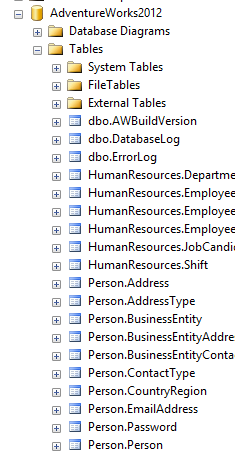
ETL with SQL Server SSIS – From transactional data to SQL Server DW

# Objective:

Through this assignment, you will learn a typical ETL process that moves data from a production transactional system to a dimensional data warehouse. You will use SQL Server 2016 Integration Services to extract data and populate the data warehouse. You will use connection managers to connect to data sources, create audit table and populate them to track ETL activities and use data flow transformation to modify the source data and populate the data warehouse.

# Preparation:

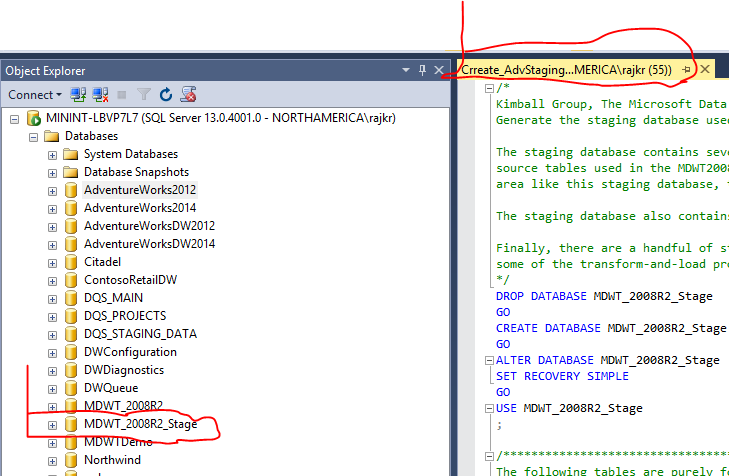
1. Create all the databases that will be used in the assignment
   1. Transactional database – Download the Adventure Works 2012 (.bak file provided) and restore it to your database using SQL Server Management Studio



* 1. Staging database – This is the database that will be used to move production data to a staging database. Use the attached SQL DDL file to create the Staging database. This script creates the intermediary database to move the data from transactional system to a staging area to minimize impact on the transactional system.

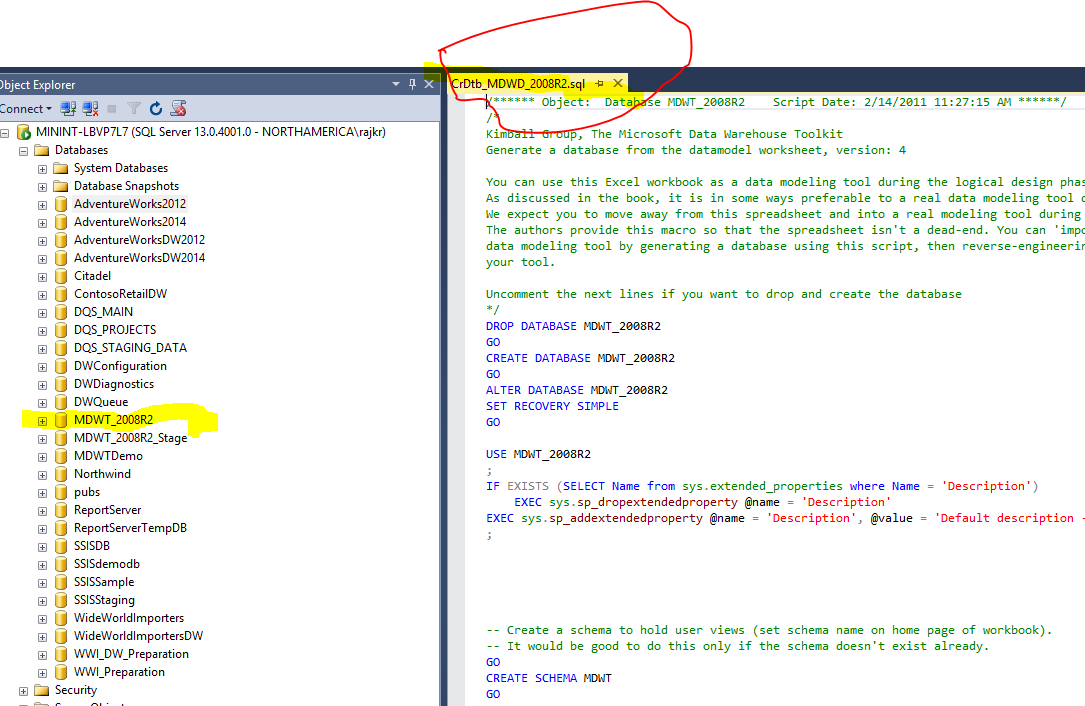
Name of the SQL File to execute: CrDtb\_MDWD\_2008R2\_Stage.sql

This will create the staging tables with no data



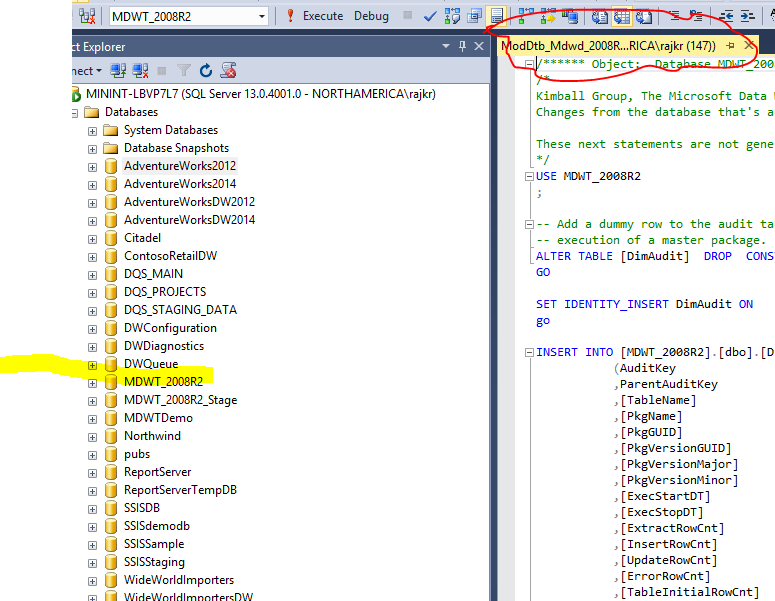
* 1. Now create the Data Warehouse database where we finally populate the dimensional model using this SQL Script CrDtb\_MDWD\_2008R2.sql

This will create the tables as shown below with no data.



* 1. Finally run this SQL to modify the Data Warehouse table using this script ModDtb\_Mdwd\_2008R2.sql

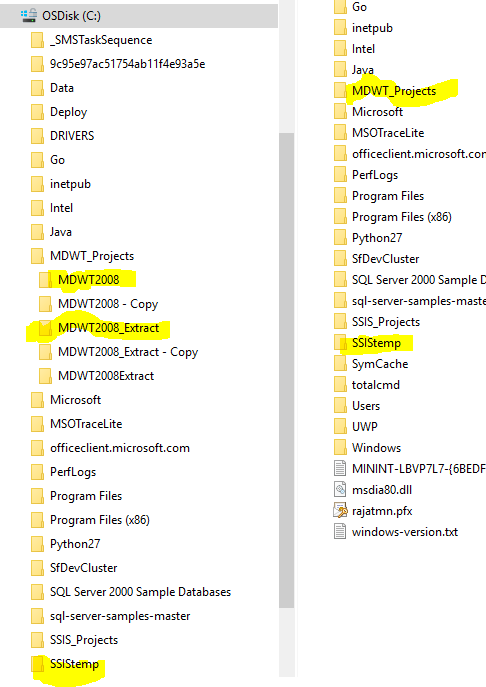
This modifies the Data warehouse by creating an Audit table and populates some base line data to get the ETL auditing started



Now you have all the databases needed to create and run ETL packages.

* 1. Finally create a folder for dumping error raw files C:\SSIStemp; Create another folder and extract the attached solution files into that folder;

C:\MDWT\_Projects

  
The two projects in the folder MDWT2008\_Extract and MDWT2008 contain the full SSIS solution used to move data to Staging table and from staging tables to Data Warehouse respectively; You will use this as a reference to build your solution.

# Deliverables:

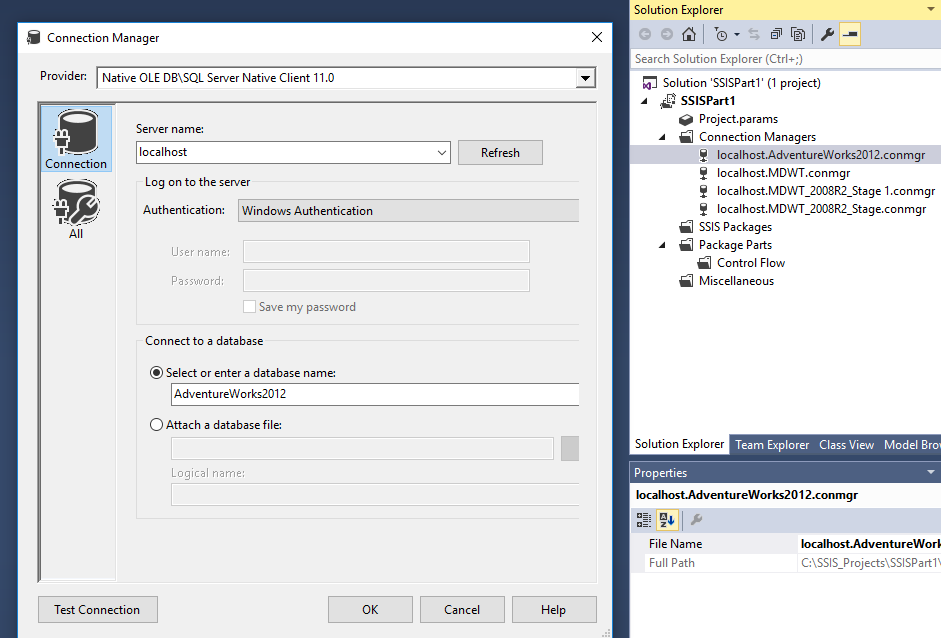
You have 4 tasks to complete as part of the assignment.

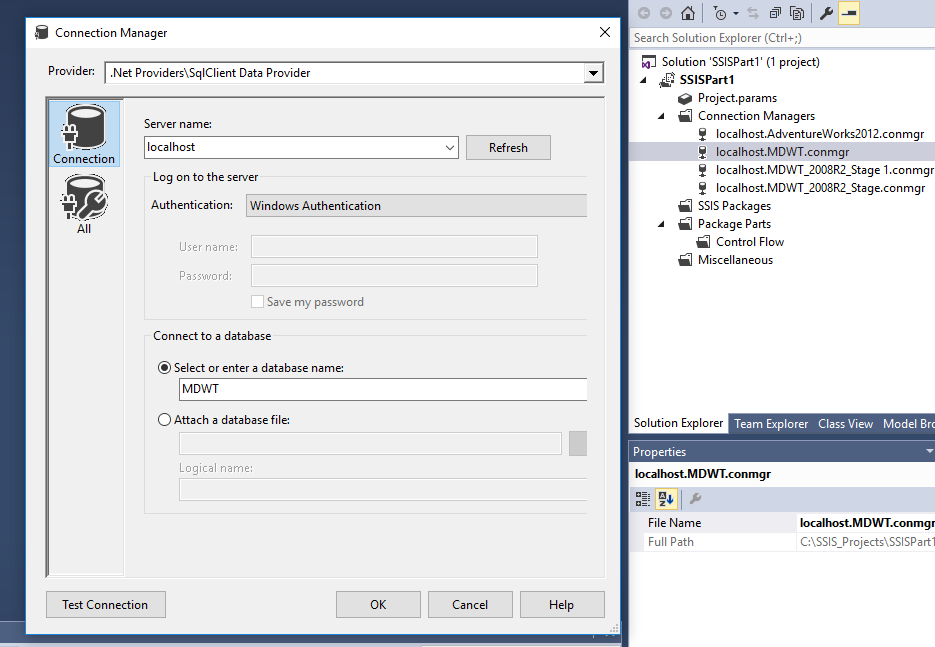
## Task 1

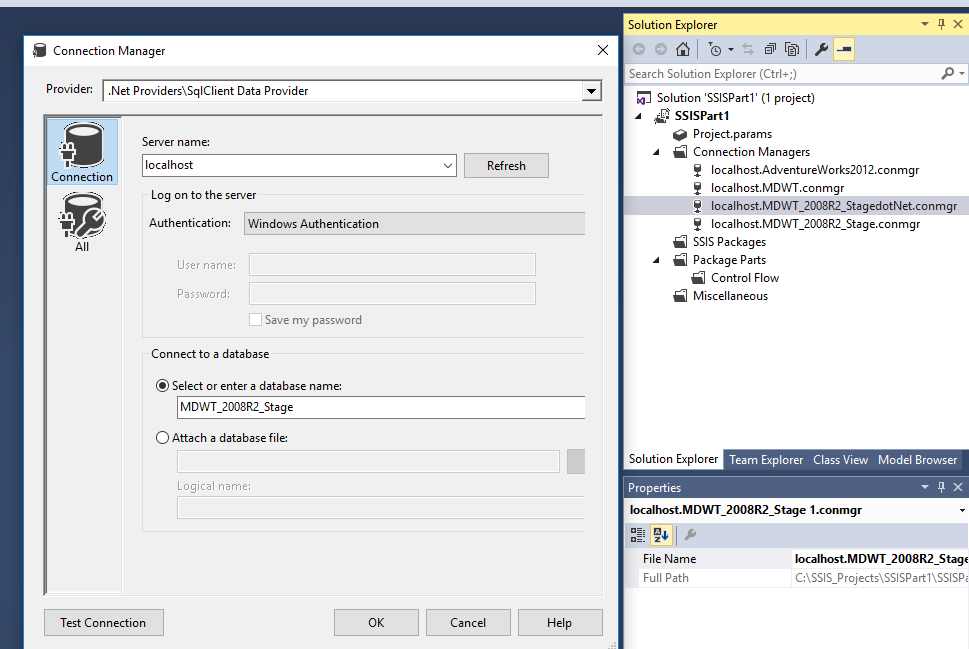
1. Create a SSIS Projects that has two packages (naming convention: each package should have 3 letters from your last name as part of the name of the package -e.g kriCurrency.dtx.)

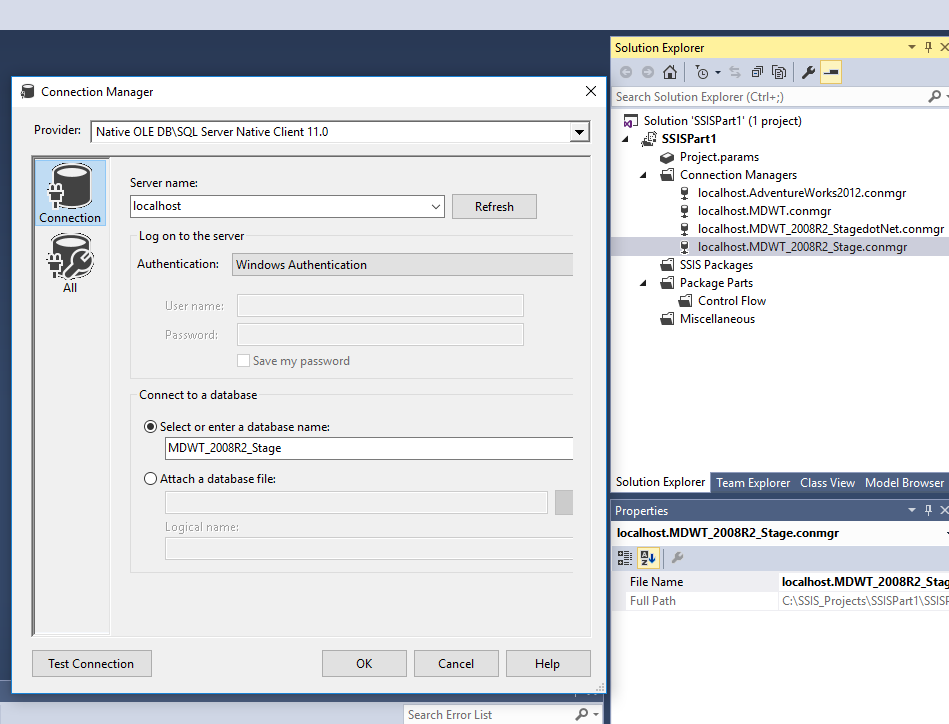
Each of these packages will populate the StageCurrency and StageProduct table in the staging database. You should include the Control flow, data flow and a screen that shows all the tasks are completed as shown below.

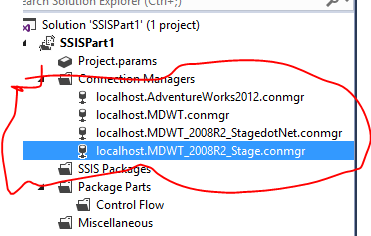
1. Create a new SSIS Project
2. Create **4 connection Managers** as shown below



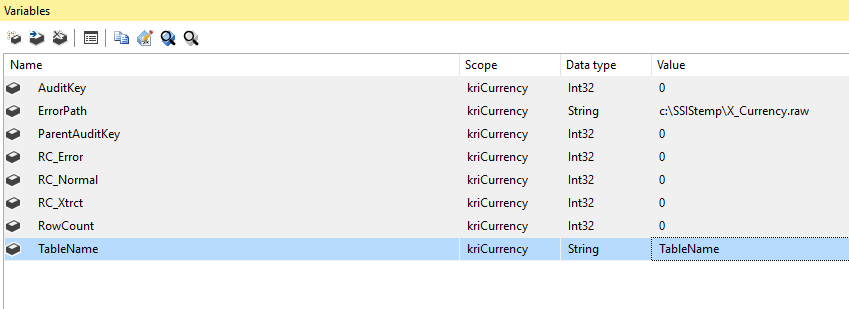








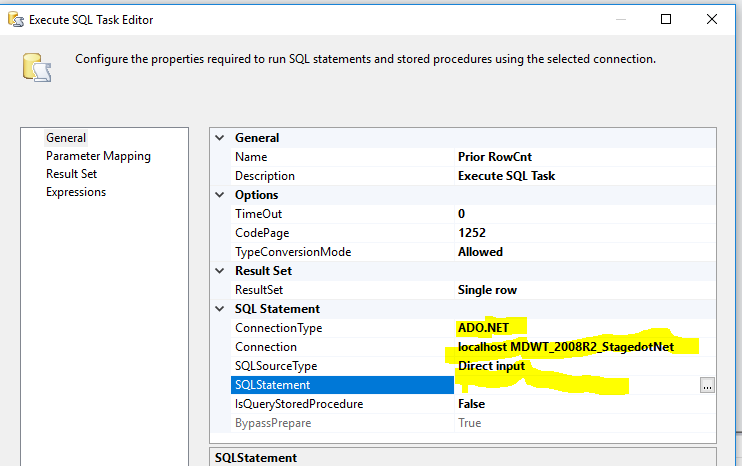
1. Create a new package and name it using the naming convention – kriCurrency
2. Click SSIS menu -> Variables and add the following variables – these are variable we will be using through the package to update the Audit Table and other execution relevant data



## Building the package

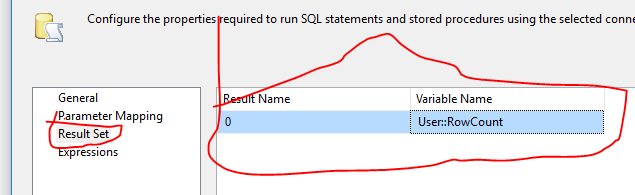
Start with the control flow and add the following:

1. **Execute SQL Task – Double click and change the following:**



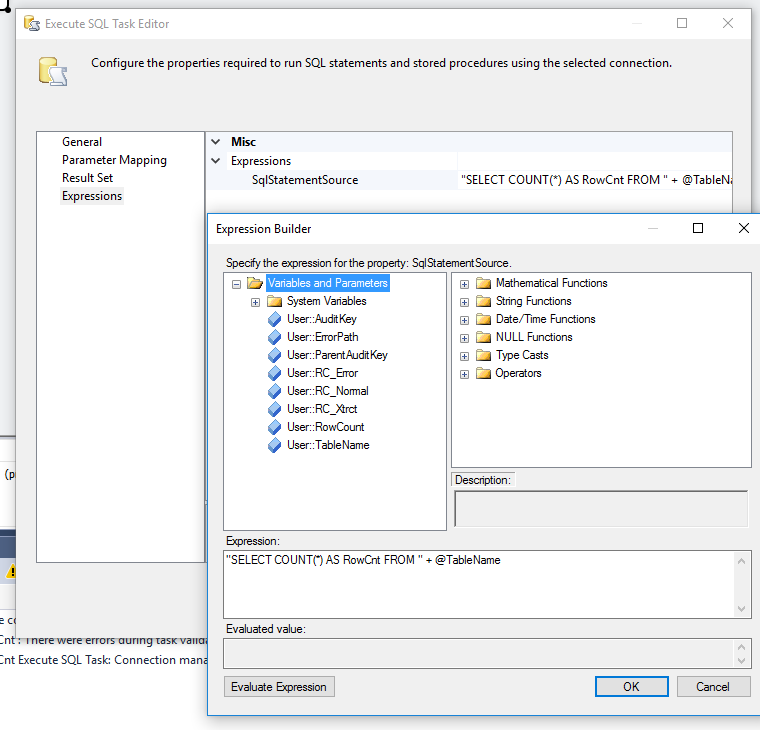
The SQL Statement will be built using an expression.

The output from this query will be written to a variable so that we can use it to write to the audit table.



We will now write an expression for the SQL Query so that we can reuse this package and make changes to the name of the table using a user variable.

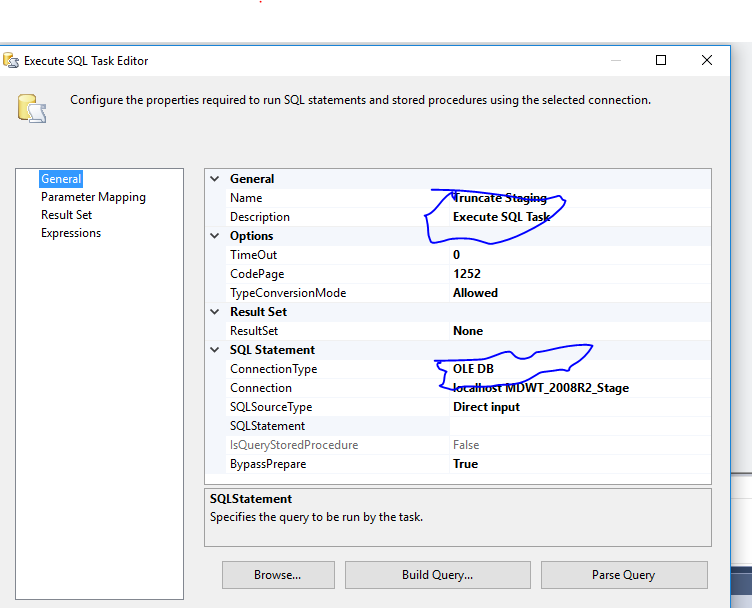
Write an expression for the sql statement as follows:



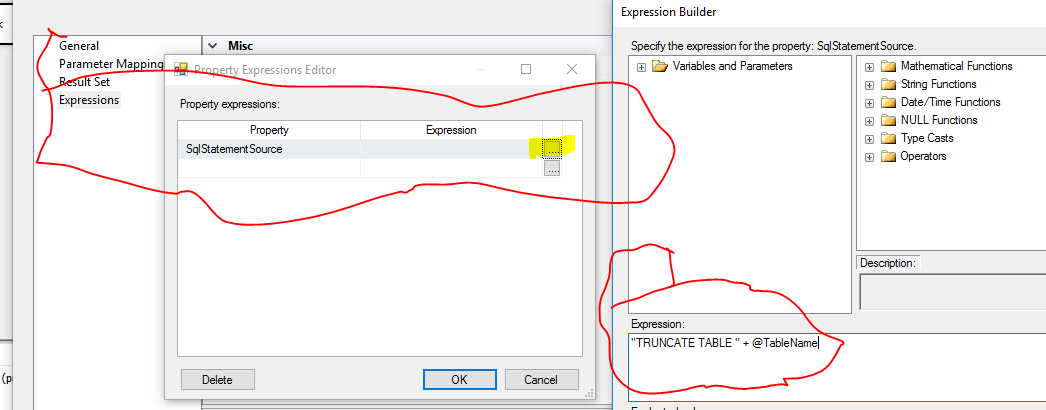
Click on evaluate expression to verify that the statement reads correctly and uses the variables.

1. **Add another Execute SQL Task with the following properties**

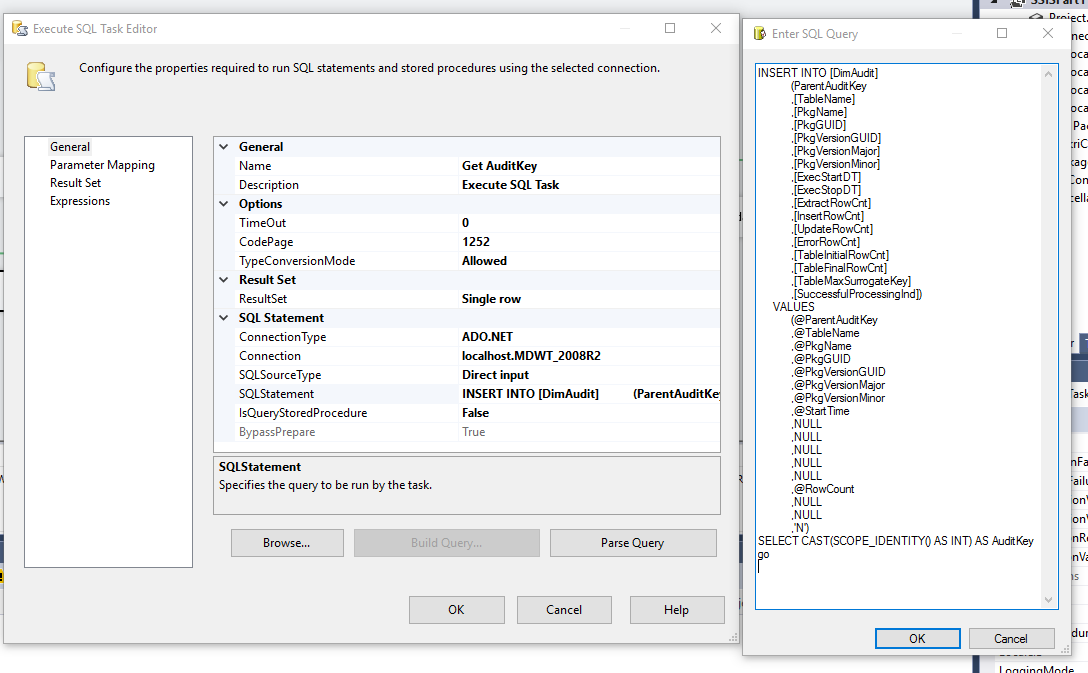
Basic Set -up including name:



Expression for the SQL Statement



1. **Add another execute SQL Task with the following definition**



Name: Get AuditKey

Result Set: Single row

Connection Type: ADO.NET

Connection: localhost.MDWT\_2008R2

SQL Statement:

INSERT INTO [DimAudit]

(ParentAuditKey

,[TableName]

,[PkgName]

,[PkgGUID]

,[PkgVersionGUID]

,[PkgVersionMajor]

,[PkgVersionMinor]

,[ExecStartDT]

,[ExecStopDT]

,[ExtractRowCnt]

,[InsertRowCnt]

,[UpdateRowCnt]

,[ErrorRowCnt]

,[TableInitialRowCnt]

,[TableFinalRowCnt]

,[TableMaxSurrogateKey]

,[SuccessfulProcessingInd])

VALUES

(@ParentAuditKey

,@TableName

,@PkgName

,@PkgGUID

,@PkgVersionGUID

,@PkgVersionMajor

,@PkgVersionMinor

,@StartTime

,NULL

,NULL

,NULL

,NULL

,NULL

,@RowCount

,NULL

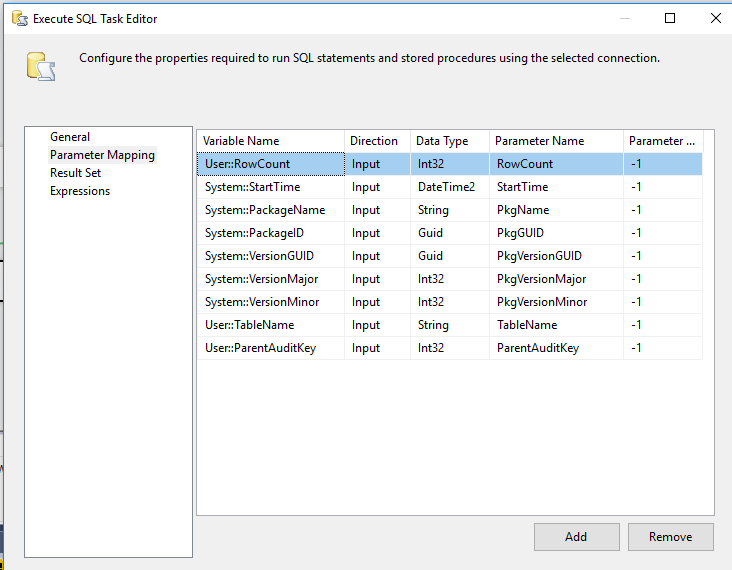
,NULL

,'N')

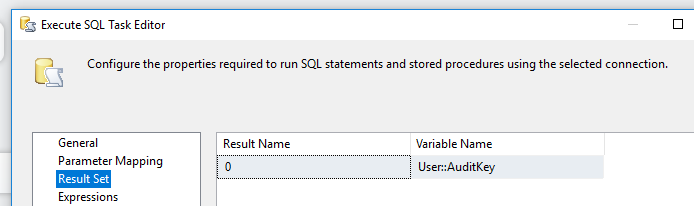
SELECT CAST(SCOPE\_IDENTITY() AS INT) AS AuditKey

Go

Parameter Mapping



Result Set

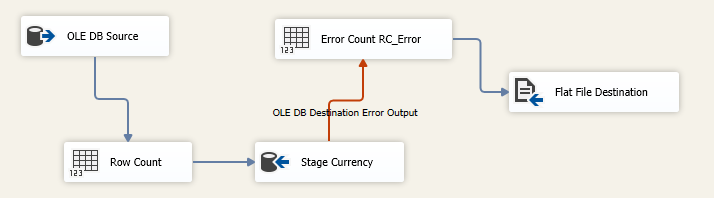


1. **Add a data flow task and connect it**



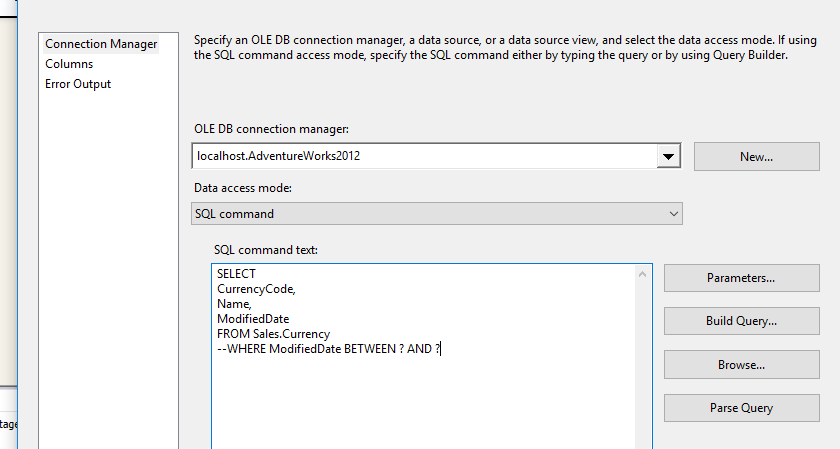
Define Data flow task – Name it Extract from Trxn System

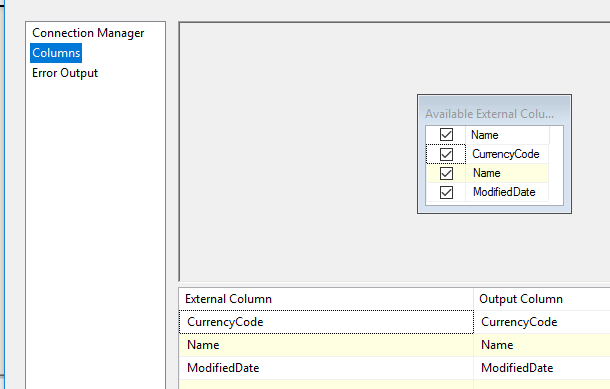
Here is the data flow task



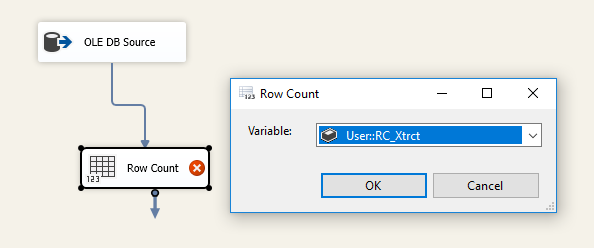
Configure each one of them.

1. Add an OLEdb Source and configure as follows:

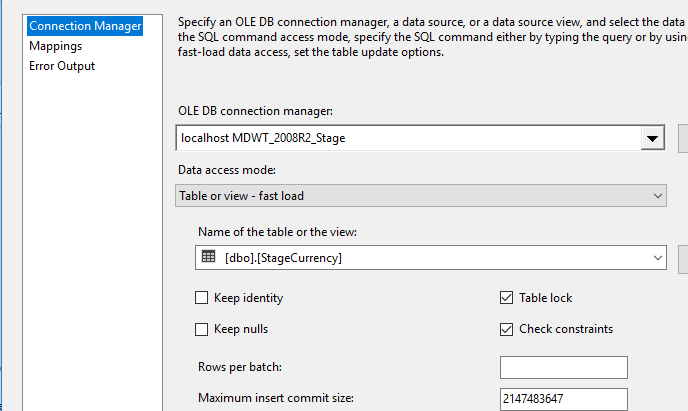


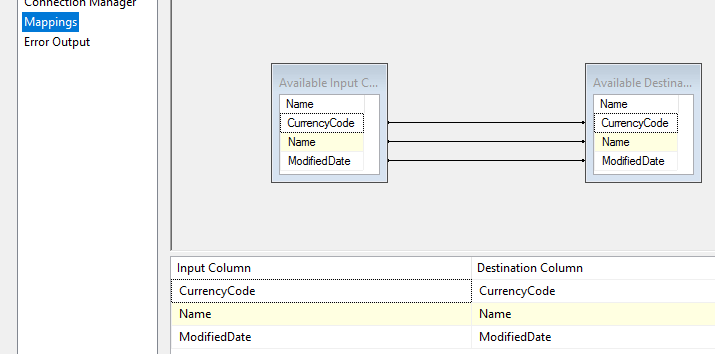


1. Add a Row Count task and connect to the User variable RC\_Xtrct as shown:

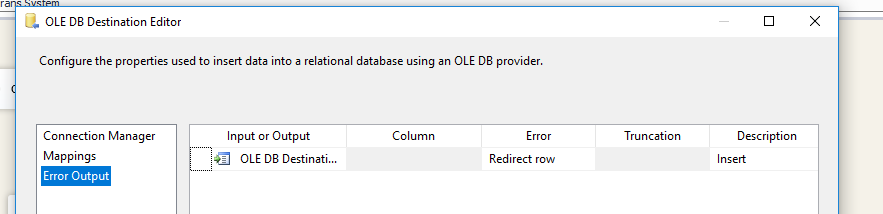


1. Add an OleDb destination Task and configure as follows:

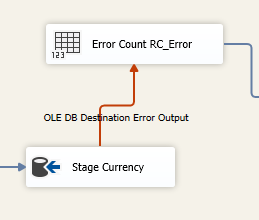


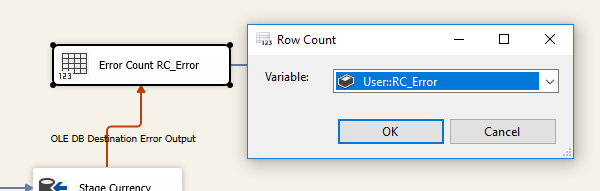


Send the output rows as error entries to be recorded

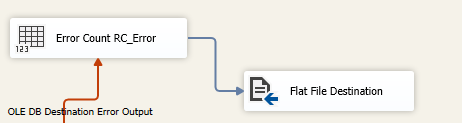


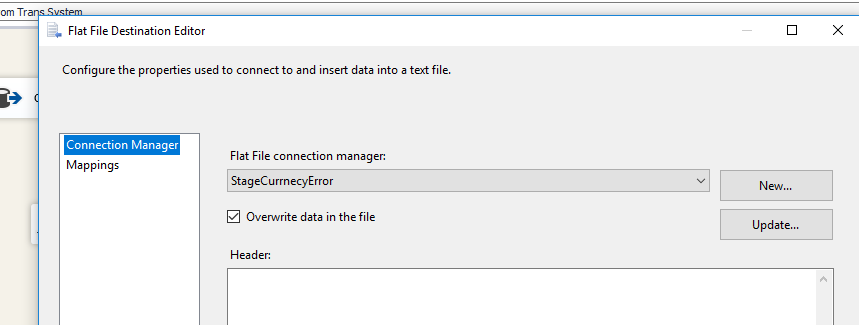
1. Add a row count task to count the error and assign it to a variable

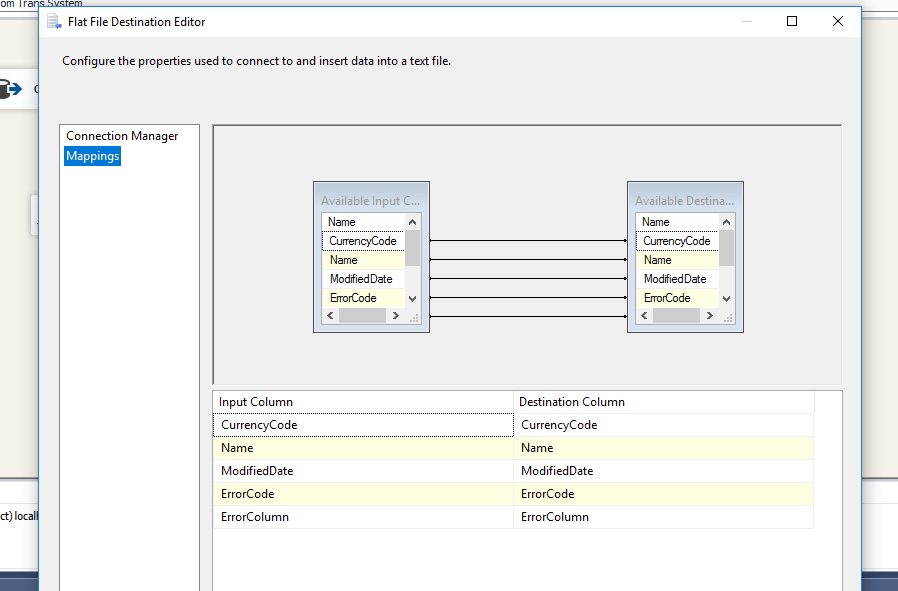


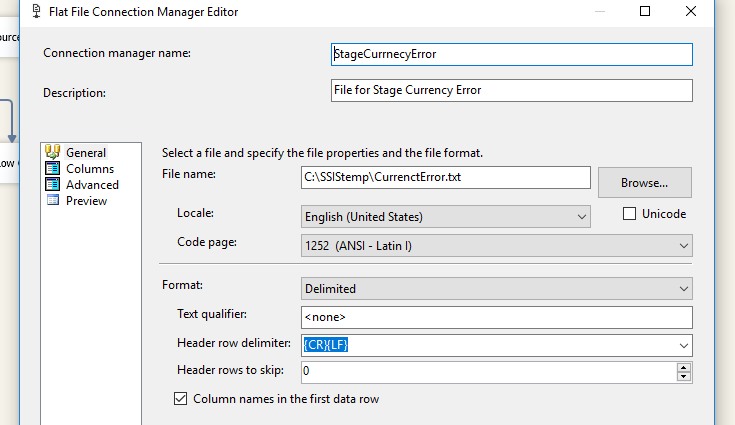


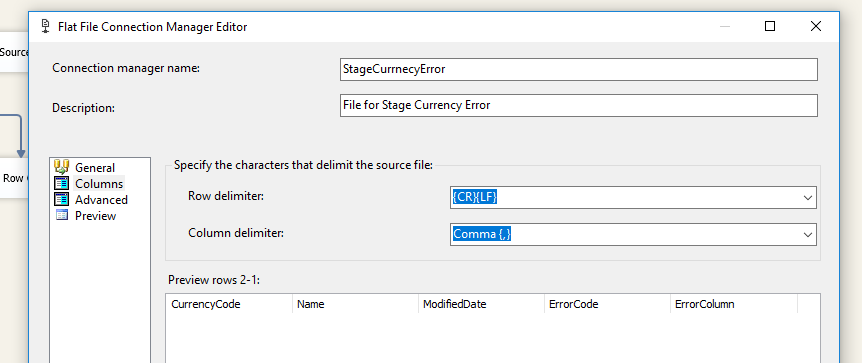
1. Capture the errors in a file using a File destination; Add a File destination, connect the error output from the previous task and configure as follows:



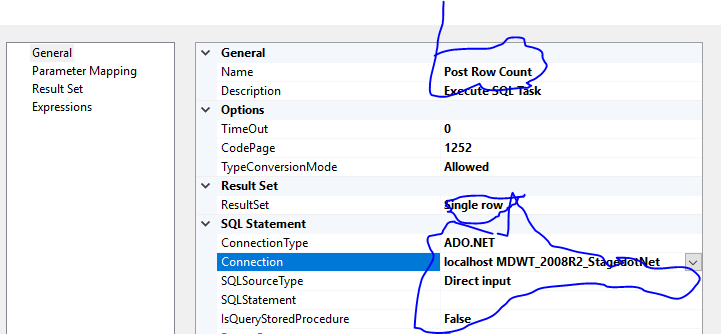


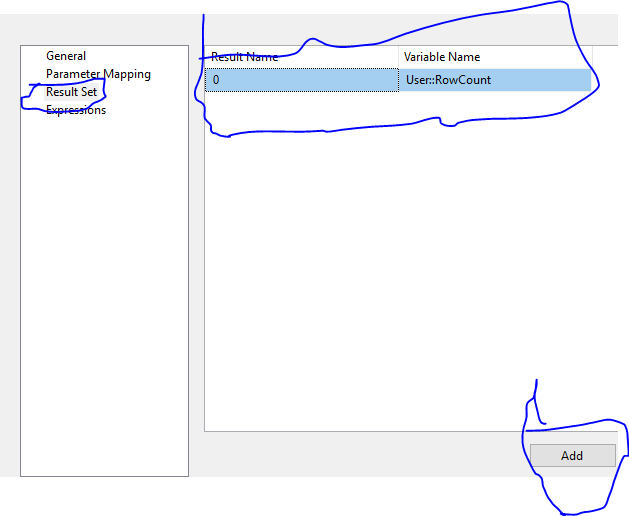


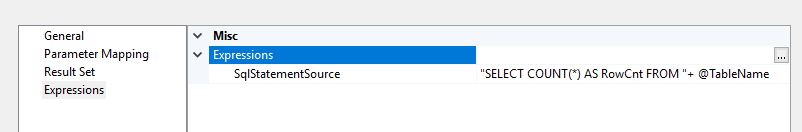




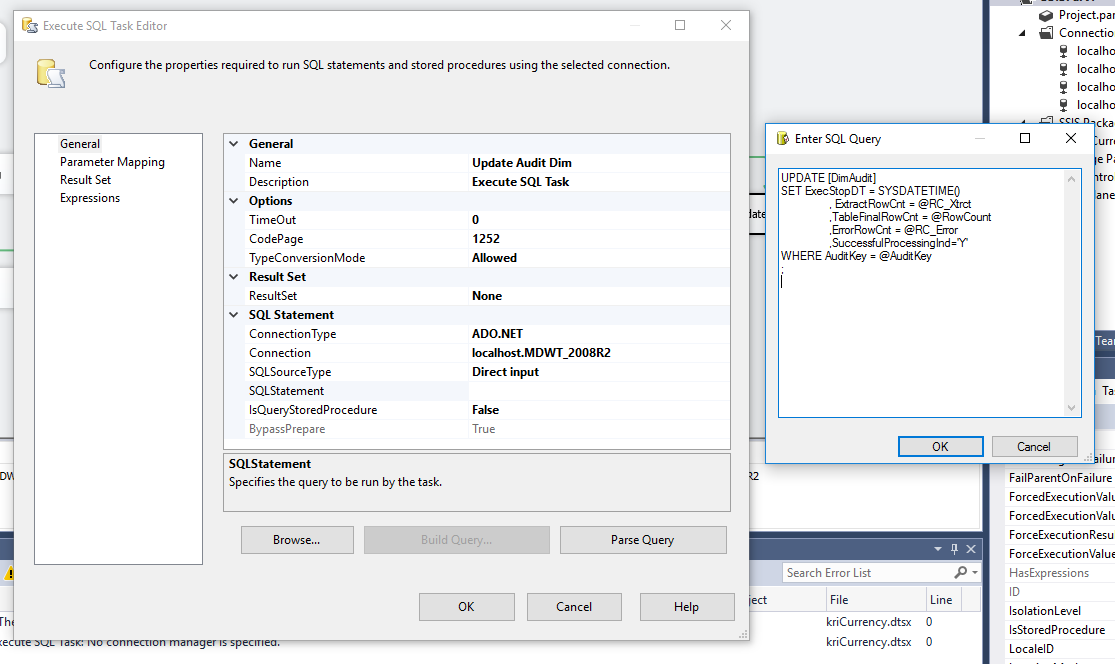
1. **Add an Execute Task List in the control flow and configure the following (This is used to count the rows that are added:**







1. **Add another Execute SQL Task to update the Audit Key**



Name: Update AuditDim row

ConnectionType: ADO.NET

Connection: localhost.MDWT\_2008R2

UPDATE [DimAudit]

SET ExecStopDT = SYSDATETIME()

, ExtractRowCnt = @RC\_Xtrct

,TableFinalRowCnt = @RowCount

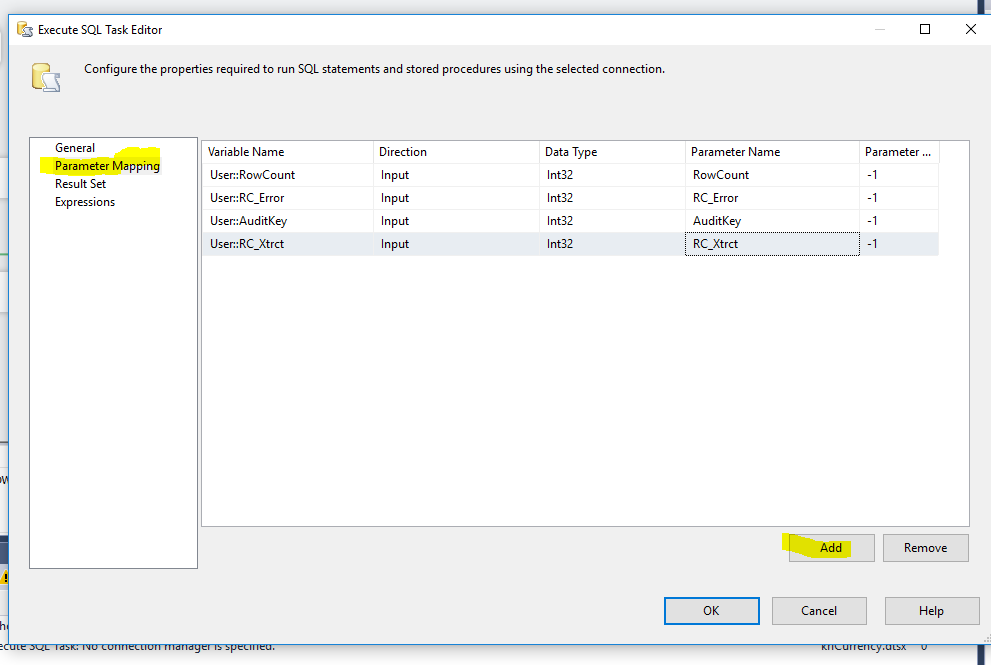
,ErrorRowCnt = @RC\_Error

,SuccessfulProcessingInd='Y'

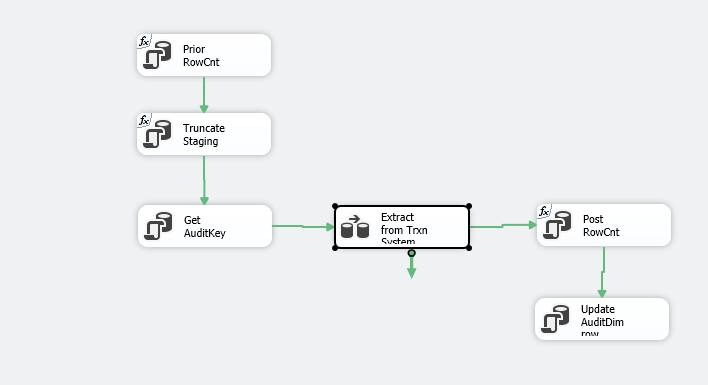
WHERE AuditKey = @AuditKey

;

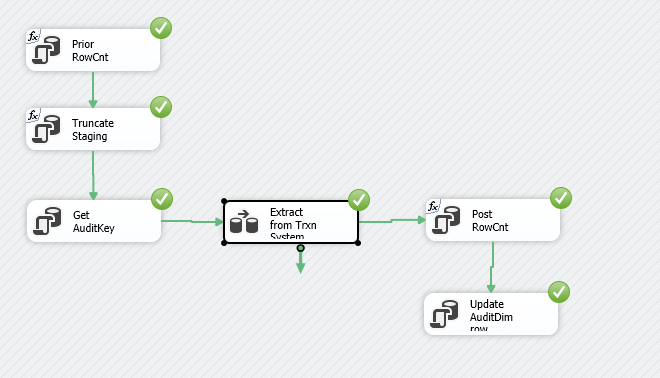
Update the Parameters



Your complete solution should look like this with all the control flow.



Run the package and show the results (screenshot)



## TASK 2

Repeat this for the StageProduct

## TASK 3

Using the solution provided, run the Master Package

# Extracting to Stage

You will need to run two sets of packages to populate the target database. The first solution, MDWT2008\_Extract, extracts the necessary tables from the source database AdventureWorks2008R2. This is a little silly in the context of a static sample application, but illustrates the best practice of breaking your ETL system into an “extract” piece and a “transform and load” piece, as discussed in the MDWT book.

To run the extract to the newly-created staging database, navigate to the file c:\MDWT\_Projects\MDWT2008\_Extract\MDWT2008\_Extract.sln. Double-click to launch BI Studio.

You should have stored these packages in the directory c:\MDWT\_Projects. If you put the packages somewhere other than in c:\MDWT\_Projects, simply edit the connections in the master package to point to the correct location.

Run the MASTER EXTRACT PACKAGE.

Once the package completes, take a look at the results and explore the solution. In SQL Server Management Studio, look at the audit table (SELECT \* FROM DimAudit).

## Packages in the MDWT2008\_Extract solution

The solution consists of the following packages, following the principle of creating one package per target table:

|  |  |  |
| --- | --- | --- |
| MASTER EXTRACT PACKAGE  Currency  CurrencyInUse  CurrencyRate  CustomerIndividual  Demographics  Department  Employee | EmployeeTerritory  Product  ProductCategory  ProductModel  ProductSubcategory  SalesOrderDetail SalesOrderHeader  SalesReason | SalesTerritories  SpecialOffer  Store  StoreContacts  StoreDemographics |

## Characteristics of the extract solution

The packages in this solution are extremely simple. The dimension tables are all structured the same:

* A few tasks to set up the load and the audit dimension
* A data flow task with no transformations in it, to copy data to the staging database
* A few tasks to clean up and update the audit dimension

Because the packages are so simple, it’s easy to see how similar they are. There are a few important things to learn from these packages:

* Keep each package as simple as possible.
* Keep packages consistent. Use one package as the template for the next. (Don’t forget to update the package GUID in the properties pane for the package.)
* Look at how the audit dimension is set up and populated. Design this once and you never need to touch it again.
* Look at the design pattern for loading data into a target table.
* Look at the SalesOrderDetail package for a different design pattern: to stop execution of a package if the package violates a business rule. In this case, the business rule is to stop processing if we extracted fewer than X rows or Y distinct products.

# Transform and Load

There is a second solution that contains packages to transform the extracted data in the staging database, and load it into the final DW dimension and fact tables. This solution is called MDWT\_2008, and you can bring it up in BIDS by double-clicking C:\MDWT\_Projects\MDWT2008\MDWT\_2008.sln.

To run the package, execute the master package RUN THIS TO LOAD ALL.

## Packages in the MDWT\_2008 solution

The solution consists of the following packages, following the principle of creating one package per target table:

|  |  |
| --- | --- |
| Package | Purpose |
| RUN THIS TO LOAD ALL | Master package |
| Currency | Load the currency dimension – very simple. |
| Customer | Load the customer dimension. Contains two data flows, one for individual customers and one for resellers. |
| Date | Load the date dimension from a .csv created from the Excel spreadsheet. You could easily define the package to read Excel directly, but we chose a .csv to reduce problems with Excel versions. |
| Employee | Load the employee (salesperson) dimension. The source query joins two staging tables, but a third table is brought in via an SSIS lookup. Chapter 7 in MDWT discusses the pros and cons of these alternative approaches. |
| OrderInfo | Load the order info dimension – very simple. |
| Product\_SSIS | Load the product dimension, with a simple source query and using SSIS lookups to bring in decodes and additional attributes. This package includes hard-coded decodes for quite a few attributes. These really should be stripped out to lookup transforms as well, in order to make them table-driven. |
| Product\_SQL | Load the product dimension, doing most of the work in SQL. This version is not called by the master package, but is included because Chapter 7 discusses this approach versus Product\_SSIS above. |
| Orders\_Lookups | Load the orders fact table, implementing the surrogate key pipeline in lookups. |
| Orders\_SQL | Load the orders fact table, implementing the surrogate key pipeline via a complex query in the Source. This version is not called by the master package. |
| ExchangeRates | Load the exchange rates fact table – very simple. |

# Further Exploration

a SQL Script (TruncateTables.sql) that will clear out fact and dimension tables so that you can start over.