Ordered Loading of Databases Tables Using SSIS

By Joseph Ferry, 2013/04/30

Description of the Problem

Recently I was assigned the task of populating several databases that have identical schemas. The sources of the data would be several databases that had their own identical schemas, but this was different than the schema of the destination databases. There were about one hundred tables in each of the destination databases.

The order that tables are loaded is important. Tables which have foreign keys dependent upon primary or unique keys of other tables must be loaded after the tables upon which they are dependent. Virtually all well-designed databases have many such relationships.

SQL Server Integration Services (SSIS) is an ideal product for performing the above-described data migrations because once packages are created that correctly populate a single destination database from its associated source database, the packages should work without modification when configured to point to a different source database and a different destination database. Configuring the designation of source and destination databases can be implemented with XML configuration files. Using XML configuration files does not require the opening and modification of any component package.

Reconfiguration issues similar to this more frequently arise when moving packages among development, quality assurance, staging and production

Related Articles

FORUM Database schema
Database schema

FORUM Schema versus Database
Schema versus Database

FORUM How to sync database schema from a database to another?
sync database design, database schema

FORUM How to compare database table schema in SSIS package
how to bypass data transformation task if schema are same

FORUM Drop and Recreate constraints
How to drop and recreate constraints in a database

Tags
foreign keys (fk)
precedence constraints
ssis load order referential integrity

Write For Us Find out more

environments. For further information about configuring packages without modifying them, see [Defining a Configuration Approach for Integration Services Packages](#).

Types of Constraints

This article may introduce a bit of confusion to some readers because the word "constraint" is associated with two concepts. First, we have referential integrity constraints, also called foreign key constraints. These types of constraints exist in a database. They are represented in database diagrams as lines going between one table and another table, or, less often, between a table and itself. They can be created graphically in SSMS, or with the T-SQL "`ALTER TABLE`" statement. If you are unclear about what referential integrity means or its importance to creating and maintaining a database, see Data Integrity

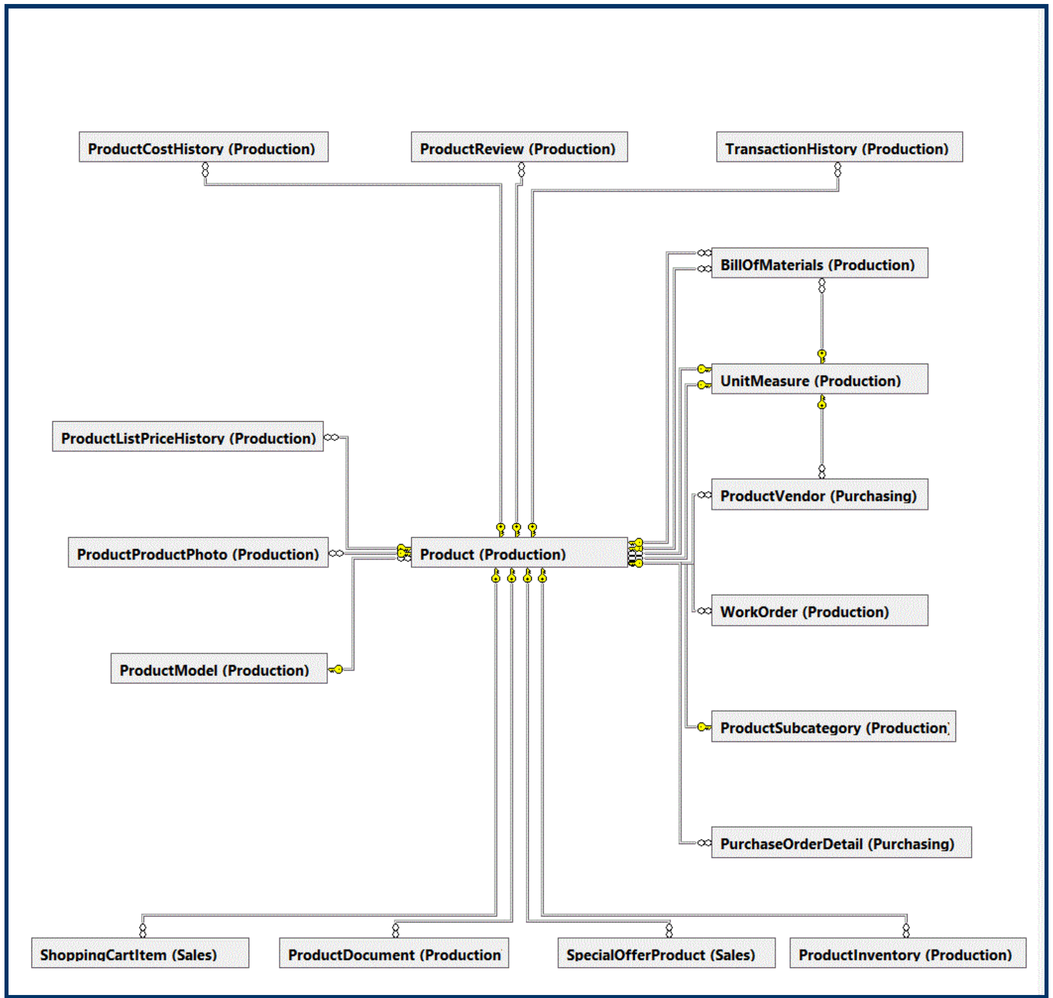
The second type of constraint is a precedence constraint in SSIS. These exist only on the Control Flow designer used for designing SSIS packages. While you will also see lines between objects on the Data Flow designer surface, connecting the objects you see there, precedence constraints can only exist between tasks in the Control Flow Designer. The Data Flow designer does not contain tasks, but contain data sources, transforms and data destinations. The lines between objects on a Data Flow designer surface represent a data pipeline where data flows between components. Data does not flow between tasks on the Control Flow surface. For more information, see Precedence Constraints.

Both types of constraints are referenced throughout this article. In the section of the article titled *Third Approach: Mimicking Foreign Key Constraints in the Design of Control Flow Precedence Constraints*, you will see that they are logically connected.

The AdventureWorks2012 Database

The AdventureWorks database will be the basis for demonstrations in this article and is available as a download from [Adventure Works for SQL Server 2012](#). This sample database was designed to permit a demonstration of virtually every feature of SQL Server 2012 and therefore has more numerous and more complex relationships than I have encountered in actual production databases with which I have previously worked. For simplicity in demonstrating core concepts in this article, I will use only a few tables from the AdventureWorks2012 database. If all of the tables were included in my examples, the complex relationships would have so many consequences that it would be more difficult to see the points that I wish to make.

The relationships between tables in most databases are complex and are likely to be confusing. Determining the order in which to load tables while permitting enforcement and checking of referential integrity by the destination database is usually not trivial. A table may be dependent upon one or more other tables while, at the same time, there are one or more tables dependent upon it. These relationships constrain the order in which tables may be successfully loaded. Figure 1 demonstrates this complexity by showing that within the AdventureWorks2012 database, the Production.Product table is dependent upon four tables and has fourteen tables dependent upon it!


ProductCostHistory (Production)
ProductReview (Production)
TransactionHistory (Production)
BillOfMaterials (Production)
UnitMeasure (Production)
ProductListPriceHistory (Production)
ProductVendor (Purchasing)
ProductProductPhoto (Production)
Product (Production)
WorkOrder (Production)
ProductModel (Production)
ProductSubcategory (Production)
PurchaseOrderDetail (Purchasing)
ShoppingCartItem (Sales)
ProductDocument (Production
SpecialOfferProduct (Sales)
ProductInventory (Production)

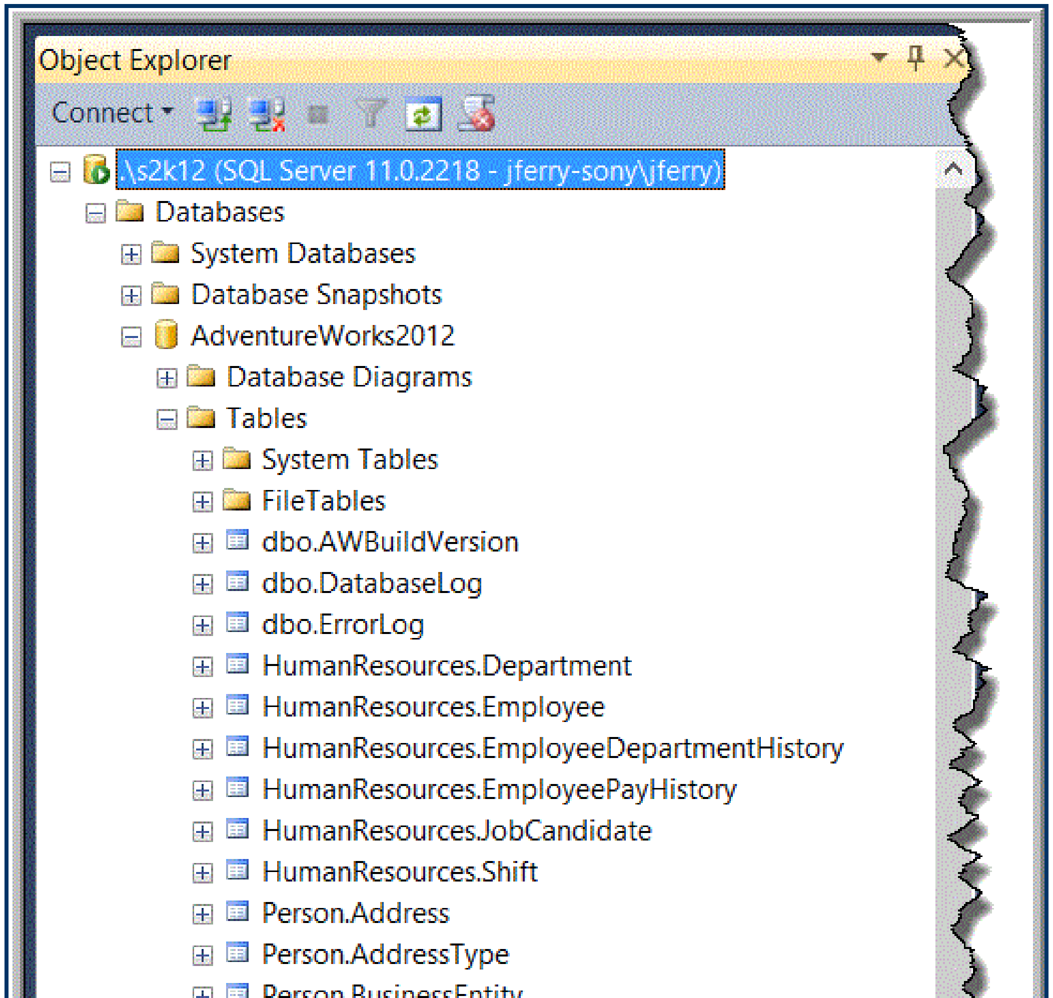
Zoom in | Open in new window

Figure 1. Database diagram showing multiple tables upon which the Production.Product table depends and the multiple tables that are dependent upon it.

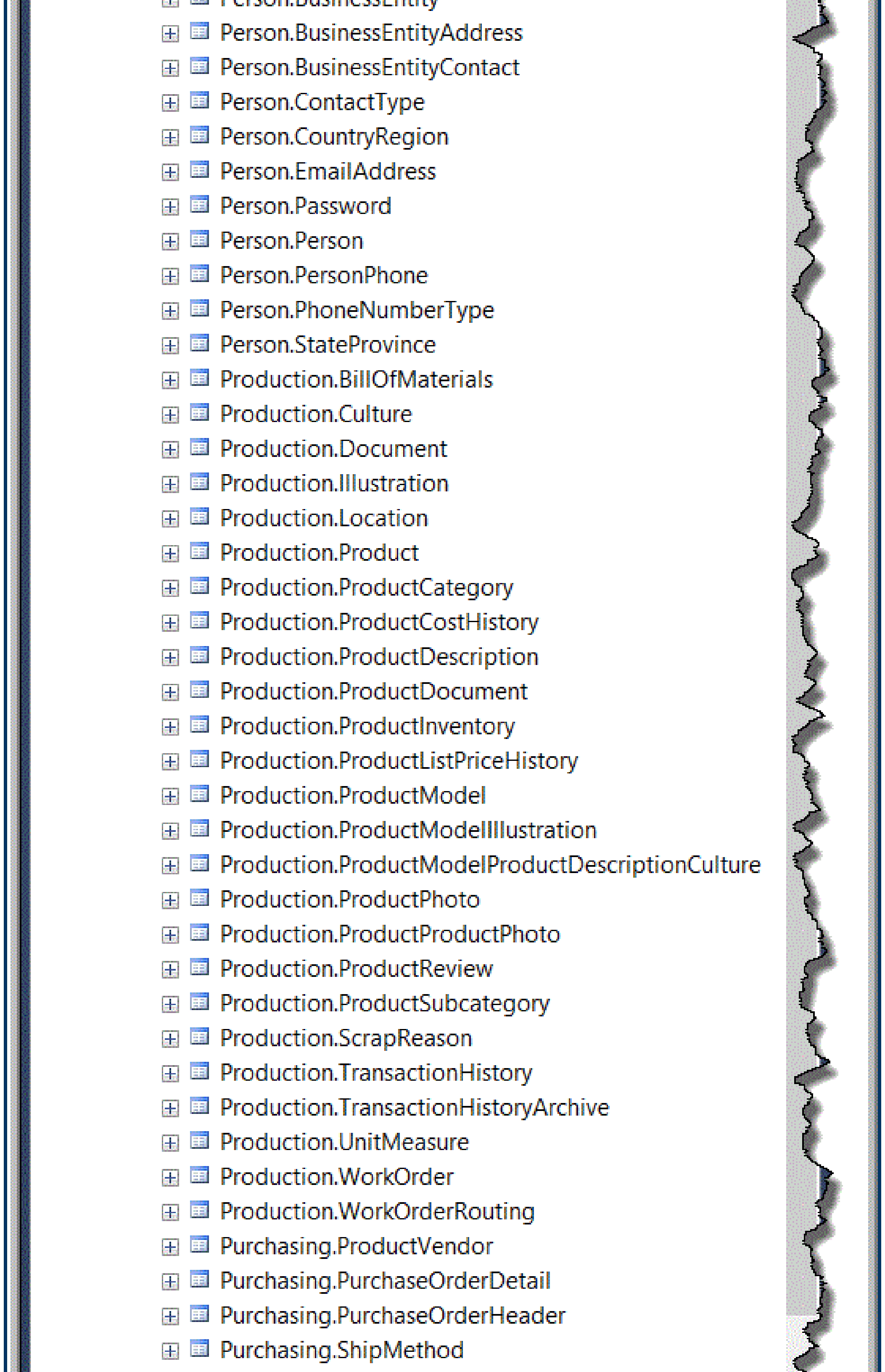
User-Defined Schema Objects

A schema object is a container within a specific database in which we can group other database objects. The AdventureWorks2012 database employs user-defined schema objects to group closely-related tables together in a logical way. By default, the "`CREATE DATABASE`" statement provides a new database with only one schema object for use with user-defined objects, named "dbo." Any time we create a new SQL Server database table, that table is placed by default into the "dbo" schema. This can result in a large list of tables that have no obvious logical associations to each other in one schema. Experience shows that the more tables there are in a database, the more helpful it becomes to group tables within user-defined schema objects. To learn how to create a new schema object, see Create a Database Schema.

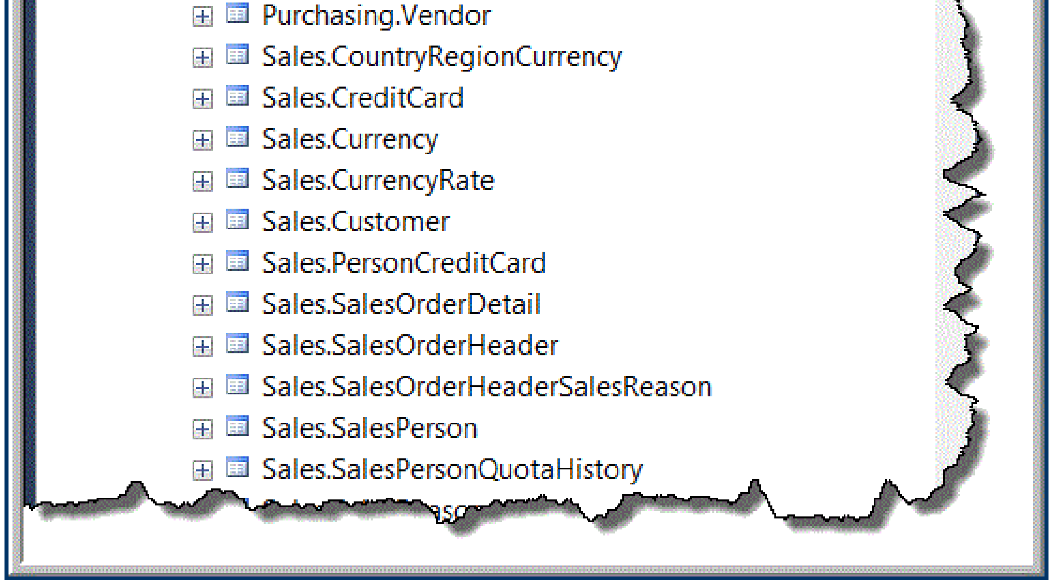
In a tree view of a database in SQL Server Management Studio ("SSMS"), tables are alphabetized first by schema name and then by table name. See Figure 2.



```
.\s2k12 (SQL Server 11.0.2218 - jferry-sony\jferry)
  Databases
    System Databases
    Database Snapshots
    AdventureWorks2012
      Database Diagrams
      Tables
        System Tables
        FileTables
        dbo.AWBuildVersion
        dbo.DatabaseLog
        dbo.ErrorLog
        HumanResources.Department
        HumanResources.Employee
        HumanResources.EmployeeDepartmentHistory
        HumanResources.EmployeePayHistory
        HumanResources.JobCandidate
        HumanResources.Shift
        Person.Address
        Person.AddressType
```



Person.BusinessEntity
Person.BusinessEntityAddress
Person.BusinessEntityContact
Person.ContactType
Person.CountryRegion
Person.EmailAddress
Person.Password
Person.Person
Person.PersonPhone
Person.PhoneNumberType
Person.StateProvince
Production.BillOfMaterials
Production.Culture
Production.Document
Production.Illustration
Production.Location
Production.Product
Production.ProductCategory
Production.ProductCostHistory
Production.ProductDescription
Production.ProductDocument
Production.ProductInventory
Production.ProductListPriceHistory
Production.ProductModel
Production.ProductModelIllustration
Production.ProductModelProductDescriptionCulture
Production.ProductPhoto
Production.ProductProductPhoto
Production.ProductReview
Production.ProductSubcategory
Production.ScrapReason
Production.TransactionHistory
Production.TransactionHistoryArchive
Production.UnitMeasure
Production.WorkOrder
Production.WorkOrderRouting
Purchasing.ProductVendor
Purchasing.PurchaseOrderDetail
Purchasing.PurchaseOrderHeader
Purchasing.ShipMethod



Zoom in | Open in new window

Figure 2. Treeview within SSMS showing tables in a database alphabetized first by schema name and then by table name.

When viewing tables placed on an AdventureWorks2012 database diagram, we find user-defined schemas named HumanResources, Person, Production, Purchasing and Sales. In Figure 3 we see that the associated user-defined schema name appears in parentheses following each table name. Tables that are part of the “dbo” schema do not show “dbo” in parentheses following their table names.

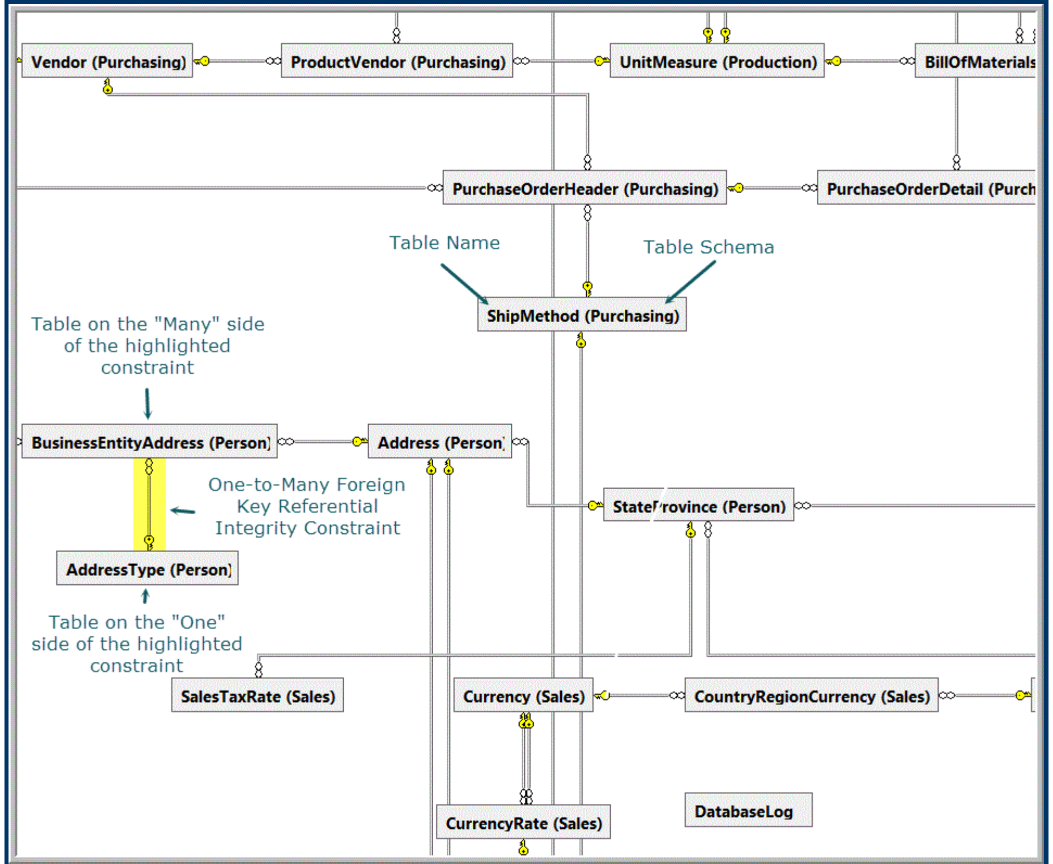
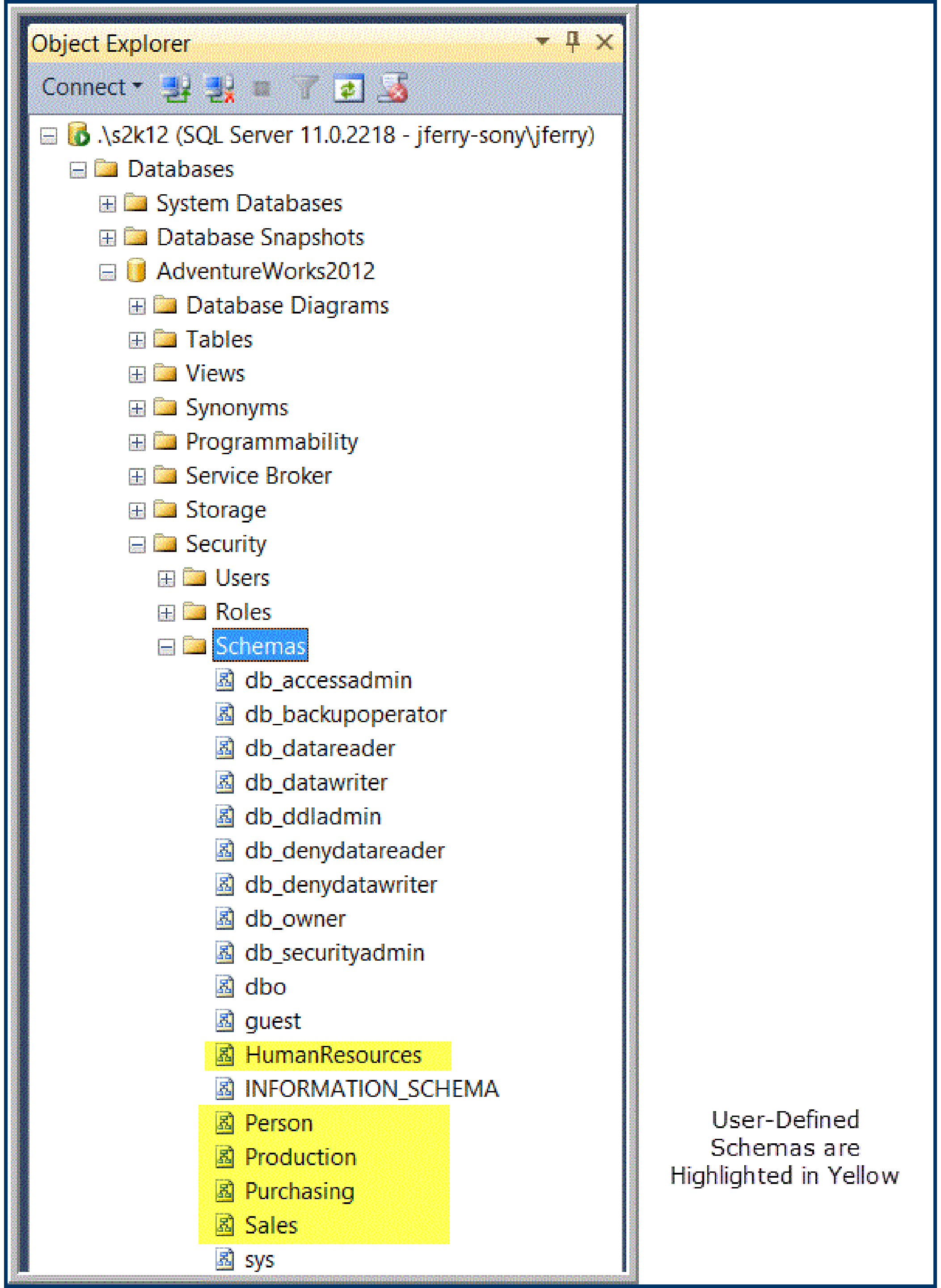

Vendor (Purchasing)
ProductVendor (Purchasing)
UnitMeasure (Production)
BillOfMaterials
PurchaseOrderHeader (Purchasing)
PurchaseOrderDetail (Purch
Table Name
Table Schema
ShipMethod (Purchasing)
Table on the "Many" side of the highlighted constraint
BusinessEntityAddress (Person)
Address (Person)
One-to-Many Foreign Key Referential Integrity Constraint
StateProvince (Person)
AddressType (Person)
Table on the "One" side of the highlighted constraint
SalesTaxRate (Sales)
Currency (Sales)
CountryRegionCurrency (Sales)
CurrencyRate (Sales)
DatabaseLog
Zoom in | Open in new window

Figure 3. Partial database diagram showing table names followed by schema name in parentheses (except if the table is in the dbo schema).

Figure 4 shows *all* the schema objects that belong in the AdventureWorks2012 database. The dbo, guest, and sys schema objects as well as the schema objects with names beginning with “db_” were created by the system and exist in every database before any user-defined objects are created within it. For further information on creating and using schemas in the AdventureWorks2012 database, see Schemas in AdventureWorks.



Object Explorer

Connect

.\s2k12 (SQL Server 11.0.2218 - jferry-sony\jferry)
- Databases
  - System Databases
  - Database Snapshots
  - AdventureWorks2012
    - Database Diagrams
    - Tables
    - Views
    - Synonyms
    - Programmability
    - Service Broker
    - Storage
    - Security
      - Users
      - Roles
      - Schemas
        - db_accessadmin
        - db_backupoperator
        - db_datareader
        - db_datawriter
        - db_ddladmin
        - db_denydatareader
        - db_denydatawriter
        - db_owner
        - db_securityadmin
        - dbo
        - guest
        - HumanResources
        - INFORMATION_SCHEMA
        - Person
        - Production
        - Purchasing
        - Sales
        - sys

User-Defined Schemas are Highlighted in Yellow

Zoom in | Open in new window

Figure 4. Listing of all the schema objects in a database under the Security node of a database.

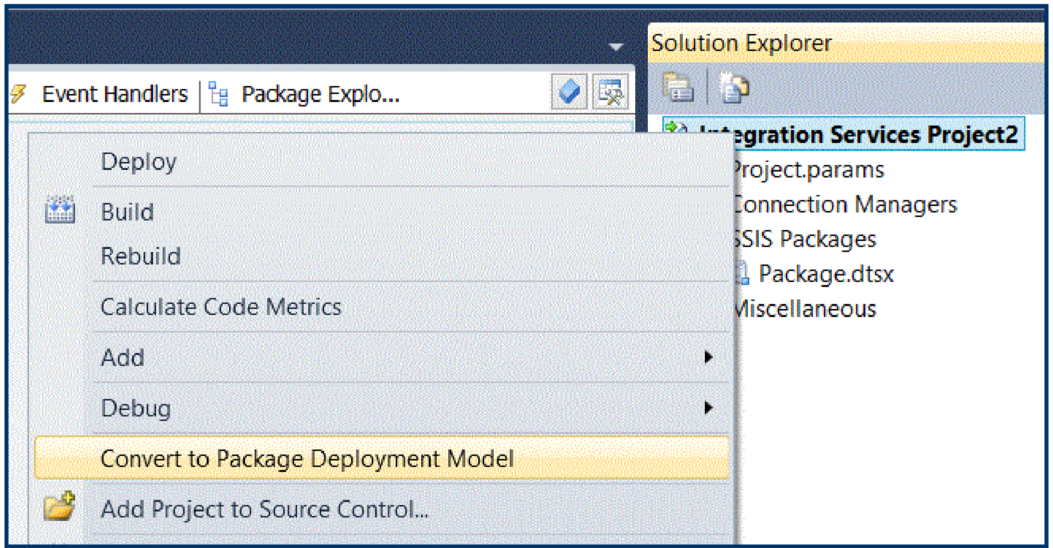
If you wish to reference a table that has a schema other than “dbo” within Transact SQL code, you must precede the table name with the schema name and separate them with a dot (“.”). In general, where you otherwise might expect to write “SELECT * FROM Product,” you will instead write “SELECT * FROM Production.Product.”

In Figure 4 you may notice that the schema names appear in SSMS under the “Security” node associated with each separate database, and the schema names are not associated with the entire SQL Server instance. Database security can often be more easily implemented through schema objects rather than by setting properties of each object separately. There is a difference between the use of schemas to manage security in SQL Server 2005 and SQL Server 2008. Discussing security is beyond the scope of this article. For more information, see User-Schema Separation.

My experience suggests that schema objects should be used to group tables closely-related to each other and not be used to provide a distinction between two tables having the same name. To avoid confusion, table names should be unique across an entire database.

Differences Between Versions of BIDS and SQL Server 2012 Data Tools

The figures used in this article to show how objects appear in SSMS and Business Intelligence Development Studio (“BIDS”) display the 2012 version of these tools. In the 2012 toolset, BIDS has been renamed SQL Server Data Tools (“SSDT”). For the purpose of using these images and viewing the images, the most significant differences between the 2012 versions of these tools and previous version are consequences of the new default deployment model in 2012 which permits deployment of *projects* rather than the deployment of *packages*. While it may have appeared that one was deploying a project with the Package Deployment Wizard, one was actually deploying multiple packages at the same time where the dependencies were implemented by extra code, property settings and configurations implemented by the developer. Previously, deployment had no concept of a *project object*. In the new 2012 model, deployment recognizes the *project object*, which can contain project-level variables (called "parameters") that are visible to each of the component packages without further work by the developer. The tree view in the project explorer of 2012 Data Tools has different nodes than the tree view of BIDS projects in previous versions. Despite these differences, the concepts discussed herein work in any version of these tools from SQL Server 2005 to the present. If you wish to make an SSIS project using SSDT 2012 appear precisely as it does in this article, then when you create a new project in 2012 you should immediately right click on the project name and select “Convert to Package Deployment Model“. See Figure 5. For more information about the 2012 deployment model, see Deployment of Projects and Packages. For more information about changes to SSIS implemented in SQL Server 2012, see What's New (Integration Services).



Zoom in | Open in new window

Figure 5. Context menu showing how to convert a new SSIS 2012 project to the legacy deployment model.

Contents of Component Packages

When I started developing my SSIS project in BIDS, I was more or less floundering around trying to get the packages to load in order in any way I could. During this process of experimentation and discovery, I was irritated to discover that I was modifying many packages over and over again. Then I realized that it would make my development and modification procedure simpler to have one destination table per package. After that, I could experiment with different load orders by modifying a single master package that dictated the order of execution of each table-level package.

I called this master package “_Main” in a file named “_Main.dtsx”. By starting the file name with an underscore, the _Main.dtsx package file can be made to appear at the beginning of the list of packages in Solution Explorer within a BIDS or SSDT project. By default, packages appear in these development environments in the order that you create them. To make them appear in alphabetical order, you must right-click on the “Packages” node in Solution Explorer and select “Sort by Name”. See Figure 6. After creating the _Main package, I placed one “Execute Package” task per destination table in the _Main package and never had to edit any of the table-level packages again (unless I wished to change the logic within a package for some reason unrelated to load order).

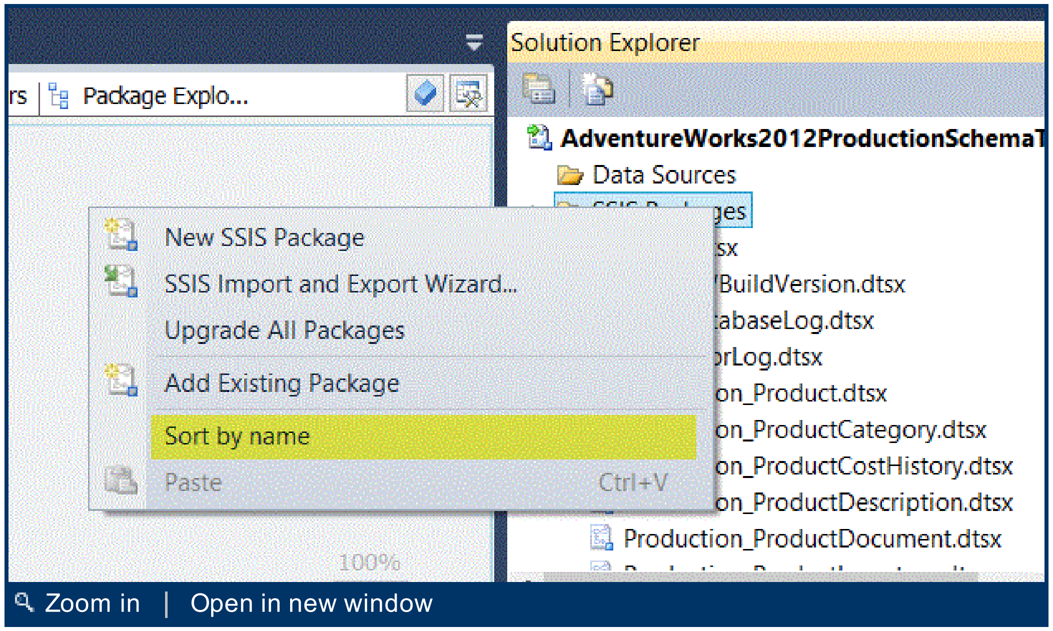
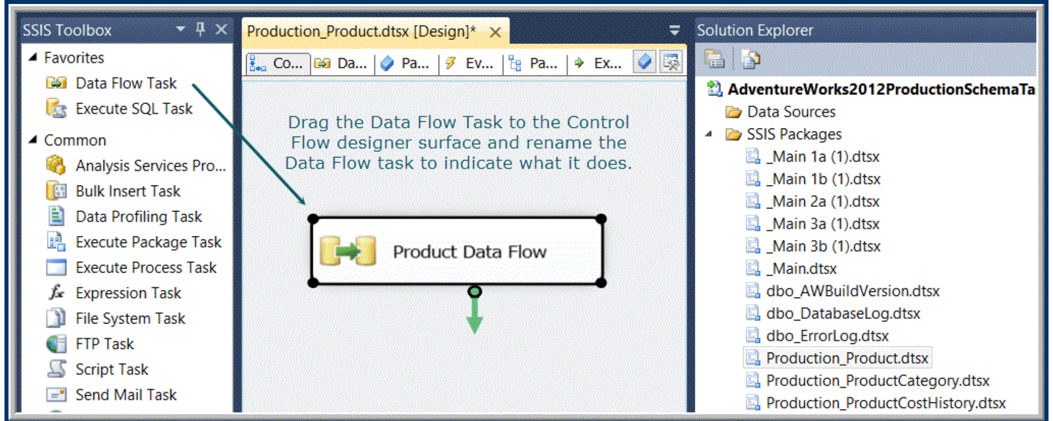


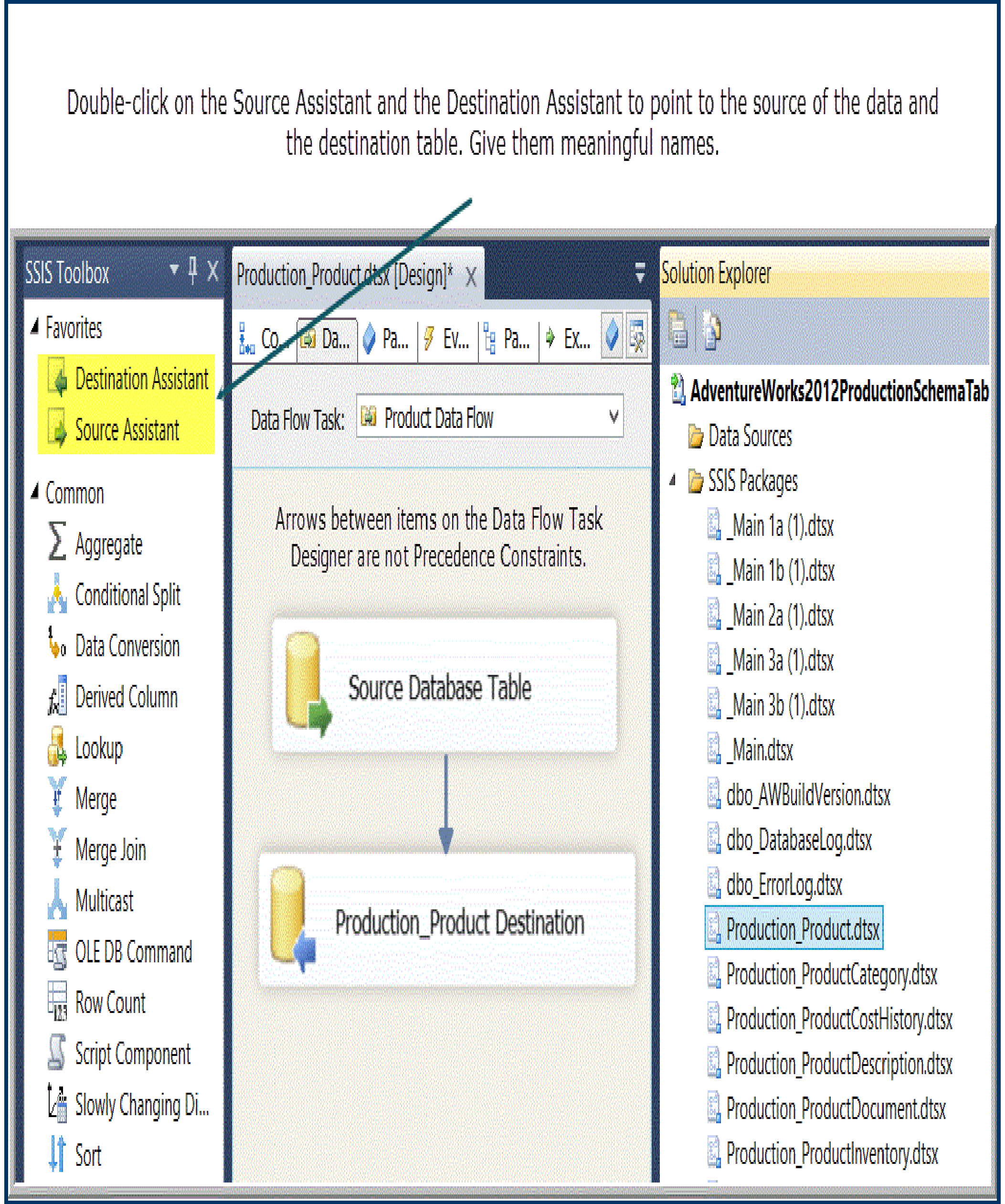
Figure 6. Context menu showing how to sort packages alphabetically in BIDS and SSDT.

If there wasn’t any complex transformation occurring between a source and destination table (and this was often the case), then the creation of such a table-level package was simple, requiring only one task, a data flow task (see Figure 7), which in turn, contained one OLEDB Source and one OLEDB Destination (see Figure 8, showing the data flow designer view of the data flow task in Figure 7.).



Zoom in | Open in new window

Figure 7. The control flow designer of a simple package.


Double-click on the Source Assistant and the Destination Assistant to point to the source of the data and the destination table. Give them meaningful names.

Arrows between items on the Data Flow Task Designer are not Precedence Constraints.

Zoom in | Open in new window

Figure 8. The data flow designer of a simple package.

Naming Conventions and Logging

I gave each object, whether a task, transform, data source or data destination, a descriptive name that is unique across the entire project. Because each object is in the context of a specific package or data task, this would seem to be unnecessary; however, the name we give an object will be the text that appears under the “SourceName” column of the msdb.sys.sysssislog table (or in the parallel field of some other logging provider). This is true for SQL Server 2005, 2008 and 2008 R2 as well as SQL Server 2012 running packages in the legacy mode. In logs we usually see that a package, task or transform is referenced by its “SourceID”, which is a GUID, and this is not as helpful as using the “SourceName” column which has been populated with a descriptive, unique name.

Because table names are unique across all schema objects in the AdventureWorks2012 database, I arbitrarily chose to omit a reference to schema names associated with tables when naming packages, tasks, data sources and destinations and transforms. It is important to label objects with descriptive names even if they will not appear in a log reflecting a successful execution. You will probably be logging the “OnError” event of every package, task and transform because the logging of the OnError event will contain helpful information in the “message” field. Quickly identifying the atomic component by which generated the error is facilitated by using unique descriptive names. See Implementing Logging in Packages for more information about logging events in SSIS packages.

Finding an Approach to Loading Tables in Order

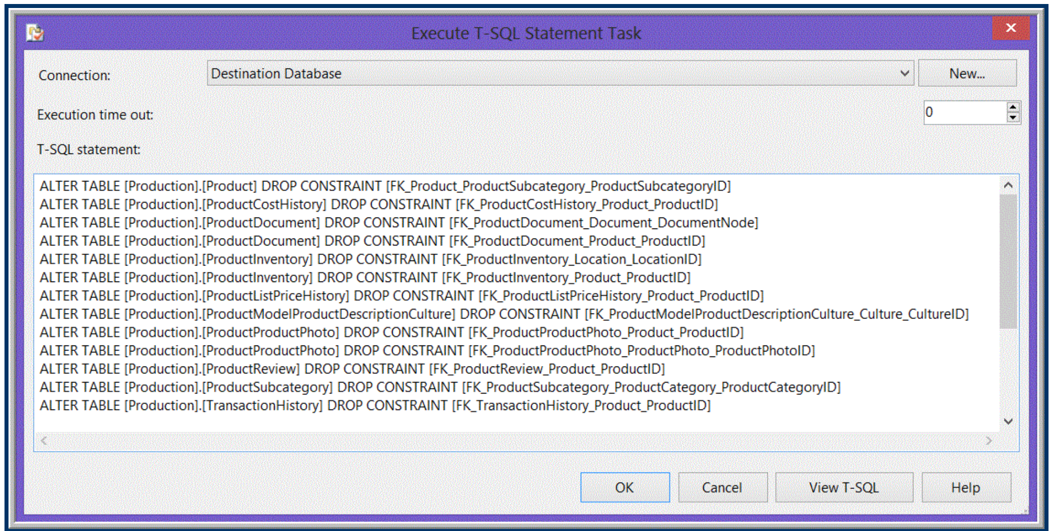
I found some articles on the web regarding the load order of tables, but they seemed to show examples where the relationships between tables were very simple, i.e., every table was dependent upon at most one table and had at most one table dependent upon it. That did not describe my situation, and the solutions described in those articles did not work.

First Approach: Dropping and Recreating Foreign Key Constraints

A colleague suggested that I drop all referential integrity constraints in a destination database, populate the tables in any order, and then recreate the constraints. This this works best when you already know that the data conforms to the data integrity requirements enforced by the target database. The recreation of constraints will fail if the data does not satisfy the constraints, which is exactly what I would want to happen if I used this strategy.

Figure 9 shows an example of code that would appear in the “T-SQL Statement Property” of the “Execute T-SQL Statement Task” to drop constraints, and

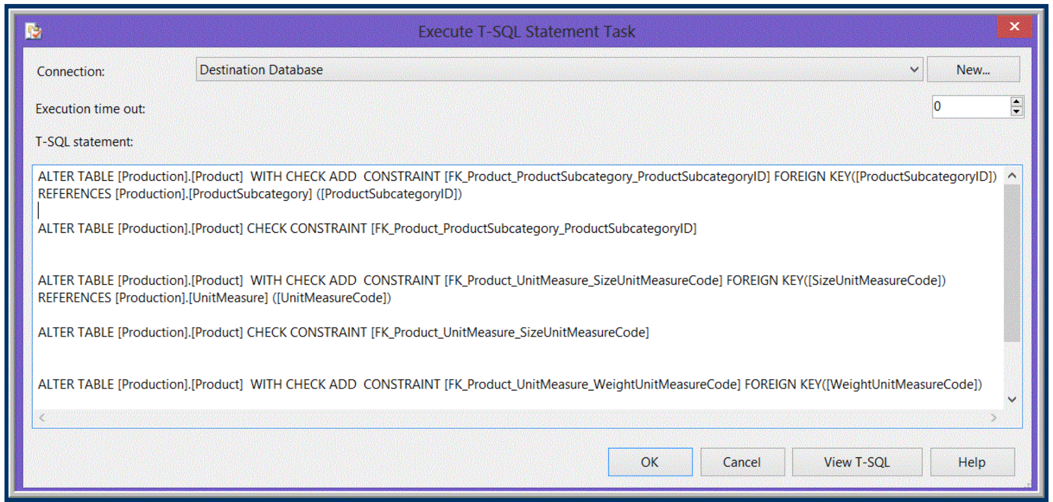
Figure 10 is an example of code that would appear in the “T-SQL Statement Property” of the “Execute T-SQL Statement Task” to recreate the constraints.



```
ALTER TABLE [Production].[Product] DROP CONSTRAINT [FK_Product_ProductSubcategory_ProductSubcategoryID]
ALTER TABLE [Production].[ProductCostHistory] DROP CONSTRAINT [FK_ProductCostHistory_Product_ProductID]
ALTER TABLE [Production].[ProductDocument] DROP CONSTRAINT [FK_ProductDocument_Document_DocumentNode]
ALTER TABLE [Production].[ProductDocument] DROP CONSTRAINT [FK_ProductDocument_Product_ProductID]
ALTER TABLE [Production].[ProductInventory] DROP CONSTRAINT [FK_ProductInventory_Location_LocationID]
ALTER TABLE [Production].[ProductInventory] DROP CONSTRAINT [FK_ProductInventory_Product_ProductID]
ALTER TABLE [Production].[ProductListPriceHistory] DROP CONSTRAINT [FK_ProductListPriceHistory_Product_ProductID]
ALTER TABLE [Production].[ProductModelProductDescriptionCulture] DROP CONSTRAINT [FK_ProductModelProductDescriptionCulture_Culture_CultureID]
ALTER TABLE [Production].[ProductProductPhoto] DROP CONSTRAINT [FK_ProductProductPhoto_Product_ProductID]
ALTER TABLE [Production].[ProductProductPhoto] DROP CONSTRAINT [FK_ProductProductPhoto_ProductPhoto_ProductPhotoID]
ALTER TABLE [Production].[ProductReview] DROP CONSTRAINT [FK_ProductReview_Product_ProductID]
ALTER TABLE [Production].[ProductSubcategory] DROP CONSTRAINT [FK_ProductSubcategory_ProductCategory_ProductCategoryID]
ALTER TABLE [Production].[TransactionHistory] DROP CONSTRAINT [FK_TransactionHistory_Product_ProductID]
```

Zoom in | Open in new window

Figure 9. Example of code in an Execute T-SQL Statement Task dropping foreign key constraints.



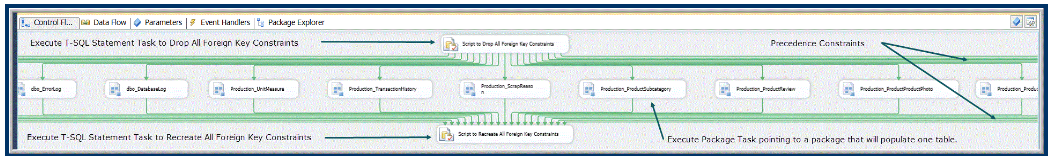
```
ALTER TABLE [Production].[Product]  WITH CHECK ADD  CONSTRAINT [FK_Product_ProductSubcategory_ProductSubcategoryID] FOREIGN KEY([ProductSubcategoryID])
REFERENCES [Production].[ProductSubcategory] ([ProductSubcategoryID])

ALTER TABLE [Production].[Product] CHECK CONSTRAINT [FK_Product_ProductSubcategory_ProductSubcategoryID]


ALTER TABLE [Production].[Product]  WITH CHECK ADD  CONSTRAINT [FK_Product_UnitMeasure_SizeUnitMeasureCode] FOREIGN KEY([SizeUnitMeasureCode])
REFERENCES [Production].[UnitMeasure] ([UnitMeasureCode])

ALTER TABLE [Production].[Product] CHECK CONSTRAINT [FK_Product_UnitMeasure_SizeUnitMeasureCode]


ALTER TABLE [Production].[Product]  WITH CHECK ADD  CONSTRAINT [FK_Product_UnitMeasure_WeightUnitMeasureCode] FOREIGN KEY([WeightUnitMeasureCode])
```

Zoom in | Open in new window

Figure 10. Example of code in an Execute T-SQL Statement Task recreating foreign key constraints.

If you use an “Execute T-SQL Statement Tasks” to drop and recreate the constraints, each script will have many statements in them. Do not place lines containing “GO” between the statements in the script dropping the constraints. Although the task will complete, the precedence constraints to the subsequent tasks will be evaluated after the first “GO” statement, and subsequent tasks will begin running before the end of the script that drops the constraints. If you believe you must put “GO” statements in your T-SQL, place each batch of T-SQL statements in a separate Execute T-SQL Statement Task instead. In the cases of dropping and altering of constraints as discussed here, “GO” statements are not necessary.

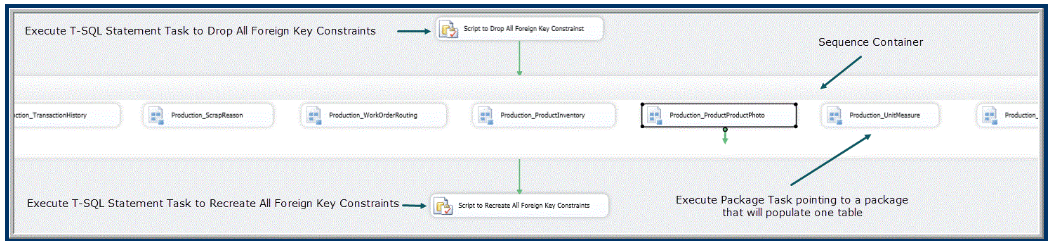
Figure 11shows the control flow for a _Main package that drops constraints, populates tables and recreates constraints.


Control Fl...
Data Flow
Parameters
Event Handlers
Package Explorer
Execute T-SQL Statement Task to Drop All Foreign Key Constraints
Script to Drop All Foreign Key Constraints
Precedence Constraints
dbo_ErrorLog
dbo_DatabaseLog
Production_UnitMeasure
Production_TransactionHistory
Production_ScrapReason
Production_ProductSubcategory
Production_ProductReview
Production_ProductProductPhoto
Production_Produ
Execute T-SQL Statement Task to Recreate All Foreign Key Constraints
Script to Recreate All Foreign Key Constraints
Execute Package Task pointing to a package that will populate one table.

Zoom in | Open in new window

Figure 11. Example of control flow for a _Main package dropping constraints, populating tables and recreating constraints.

This package can be configured a bit more easily by using a sequence container to contain the packages that migrate the data from individual tables. Items placed in a sequence container with no constraints among them are all eligible to run in parallel, which is logically the same as the flow shown in Figure 10. See Figure 12.


Execute T-SQL Statement Task to Drop All Foreign Key Constraints
Script to Drop All Foreign Key Constrainst
Sequence Container
ction_TransactionHistory
Production_ScrapReason
Production_WorkOrderRouting
Production_ProductInventory
Production_ProductProductPhoto
Production_UnitMeasure
Production_
Execute T-SQL Statement Task to Recreate All Foreign Key Constraints
Script to Recreate All Foreign Key Constraints
Execute Package Task pointing to a package that will populate one table

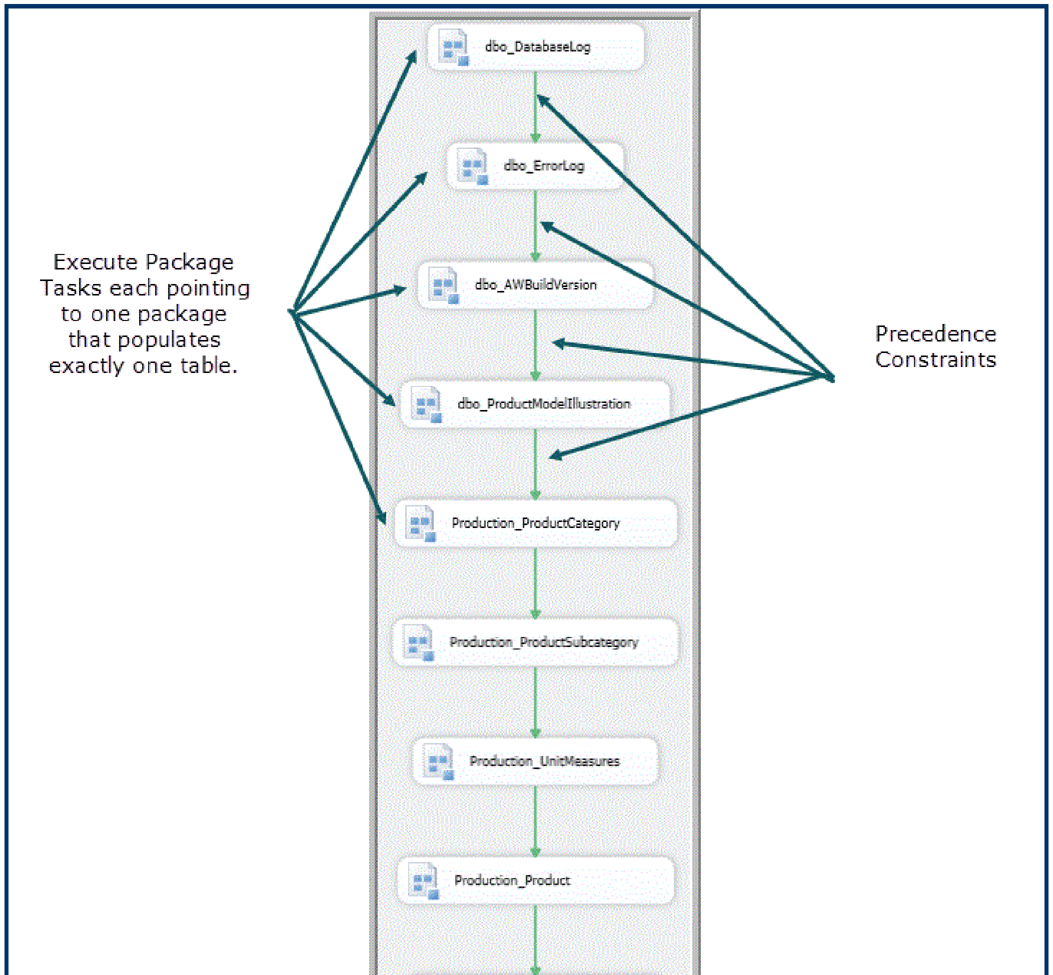
Zoom in | Open in new window

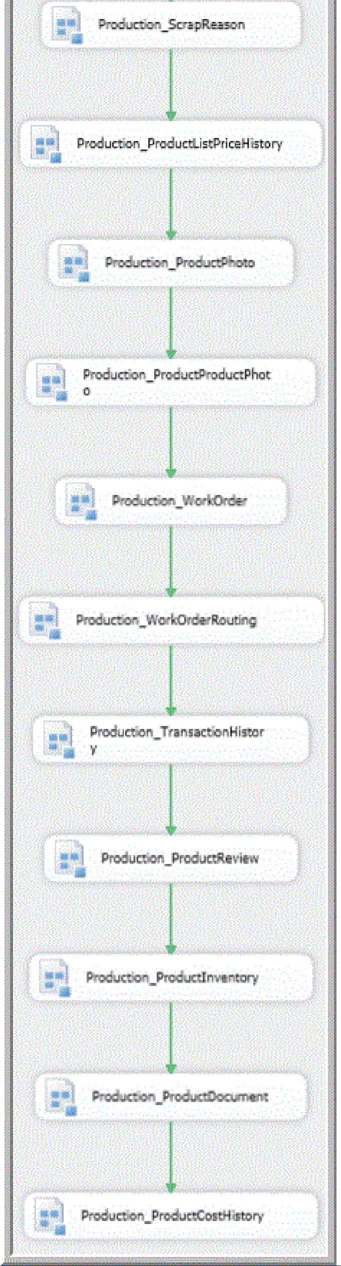
Figure 12. Example of control flow for a _Main package dropping constraints, populating tables and recreating constraints using a sequence container.

In my case, I knew that the data contained many errors. These errors meant that there were many records to be inserted in a destination table having a foreign key that did not have a corresponding key value in a record in the referenced table. I wanted to discover these errors on a table-by-table basis at the same time as attempts to load bad data were occurring, and I was interested in loading them in the logically correct order. Therefore, dropping and recreating constraints was not a strategy I wanted to use. If you wish to use this strategy, see Disable, enable, drop and recreate SQL Server Foreign Keys. Be sure to save a copy of your target database schema before executing any scripts dropping referential integrity because once you have run a script dropping constraints, you can't use the same database to create a script to recreate the constraints!

Second Approach: Ad Hoc Creation of Load Order

In an effort to determine a load order that would merely work, optimally or not, I tried staring at a database diagram showing the dependencies of tables upon each other while I attempted to determine a possible load order. This resulted in a list (and potentially many lists) that was a complete solution to the problem where one package executed at a time. This approach did result in satisfying my goal of loading the tables in a correct order. The _Main package had a very vertical linear appearance and this approach results in a _Main package looking similar to Figure 13.


Execute Package Tasks each pointing to one package that populates exactly one table.
dbo_DatabaseLog
dbo_ErrorLog
dbo_AWBuildVersion
dbo_ProductModelIllustration
Production_ProductCategory
Production_ProductSubcategory
Production_UnitMeasures
Production_Product
Precedence Constraints


Production_ScrapReason
Production_ProductListPriceHistory
Production_ProductPhoto
Production_ProductProductPhot o
Production_WorkOrder
Production_WorkOrderRouting
Production_TransactionHistor y
Production_ProductReview
Production_ProductInventory
Production_ProductDocument
Production_ProductCostHistory

Zoom in | Open in new window

Figure 13. Example of a control flow for a _Main package populating tables in an ad hoc order that satisfies referential integrity constraints.

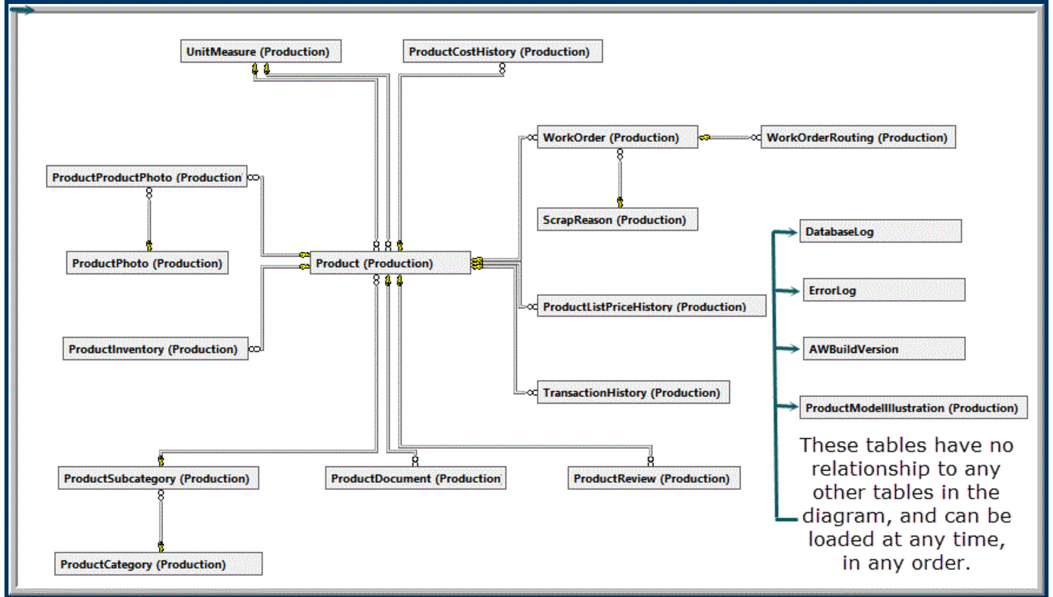
Third Approach: Mimicking Foreign Key Constraints in the Design of Control Flow Precedence Constraints

I knew I could achieve a solution that is better than one achieved through the first two approaches. Although I had not initially considered the issue, I amended my goal in completing this project to include having as many packages run in parallel as practicable. In the databases with which I have worked, it usually turns out that once several tables of central importance to a database are loaded, the dependencies upon them usually branch out to several groupings of tables. Usually, none of the tables in any grouping are dependent upon tables in any other grouping. Packages loading data in these separate groupings should be able to load data in parallel with packages in other groupings thereby increasing performance and reducing load time.

Parallel operations work very well in servers running SSIS packages. Factors that limit performance are usually associated with IO and disk contention. With modern-day storage devices such as Storage Area Networks (SANS) and fast networks, any increase in contention will still permit tables to be loaded more quickly than they would load one table at a time. I have never successfully blamed poor performance on bottlenecks in network speed in a well-implemented network.

I eventually realized that I could leverage a database diagram that shows the lines representing foreign key constraints to specify load order. All I had to do was create one precedence constraint per foreign key relationship in the SSIS package that executes packages for individual tables. This means that if a table is dependent upon several other tables the loading of each table upon which another table depends must complete before the dependent table can load. It also means when all the necessary tables are loaded, a dependent table can execute immediately and in parallel with other packages that have had their precedence constraints satisfied.

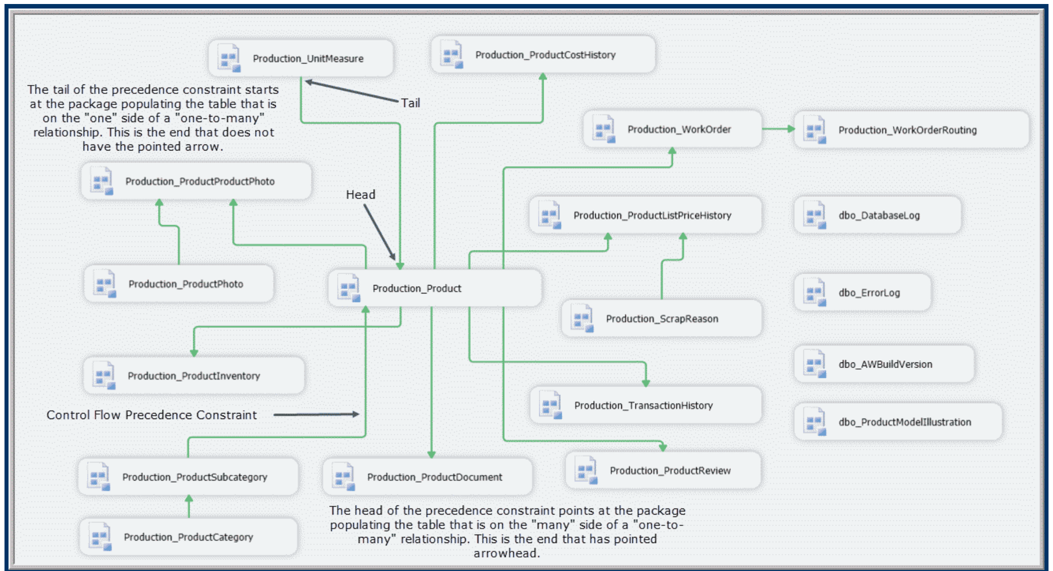
Figure 14 displays a database diagram showing several tables in the Production schema, several tables in the dbo schema, and the foreign key relationships that exist between these tables.


UnitMeasure (Production)
ProductCostHistory (Production)
WorkOrder (Production)
WorkOrderRouting (Production)
ProductProductPhoto (Production
ScrapReason (Production)
DatabaseLog
ProductPhoto (Production)
Product (Production)
ErrorLog
ProductListPriceHistory (Production)
AWBuildVersion
ProductInventory (Production)
TransactionHistory (Production)
ProductModelIllustration (Production)
These tables have no relationship to any other tables in the diagram, and can be loaded at any time, in any order.
ProductSubcategory (Production)
ProductDocument (Production
ProductReview (Production)
ProductCategory (Production)

Zoom in | Open in new window

Figure 14. A database diagram showing several tables in the Production schema, several tables in the dbo schema and the foreign key relationships that exist between these tables.

We can create a _Main package that follows the pattern of this database diagram and implements the precedence constraints that parallel the foreign key restraints the diagram displays. See Figure 15.


Production_UnitMeasure
Production_ProductCostHistory
The tail of the precedence constraint starts at the package populating the table that is on the "one" side of a "one-to-many" relationship. This is the end that does not have the pointed arrow.
Tail
Production_WorkOrder
Production_WorkOrderRouting
Production_ProductProductPhoto
Head
Production_ProductListPriceHistory
dbo_DatabaseLog
Production_ProductPhoto
Production_Product
Production_ScrapReason
dbo_ErrorLog
dbo_AWBuildVersion
Production_ProductInventory
Production_TransactionHistory
dbo_ProductModelIllustration
Control Flow Precedence Constraint
Production_ProductReview
Production_ProductSubcategory
Production_ProductDocument
The head of the precedence constraint points at the package populating the table that is on the "many" side of a "one-to-many" relationship. This is the end that has pointed arrowhead.
Production_ProductCategory

Zoom in | Open in new window

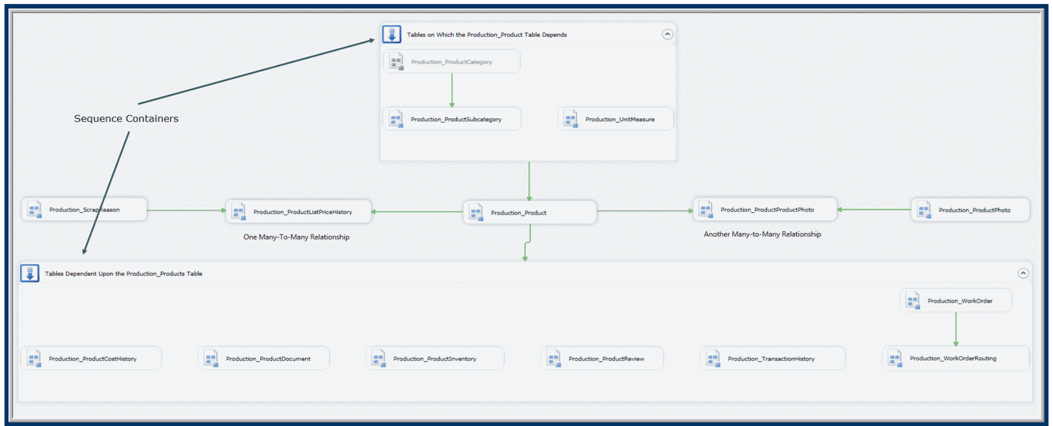
Figure 15. Example of a control flow for a _Main package populating

tables in an order dictated by referential integrity constraints.

The tail of the precedence constraint is connected to the primary or unique key table, and the head of the precedence constraint is placed on the foreign key table. You do not need to include a precedence constraint for a foreign key constraint representing a reference of a table to itself. There only needs to be (and only can be) one constraint where a table references another table multiple times.

More About Sequence Containers

Shortly after I started down the path of modeling the precedence constraints upon the foreign key constraints, I started grouping tasks into sequence containers, and this worked well. I put all of the tables that were not dependent upon any other tables and that did not have any dependencies upon them in one sequence container, and put no precedence constraints between any of them. See Figure 16.


Tables on Which the Production_Product Table Depends
Production_ProductCategory
Production_ProductSubcategory
Production_UnitMeasure
Sequence Containers
Production_ScrapReason
Production_ProductListPriceHistory
One Many-To-Many Relationship
Production_Product
Production_ProductProductPhoto
Another Many-to-Many Relationship
Production_ProductPhoto
Tables Dependent Upon the Production_Products Table
Production_WorkOrder
Production_ProductCostHistory
Production_ProductDocument
Production_ProductInventory
Production_ProductReview
Production_TransactionHistory
Production_WorkOrderRouting

Zoom in | Open in new window

Figure 16. Example of a control flow for a _Main package using sequence containers and populating tables in an order dictated by referential integrity constraints.

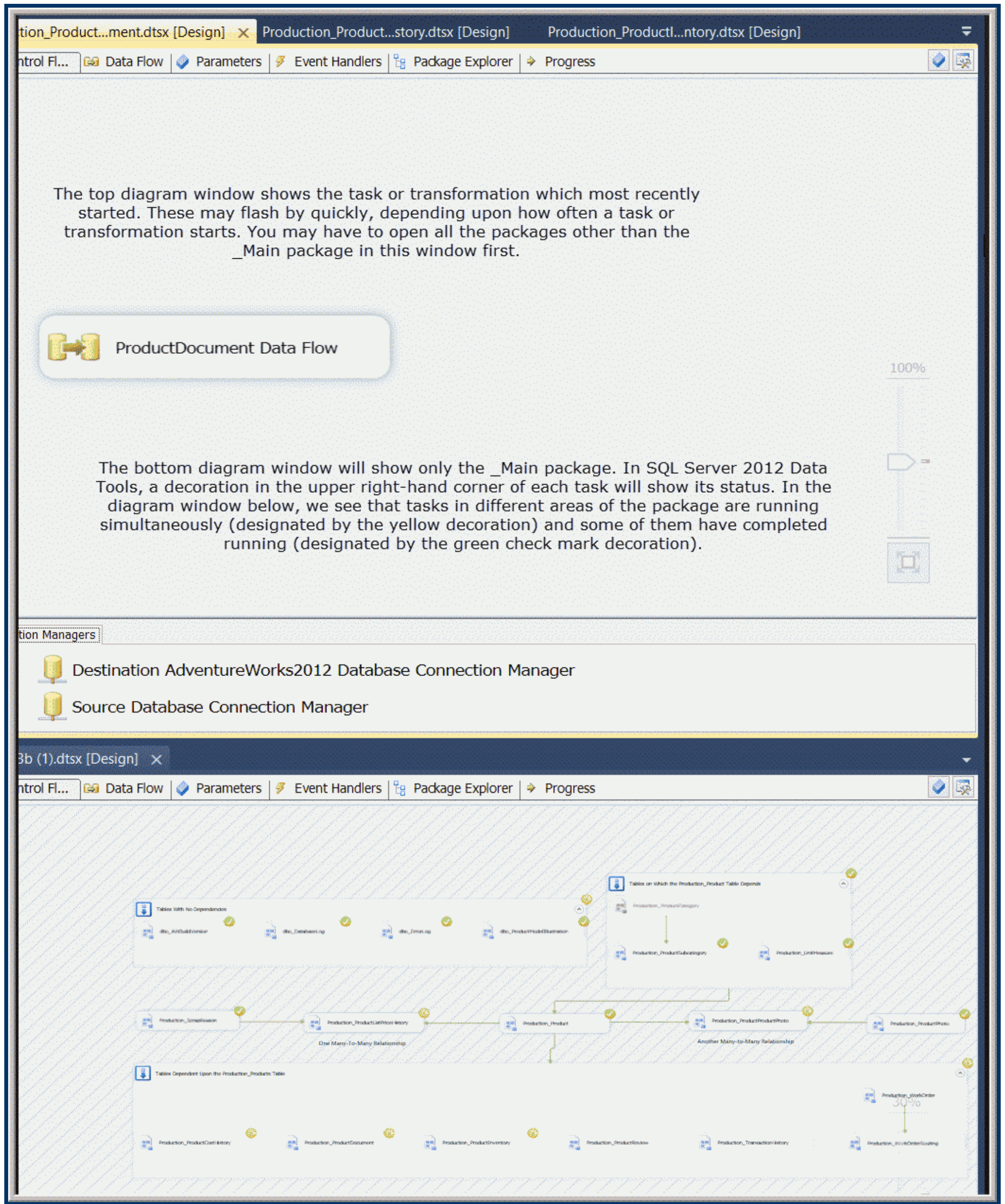
While we usually think of Sequence Containers as placeholders that merely group tasks together so we can view the package as an aggregation of related tasks, the sequence container has a TransactionOption property that can be set to a value of “Required,” which would mean that if any of the sequence containers has a failure within it, then all of the data loading associated with that container would be rolled back. This means that if the tasks in some sequence containers completed successfully and the tasks in other sequence containers did not execute successfully, then once we fixed the problems causing the failures, we could re-execute the entire sequence containers that did not complete successfully without having to agonize over exacting which packages we needed to re-execute. SSIS has another concept, called "checkpoints" (not to be confused with database checkpoints that flush "dirty" data in memory to disk) that can make re-execution of packages that previously failed even more automated. For packages to be deployed and executed in a production environment where the operators are not executing packages through BIDS or SSDT, you would have to use the checkpoint method to make re-execution easier, because an operator would not have the ability to execute only certain sequence containers.

If the database uses schema objects to group tables and each schema contains a relatively few tables, it is likely that most of these tables within a schema will wind up in the same sequence container. At least one table in a sequence container (and sometimes only one table) will likely reference a table outside of its schema. In those cases, it usually works out that once a few significant and common tables in the load order are loaded, then precedence constraints can flow to the various sequence containers in parallel.

Viewing Execution of Components Running in Parallel

Using the BIDS designer window to view the execution of a package, especially a package having tasks running in parallel, usually results in an impressive but unhelpful continuous swapping of screens within the designer window showing the beginning of execution of different components. A more useful way to view the execution of packages in parallel (providing a more impressive spectacle to passersby) is to use the screen splitting feature of the designer window. To do this, right-click on the tab of the _Main.dtsx package while it is opened in the designer and select “New Horizontal Tab Group.” Next, click on the _Main.dtsx tab in the lower window and begin execution in debug mode by pressing F5.

I find that it is more useful to turn my monitor so that it uses a portrait orientation (which I prefer in general). When you execute the _Main package, the _Main package will show its progress and remain fixed, while other activities flash by on in the other designer panel. You can easily monitor the progress of the tasks and packages in the portion of the screen showing the execution of the _Main package. See Figure 17 shows the project during execution. The tasks with a green arrow in the upper right-hand corner have successfully completed execution, the tasks with the yellow arrow in the upper right-hand corner are executing in parallel, and the tasks with no decoration in the upper right-hand corner have not yet begun to execute. I think this was a more colorful and impressive display during execution with the prior versions of BIDS where the entire box representing a task had color throughout.


tion_Product...ment.dtsx [Design]
Production_Product...story.dtsx [Design]
Production_Productl...ntory.dtsx [Design]
ntrol Fl...
Data Flow
Parameters
Event Handlers
Package Explorer
Progress
ProductDocument Data Flow
100%
tion Managers
Destination AdventureWorks2012 Database Connection Manager
Source Database Connection Manager
3b (1).dtsx [Design]
ntrol Fl...
Data Flow
Parameters
Event Handlers
Package Explorer
Progress
Tables With No Dependencies
dbo_AWBuildVersion
dbo_DatabaseLog
dbo_ErrorLog
dbo_ProductModelIllustration
Tables on Which the Production_Product Table Depends
Production_ProductCategory
Production_ProductSubcategory
Production_UnitMeasure
Production_ScrapReason
Production_ProductListPriceHistory
One Many-To-Many Relationship
Production_Product
Production_ProductProductPhoto
Another Many-to-Many Relationship
Production_ProductPhoto
Tables Dependent Upon the Production_Products Table
Production_WorkOrder
30%
Production_ProductCostHistory
Production_ProductDocument
Production_ProductInventory
Production_ProductReview
Production_TransactionHistory
Production_WorkOrderRouting

The top diagram window shows the task or transformation which most recently started. These may flash by quickly, depending upon how often a task or transformation starts. You may have to open all the packages other than the _Main package in this window first.

The bottom diagram window will show only the _Main package. In SQL Server 2012 Data Tools, a decoration in the upper right-hand corner of each task will show its status. In the diagram window below, we see that tasks in different areas of the package are running simultaneously (designated by the yellow decoration) and some of them have completed running (designated by the green check mark decoration).

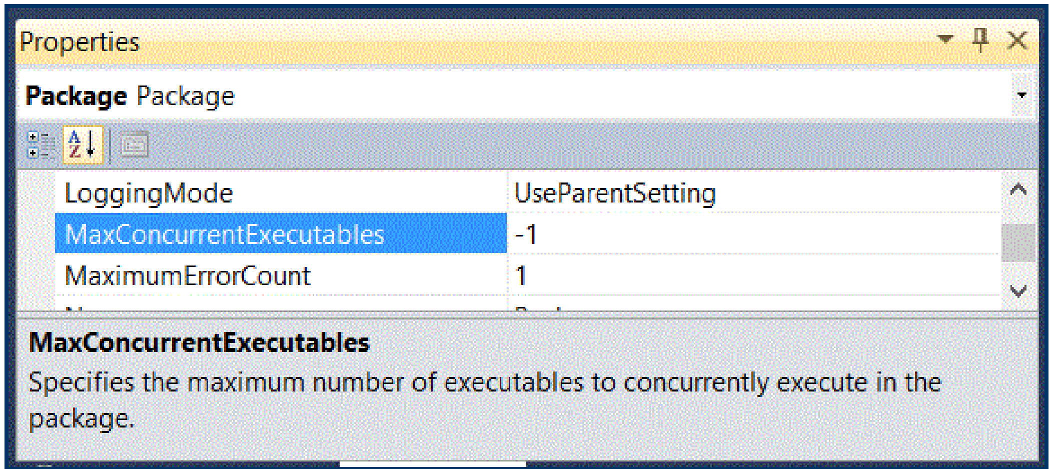
Zoom in | Open in new window

Figure 17. Image of SSDT during execution of a package using two diagram windows.

Performance Considerations

People frequently ask if a package permitting the loading of data in parallel might result in decreased performance because too many packages were simultaneously competing for resources. This can happen, but if it does, it is possible to control use of processor and memory resources through two properties.

The first property limiting or expanding the amount of concurrent activity that will be performed in the execution of an SSIS object is a property of a *package* and is called MaximumConcurrentExecutables. See Figure 18.



LoggingMode	UseParentSetting
MaxConcurrentExecutables	-1
MaximumErrorCount	1

MaxConcurrentExecutables
Specifies the maximum number of executables to concurrently execute in the package.

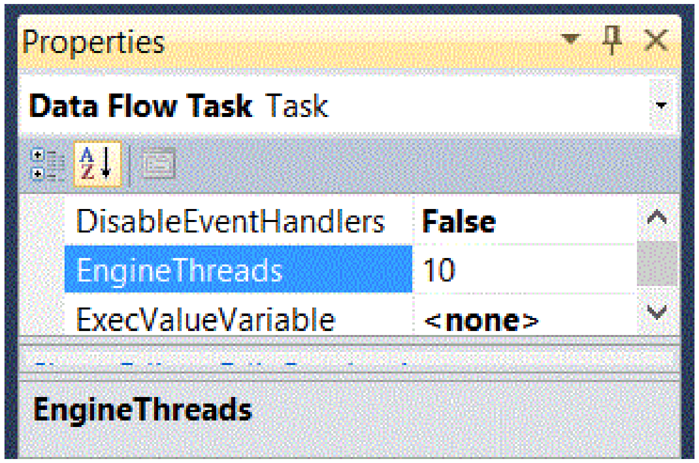
Zoom in | Open in new window

Figure 18. Properties window of a package showing the MaxConcurrentExecutables property.

We usually associate the word “executable” with a binary object containing code that can be run directly by the operating system or the .NET Common Language Runtime. Throughout SSIS, however, “executable” is consistently used to mean “control flow task,” which is represented as XML in a file with the .dtsx extension. Therefore, this property controls the number of *tasks* that can run concurrently within a package. Because each task in _Main is associated with the loading of a single table, this setting will control the number of tables that can be loaded in parallel.

When a property appears to need a value that is a positive integer, we usually associate a “0” or “-1” value as “infinite,” or any large value that the environment permits. Again, SSIS uses this property differently. The value of “-1” in the context of the MaximumConcurrentExecutables property means “the number of cores available to the SSIS runtime plus 2.” It should only be necessary to change this value if SSIS is starving other concurrent operating system processes of resources that should be more available to these other processes than what you actually experience.

The second property limiting or expanding the amount of concurrent activity that will be performed in the execution of an SSIS object is a property of each *Data Flow Task* and is called EngineThreads. See Figure 19.



Data Flow Task Task

DisableEventHandlers	**False**
EngineThreads	10
ExecValueVariable	**<none>**

EngineThreads

Figure 19. Properties of a data flow showing the EngineThreads property.

Unless you have an extremely complicated data flow, the default setting for the number of threads to be devoted to this task should be more than adequate. This setting will not force the use of all 10 threads to execute the data flow if they are not needed. In most cases the loading of a destination table from a single source table will require few threads.

Conclusion

In conclusion, there is more than one way to use SSIS to load tables in a database so that referential integrity constraints are not violated. In this article I show three of them, and recommend that that the foreign key constraints graphically represented in a SQL Server database diagram can be used to determine which precedence constraints should be created to satisfy referential integrity constraints and permit optimal performance.