings page.

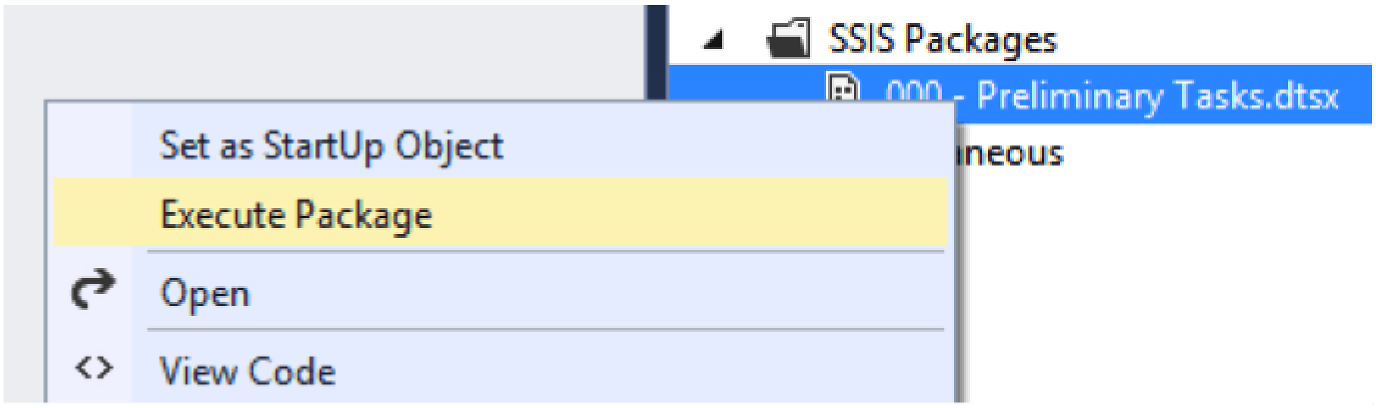
Note: Check the Mappings tab to ensure that the columns are lined up correctly.

You should now have a Data Flow Task that is similar to:

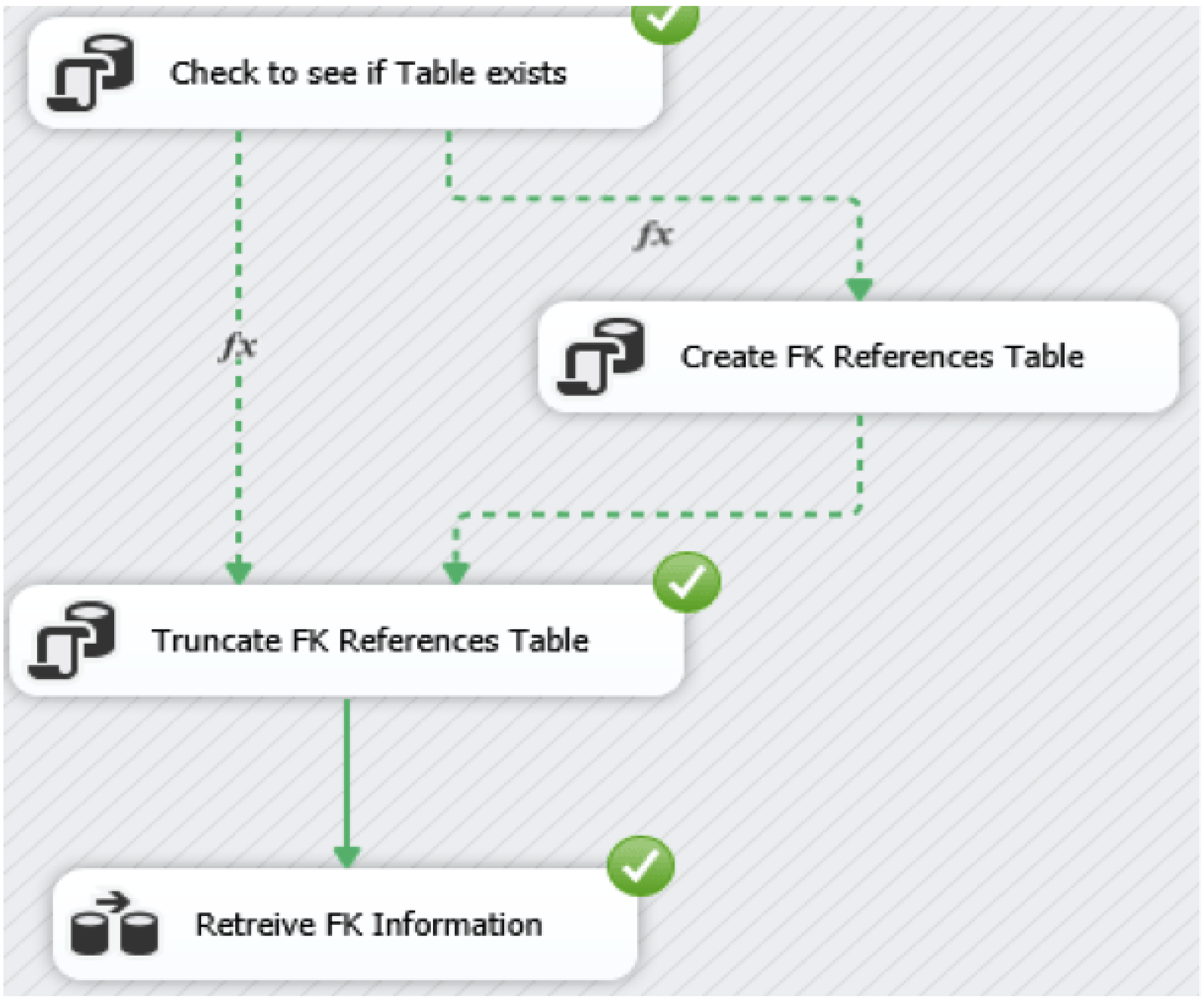


Save your work!

Now we can test the package to ensure it is running as expected. Right click on the package and select Execute:



If all is well the tasks should show up as executing successfully (ticks next to the successful tasks):


Check to see if Table exists
fx
fx
Create FK References Table
Truncate FK References Table
Retreive FK Information

Go ahead and check that the **dbo.ForeignKeyReferences** table holds the FK information.

If it doesn't or the package didn't execute successfully, check the Execution Results tab in Visual Studio, and check for any configuration mistakes, i.e. missing data source connections, incorrectly typed code (especially if copied from a web page), etc.

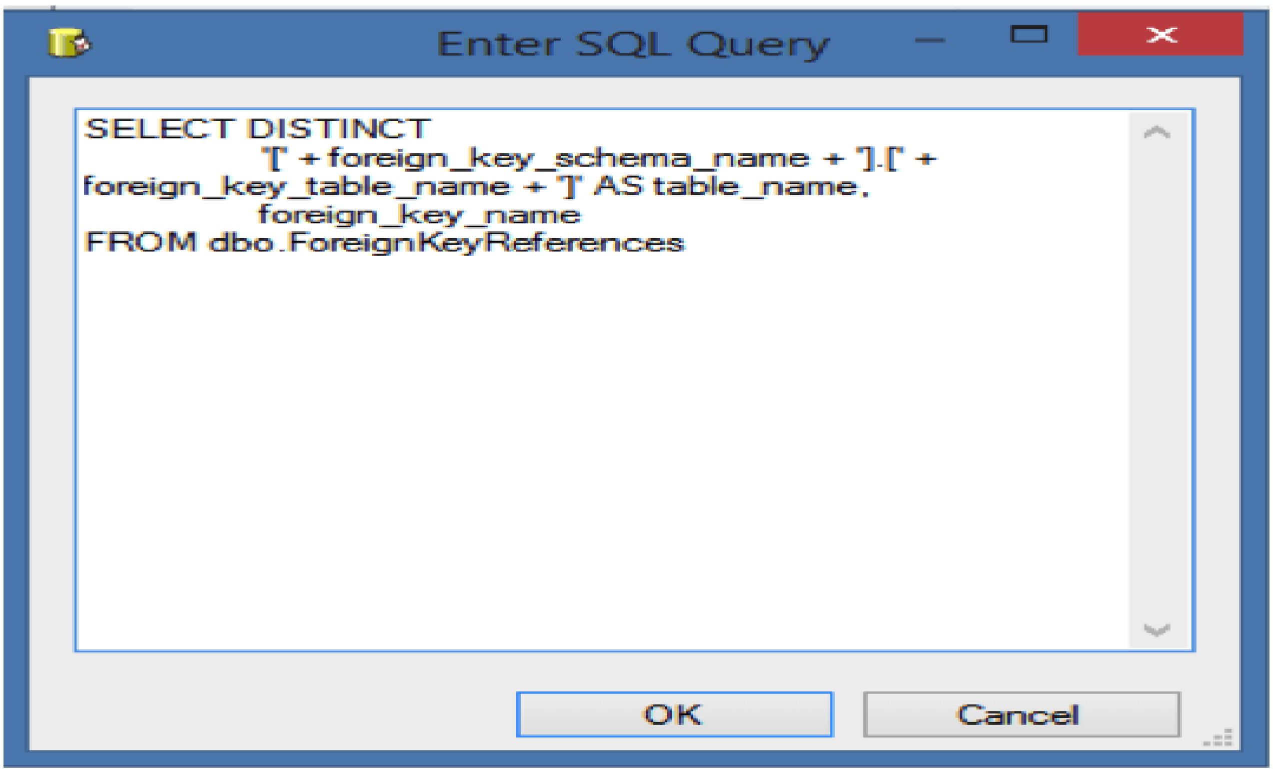
So to recap: we now have a small SSIS task that extracts the foreign key definitions from within SQL Server, and stores within a table. Next we will create a For Loop container to remove all of the foreign key definitions from the database.

Dropping the FK References:

Let's now add a new Execute SQL Task to get the names of all the FK references from the data we have stored in our table. Have the success flow from the **Retrieve FK Information** Data Flow task connect to the newly added Execute SQL Task (I have named mine **Get FK Listing**).

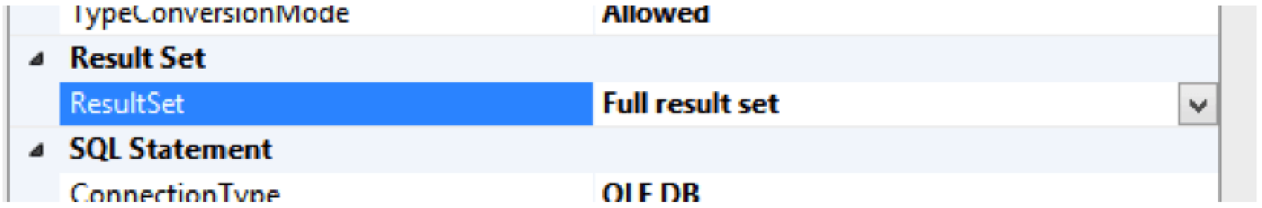
Use the following query in the Execute SQL task:

```
SELECT DISTINCT
       '[' + foreign_key_schema_name + '].[' + foreign_key_table_name + ']' AS
table_name,
       foreign_key_name
FROM dbo.ForeignKeyReferences
```



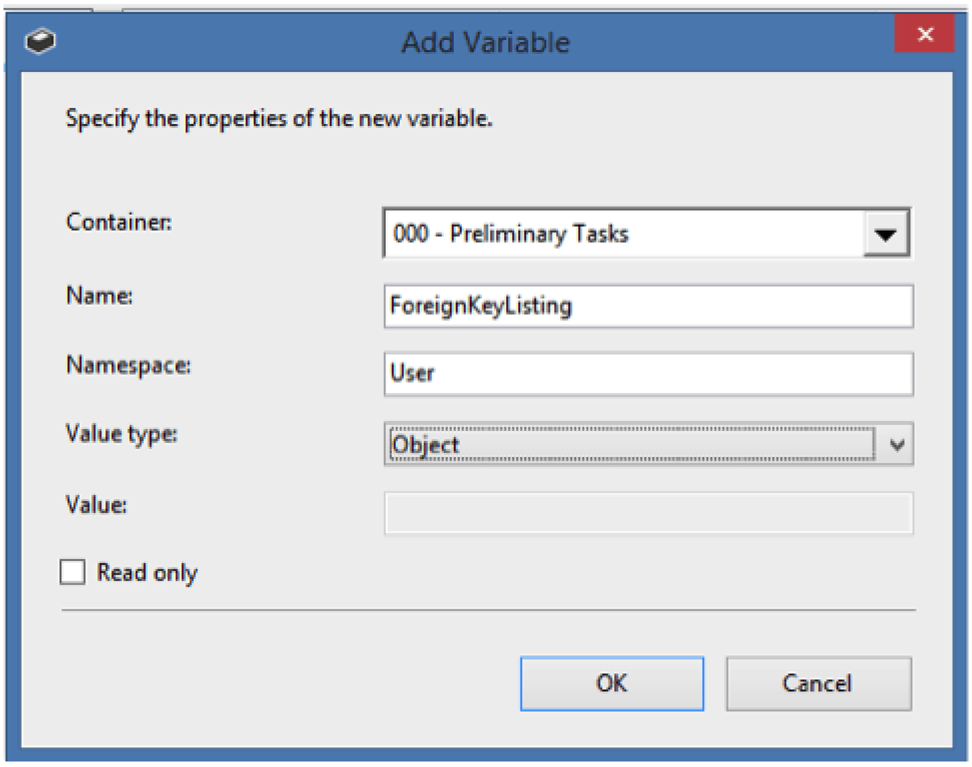
```
SELECT DISTINCT
            '[' + foreign_key_schema_name + '].[' +
foreign_key_table_name + ']' AS table_name,
            foreign_key_name
FROM dbo.ForeignKeyReferences
```

We will place this into a variable, so we can iterate through the result set and drop the constraints.



TypeConversionMode	Allowed
Result Set	
ResultSet	Full result set
SQL Statement	
ConnectionType	OLE DB

On the “Result Set” tab, Add and create the variable:

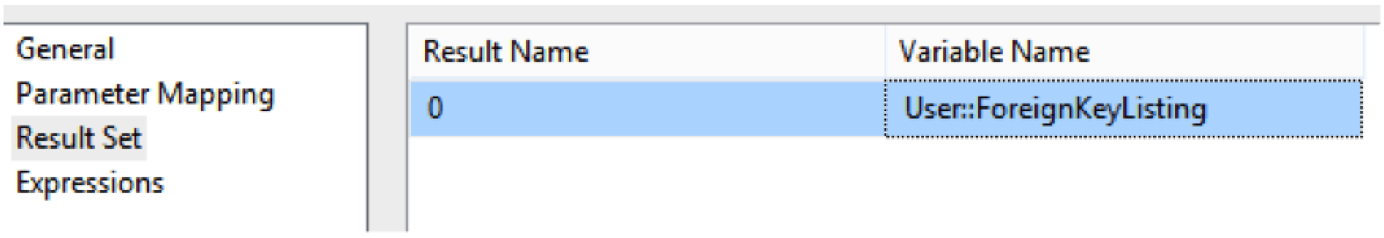


Add Variable

Specify the properties of the new variable.

Container:	000 - Preliminary Tasks
Name:	ForeignKeyListing
Namespace:	User
Value type:	Object
Value:	

☐ Read only



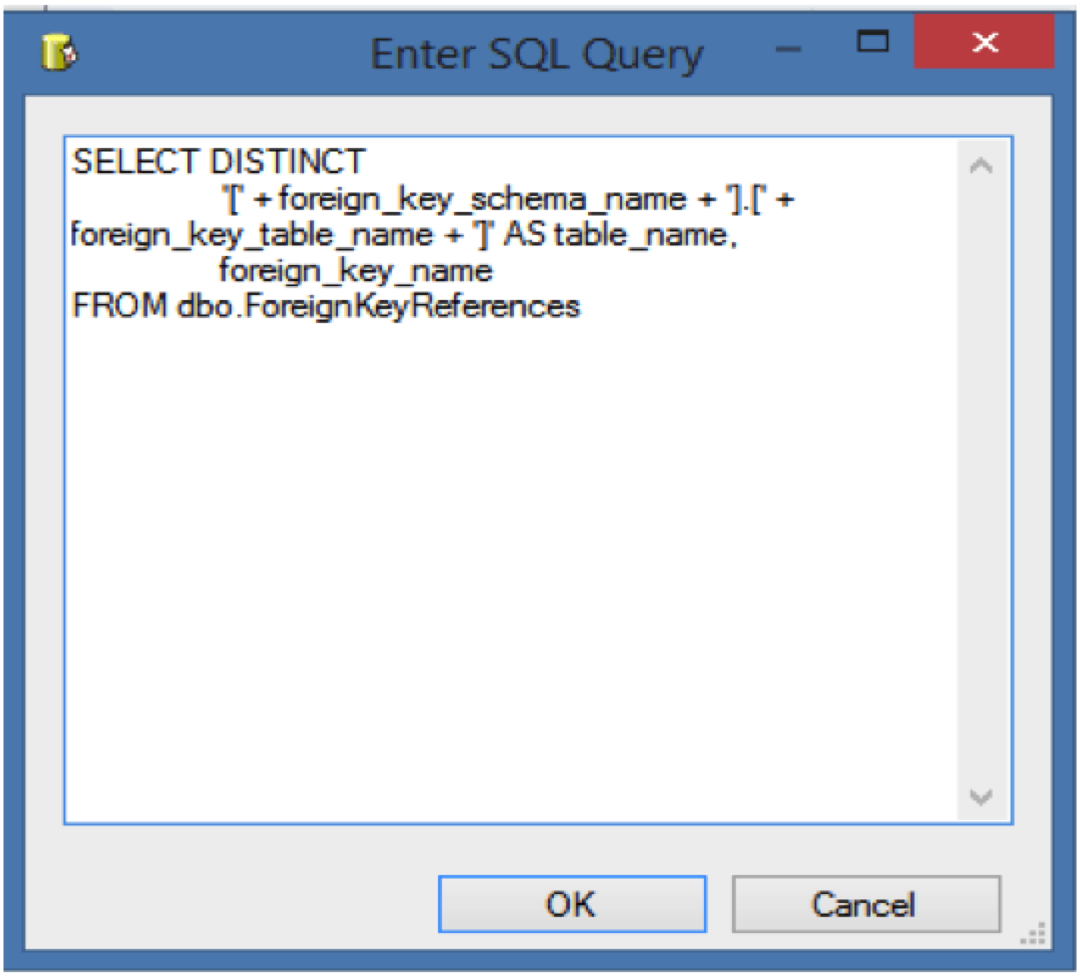
Result Name	Variable Name
0	User::ForeignKeyListing

We will use this object variable to iterate through in the next step.

Add a Foreach Loop Container to the package, and have the success flow from the **Get FK Listing** task connect to the Foreach Loop Container:

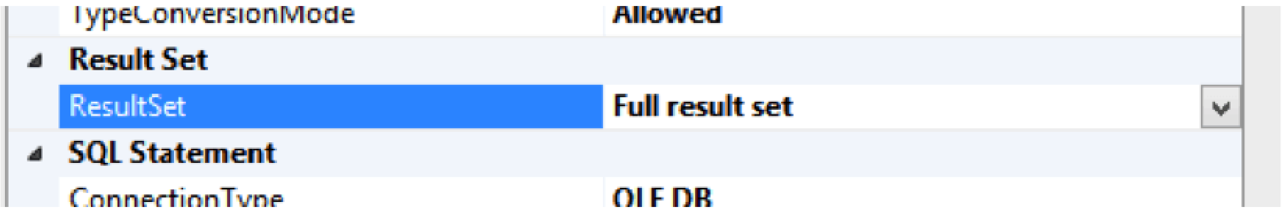
Use the following query in the Execute SQL task:

```
SELECT DISTINCT
       '[' + foreign_key_schema_name + '].[' + foreign_key_table_name + ']' AS
table_name,
       foreign_key_name
FROM dbo.ForeignKeyReferences
```



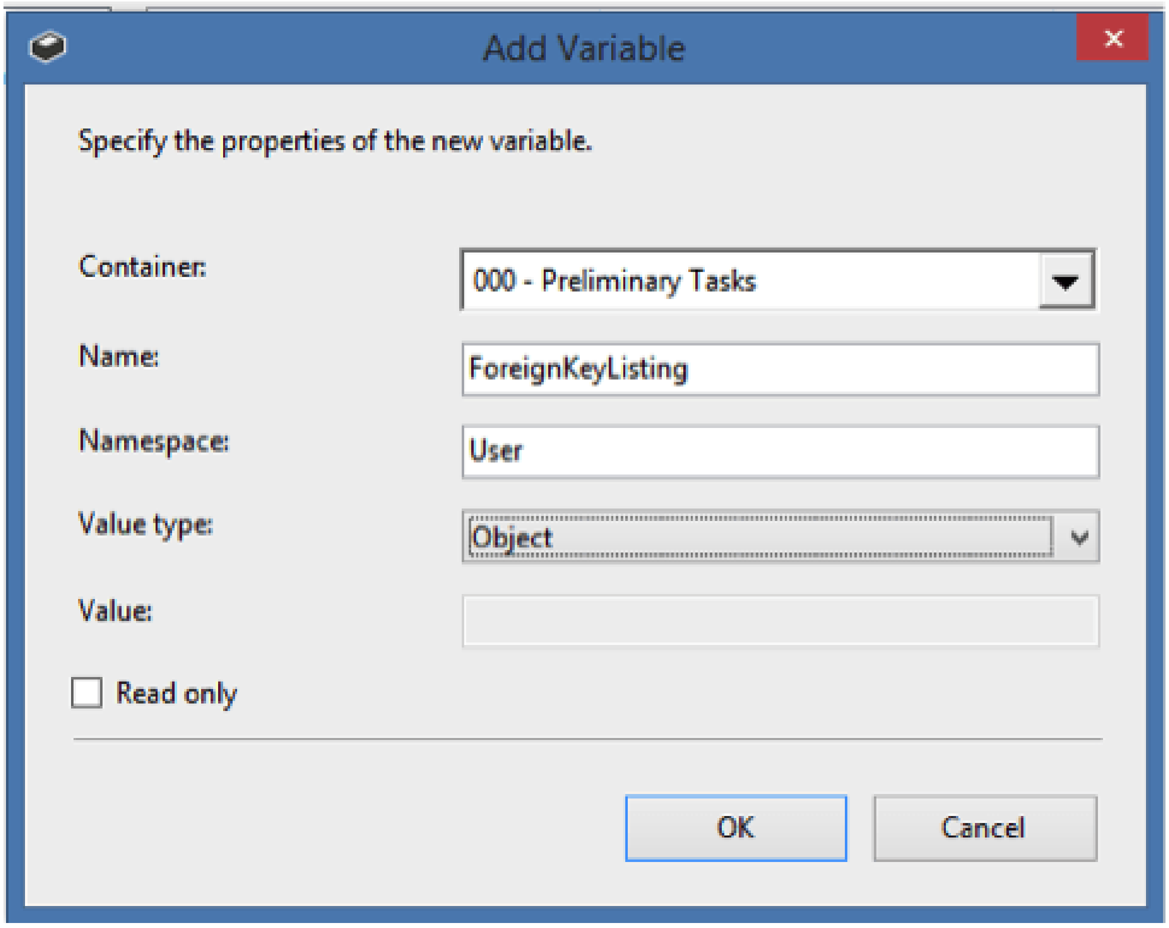
```
SELECT DISTINCT
            '[' + foreign_key_schema_name + '].[' +
foreign_key_table_name + ']' AS table_name,
            foreign_key_name
FROM dbo.ForeignKeyReferences
```

We will place this into a variable, so we can iterate through the result set and drop the constraints.



```
TypeConversionMode        Allowed
◢ Result Set
ResultSet                 Full result set
◢ SQL Statement
ConnectionType            OLE DB
```

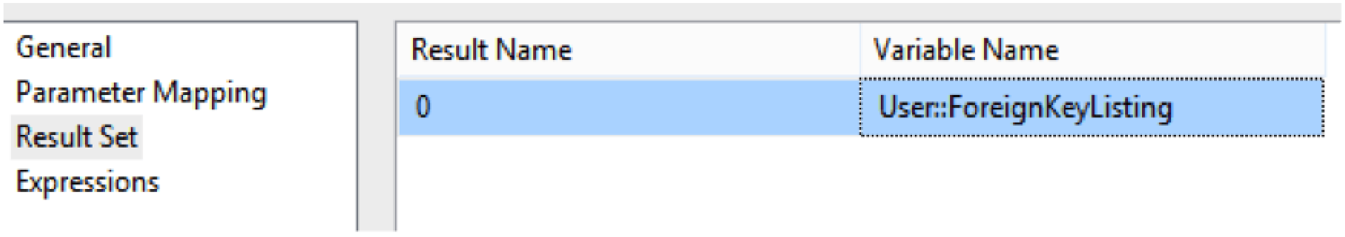
On the "Result Set" tab, Add and create the variable:


Add Variable

Specify the properties of the new variable.

Container:	000 - Preliminary Tasks
Name:	ForeignKeyListing
Namespace:	User
Value type:	Object
Value:	

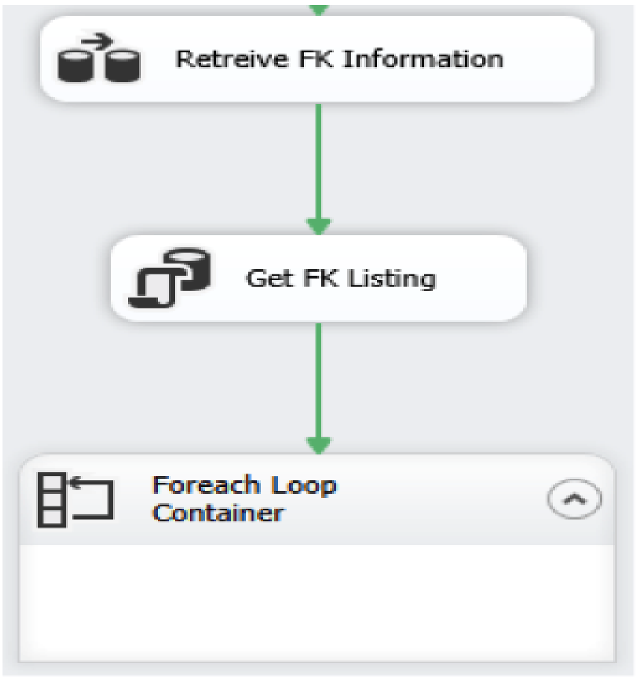
Read only [ ]



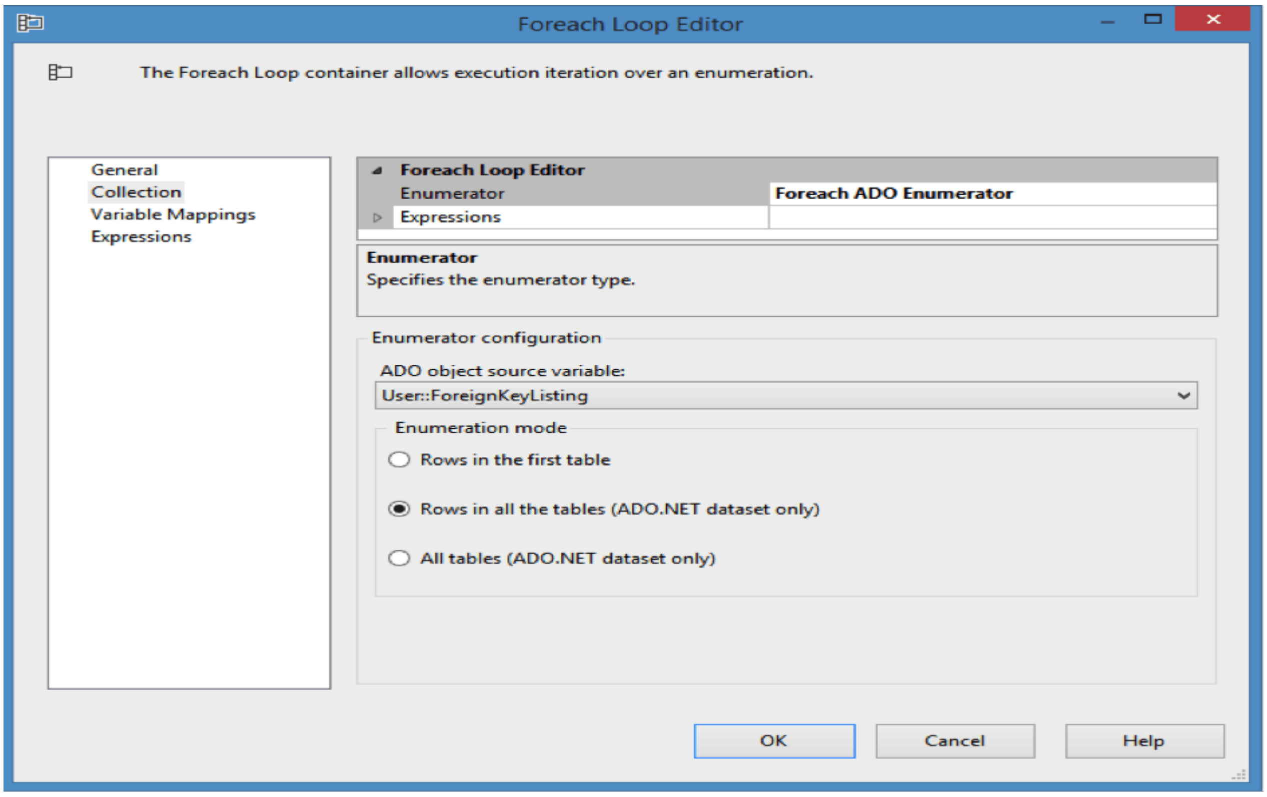
Result Name	Variable Name
0	User::ForeignKeyListing

We will use this object variable to iterate through in the next step.

Add a Foreach Loop Container to the package, and have the success flow from the **Get FK Listing** task connect to the Foreach Loop Container:



On the Foreach Loop Container editor, select the Collection tab and change the Enumerator to **Foreach ADO Enumerator** and in the **ADO object source variable:** select the newly created variable (**User:ForeignKeyListing**). Set the **Enumeration mode** to **Rows in all the tables (ADO.NET dataset only)**.



The Foreach Loop container allows execution iteration over an enumeration.

Foreach Loop Editor	
Enumerator	Foreach ADO Enumerator
Expressions	

Enumerator
Specifies the enumerator type.

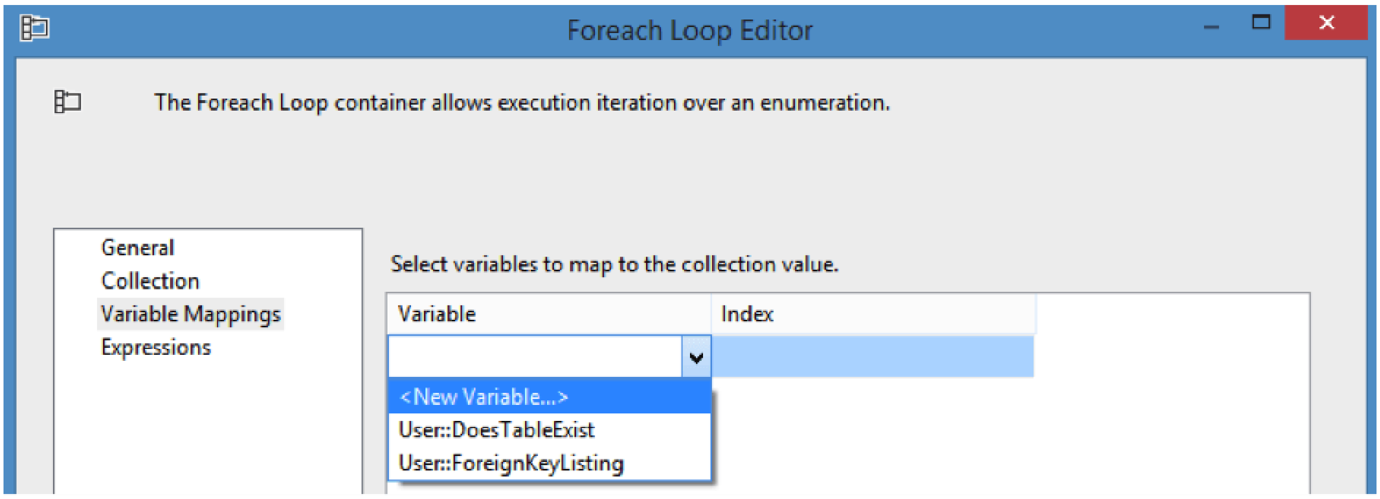
Enumerator configuration

ADO object source variable: User::ForeignKeyListing

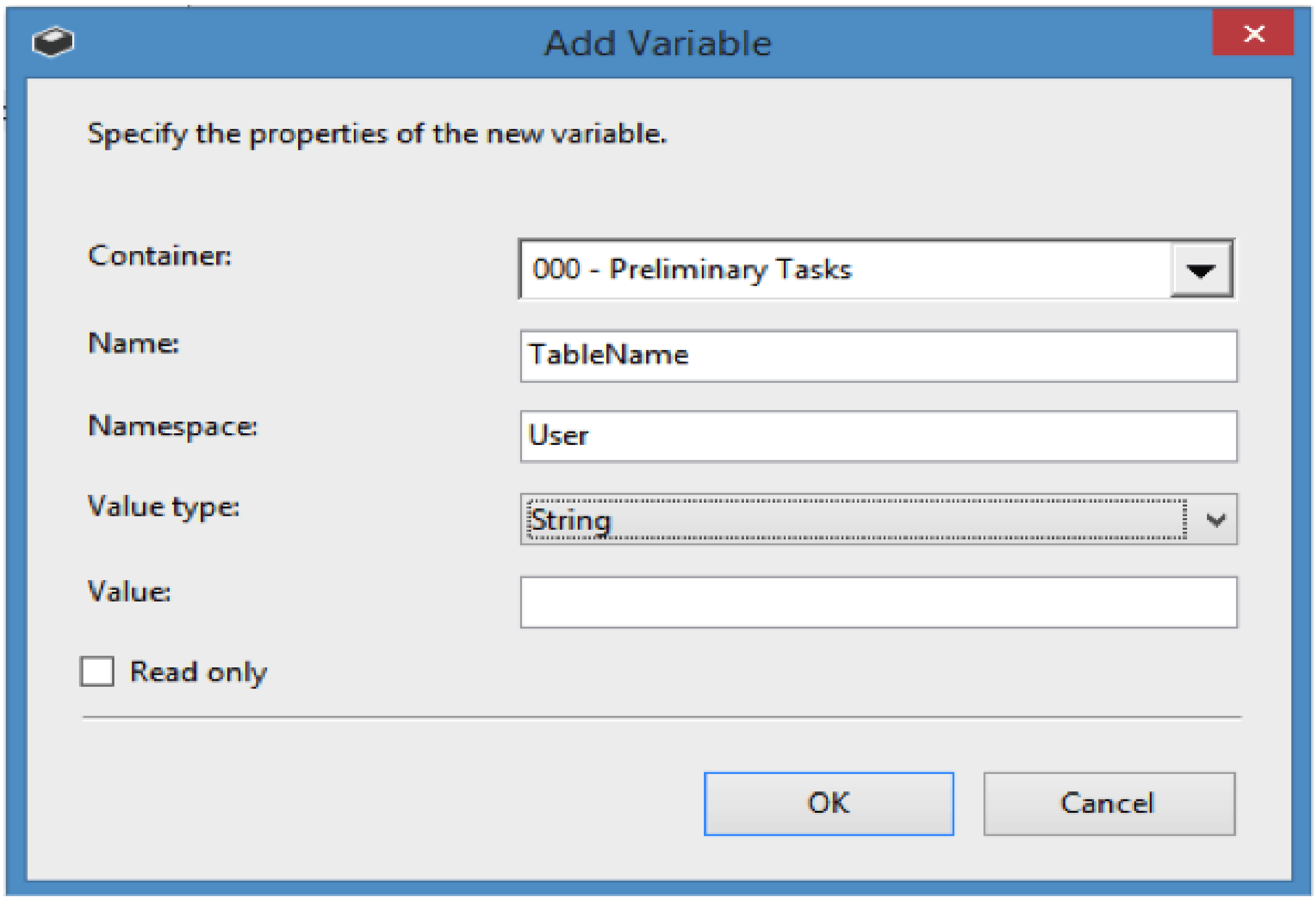
Enumeration mode
- ○ Rows in the first table
- ● Rows in all the tables (ADO.NET dataset only)
- ○ All tables (ADO.NET dataset only)

This will allow us to iterate through the return result set. Next we are going to assign our rows to 2 variables (we have 2 columns coming back from our query **table_name** and **foreign_key_name**).

Now select the Variable Mappings tab and in the **Variable** select the option **<New Variable...>**.



Create a new variable called **TableName** of type **String**.

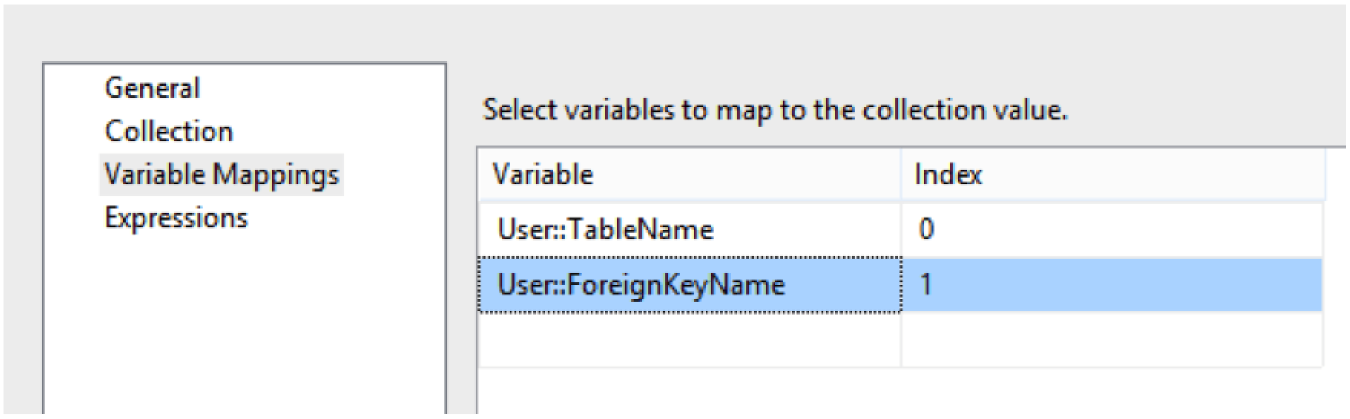


Container:	000 - Preliminary Tasks
Name:	TableName
Namespace:	User
Value type:	String
Value:	

☐ Read only

Click **OK**. When back in the **Foreach Loop Editor**, again select the **<New Variable...>** option, and create another string variable called **ForeignKeyName**.

Make sure **TableName** is assigned to **Index 0** (**table_name** – column 1 from our query) and **ForeignKeyName** is assigned to **Index 1** (**foreign_key_name** – column 2 from our query).

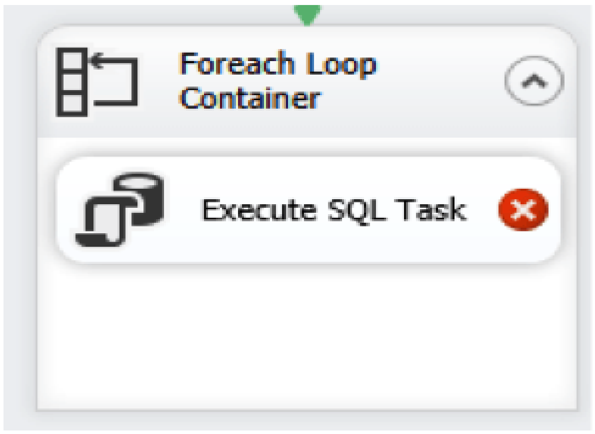


Variable	Index
User::TableName	0
User::ForeignKeyName	1

Ordering is extremely important, as the query to variable mapping is not bound to column names but ordinal position (this is why we avoid SELECT *).

Now we are set up to iterate through our loop and assign the query result set to our variables. Next we need to actually remove the foreign key references from the tables.

To do this we add an **Execute SQL Task** into the **Foreach Loop Container**.



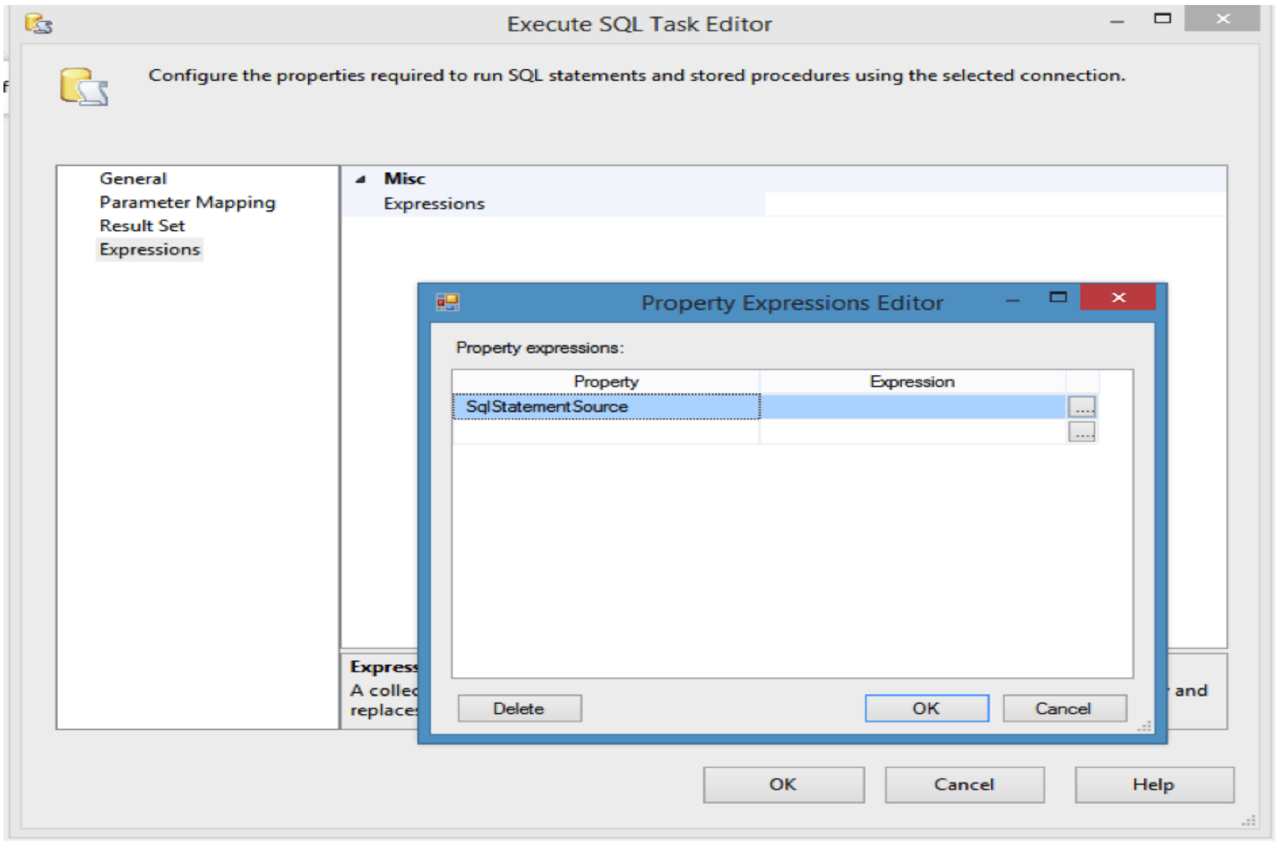
Note: This **must** be within the **Foreach Loop Container** or we will only remove the first FK (assuming your variable assignment is at package level), and not the subsequent ones.

On the **Execute SQL Task** set the connection to use the target database, and add the following SQL to the SQLStatement of the task:

```
ALTER TABLE [dbo].[FactInternetSales] DROP CONSTRAINT FK_FactInternetSales_DimCurrency
```

Note: This is just a place holder and we will modify the constraint as we iterate through the results.

On the **Expressions** tab click the ellipsis (...) button and select **SqlStatementSource** as the **Property** on the **Property Expressions Editor**.

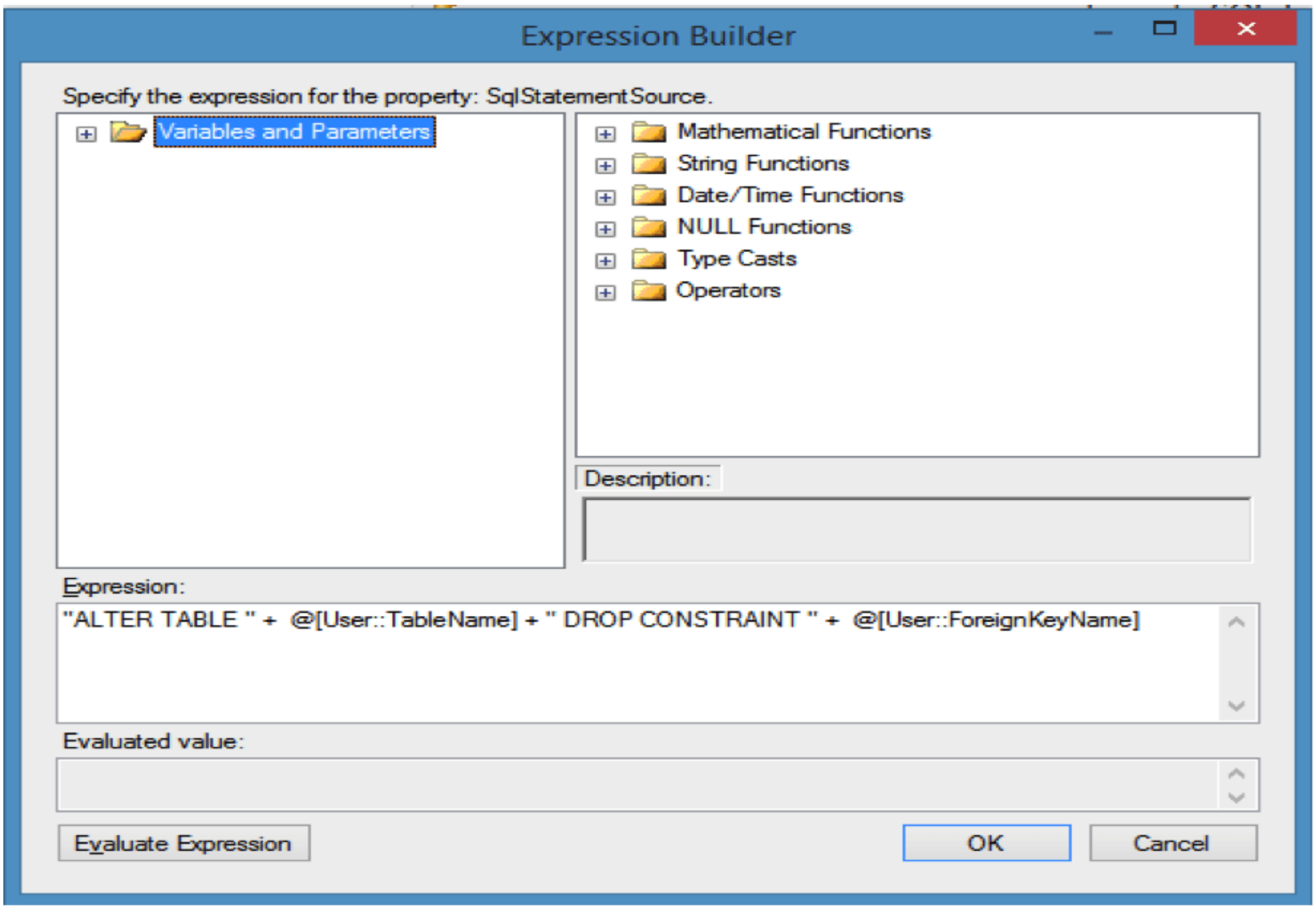


Then click the ellipsis button in the **Expression** section. This will allow us to create a custom string that replaces the original query in the **Execute SQL Task** we originally created.

Add the following code into the **Expression Builder** window:

"ALTER TABLE " + @[User::TableName] + " DROP CONSTRAINT " + @[User::ForeignKeyName]

Note: The double quotes (") dictate a string literal, and the plus (+) appends another string to this one, therefore we are now appending our table name to the ALTER TABLE statement.



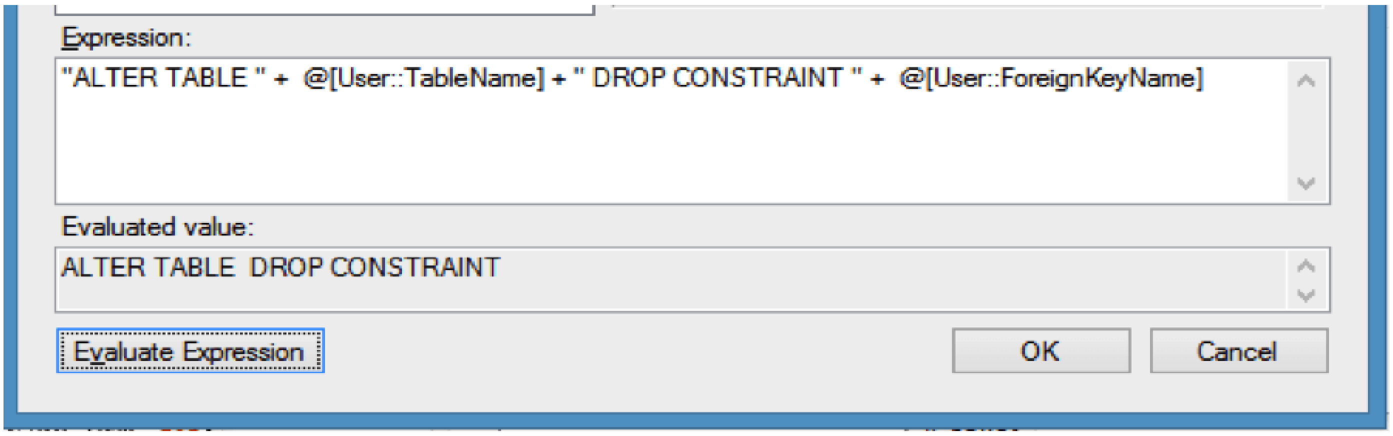
Specify the expression for the property: SqlStatementSource.

Expression:

```
"ALTER TABLE " + @[User::TableName] + " DROP CONSTRAINT " + @[User::ForeignKeyName]
```

Evaluated value:

If you click the Evaluate Expression button SSIS will attempt to determine the results of your code:



Expression:

```
"ALTER TABLE " + @[User::TableName] + " DROP CONSTRAINT " + @[User::ForeignKeyName]
```

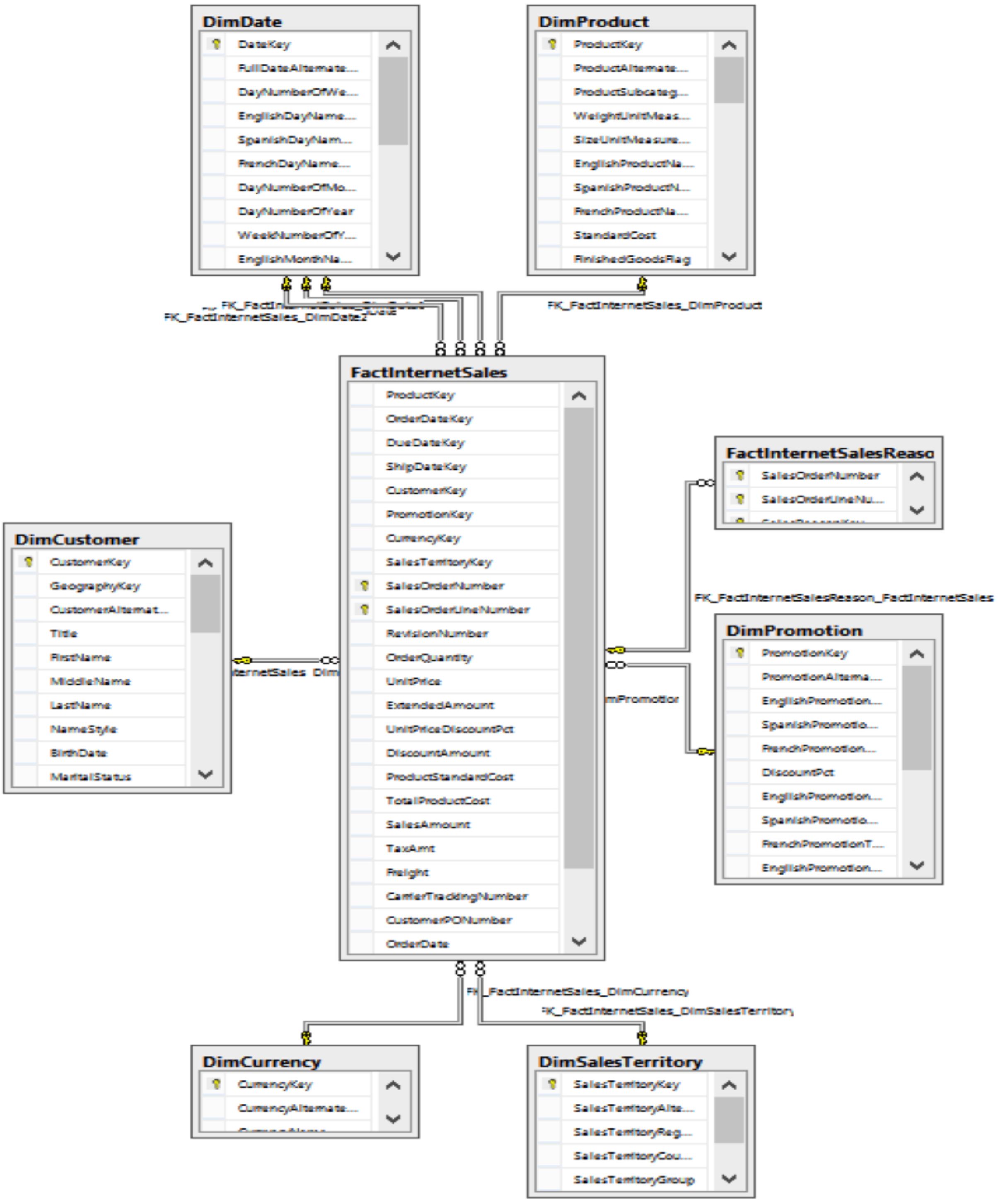
Evaluated value:

```
ALTER TABLE  DROP CONSTRAINT
```

We do not get any table or FK names in the evaluation because when we created the variables we didn’t assign a default value to them. If you modify your variable to have a default value this would evaluate with the variable values within the string.

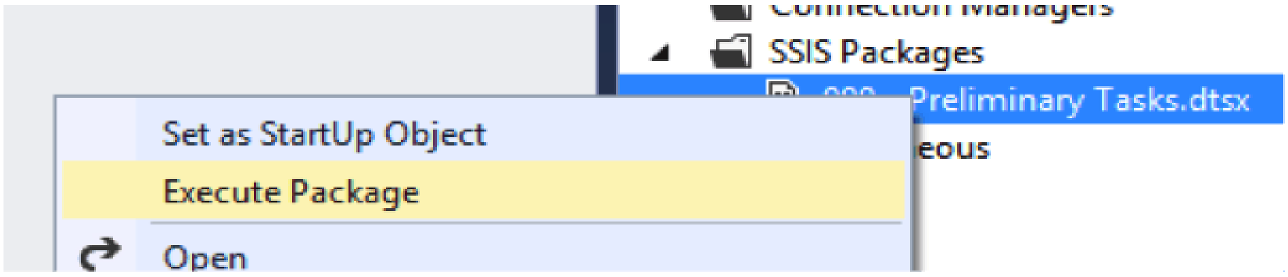
So now we should be ready to test to our results.

Let’s pop back to SQL and look at our Entity Relationship Diagram (ERD).

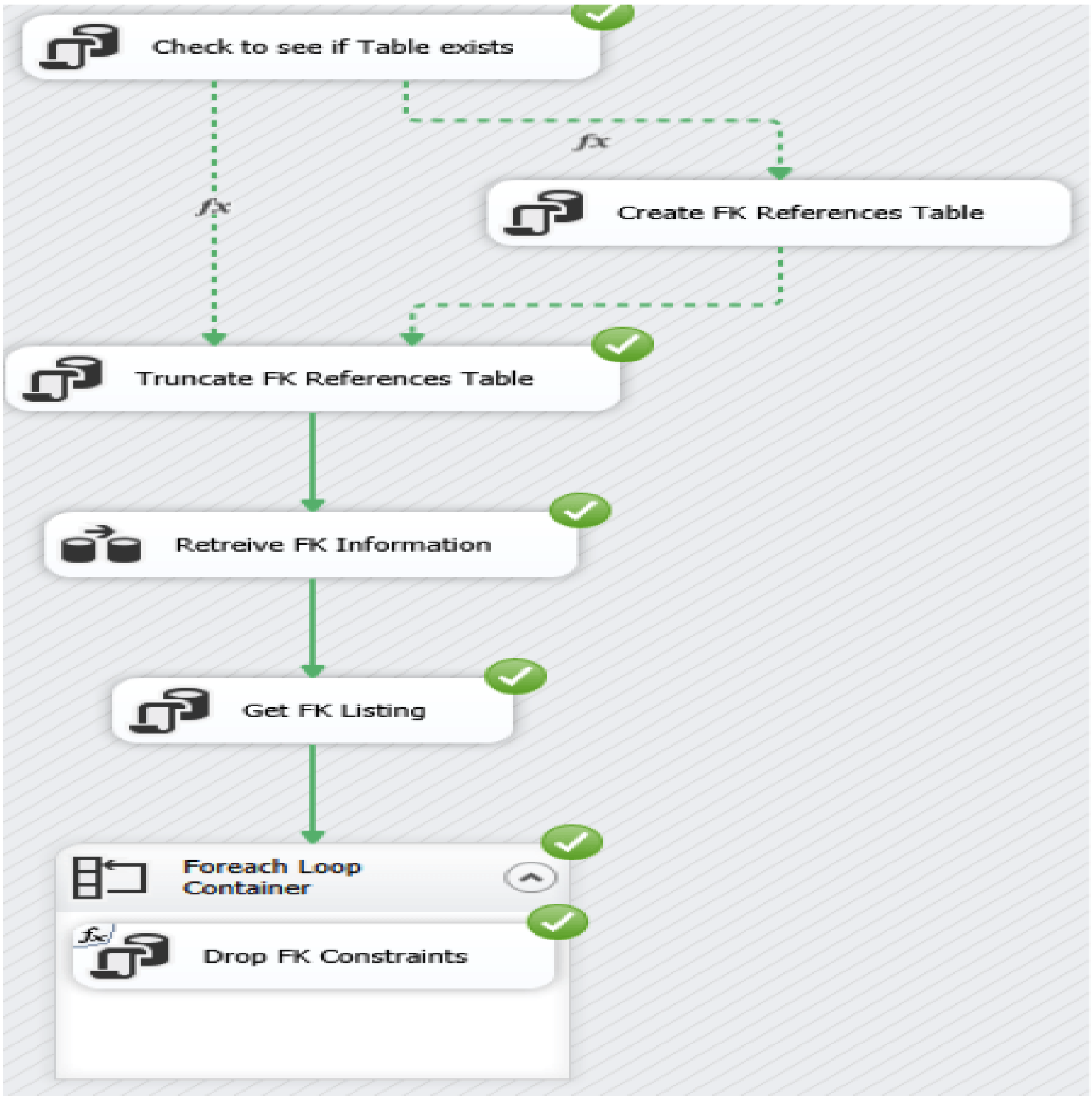


Close the diagram (no need to save it).

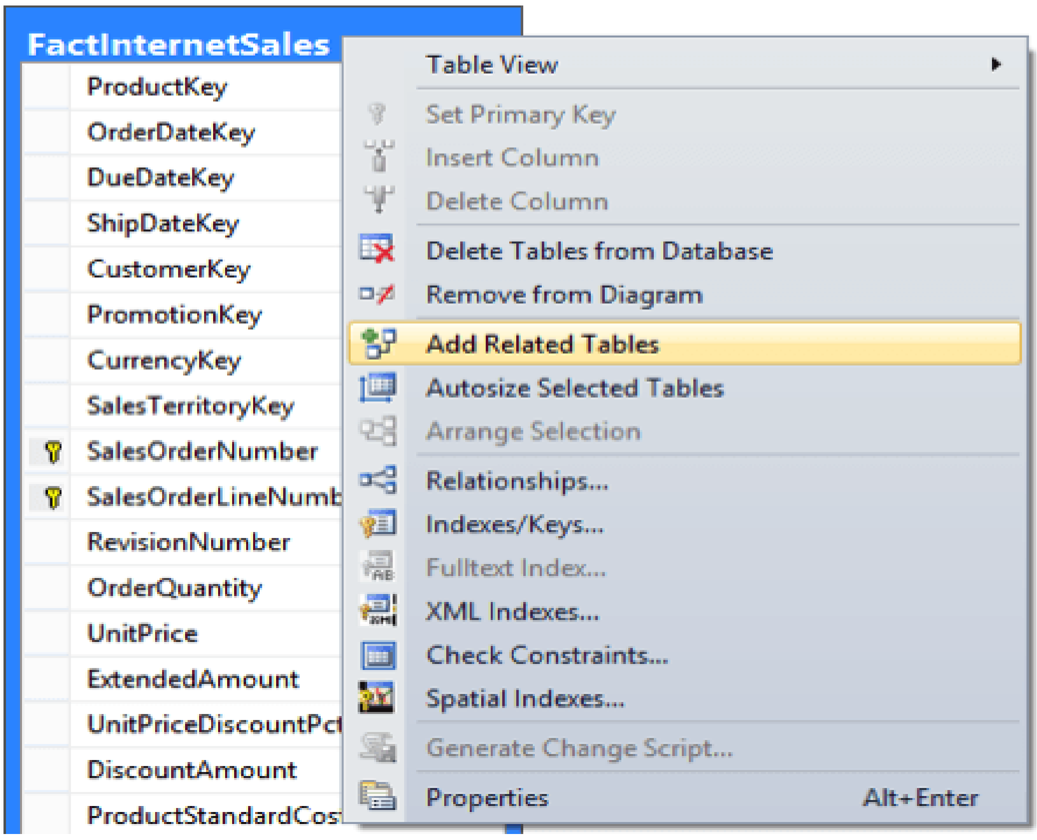
Now go back to your package and execute it:



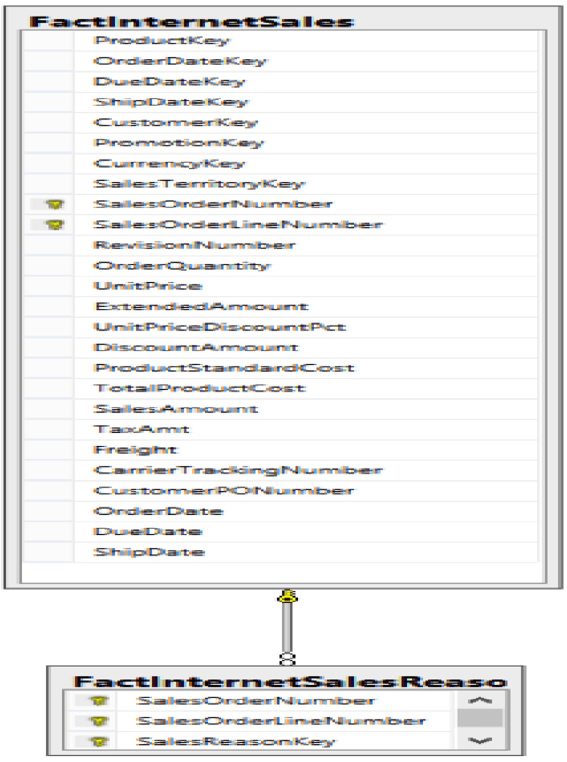
All going well it will execute successfully:


Check to see if Table exists
fx
fx
Create FK References Table
Truncate FK References Table
Retreive FK Information
Get FK Listing
Foreach Loop Container
Drop FK Constraints

Now go back to SQL and create a new ER diagram. Add the **FactInternetSales** table to the diagram, right click on the table and select **Add Related Tables**.



And – ta-da! - our diagram now looks like this:



Notice that the **FactInternetSalesReason** table is still referenced by **FactInternetSales**. This is because the **FactInternetSales** table is the parent and the **FactInternetSalesReason** is the child record. Therefore if we do this for the entire database we would need to perform the removal in the correct order, i.e. remove the constraints from **FactInternetSalesReason** and then any subsequent tables.

So far, we have collected the foreign key definitions from SQL, stored these in a table and then iterated through and removed the constraints. At this stage you would then run your full ETL and populate all your tables.

Next we are going to add the constraints back to our database. Read part two.

Rob Hawthorne is Theta's Head of Microsoft Solutions and Architecture, and has been recognized by Microsoft as a Most Valuable Professional (MVP) for his SQL Server expertise.