**SSIS Design Pattern - Incremental Loads**

|  |
| --- |
| **Introduction**  Loading data from a data source to SQL Server is a common task. It's used in Data Warehousing, but increasingly data is being staged in SQL Server for non-Business-Intelligence purposes.  Maintaining data integrity is key when loading data into any database. A common way of accomplishing this is to truncate the destination and reload from the source. While this method ensures data integrity, it also loads a lot of data that was just deleted.  Incremental loads are a faster and use less server resources. Only new or updated data is touched in an incremental load.  **When To Use Incremental Loads**  Use incremental loads whenever you need to load data from a data source to SQL Server.  Incremental loads are the same regardless of which database platform or ETL tool you use. You need to detect new and updated rows - and separate these from the unchanged rows.  **Incremental Loads in Transact-SQL**  I will start by demonstrating this with T-SQL:  0. (Optional, but recommended) Create two databases: a source and destination database for this demonstration:  CREATE DATABASE [SSISIncrementalLoad\_Source]  CREATE DATABASE [SSISIncrementalLoad\_Dest]  1. Create a source named tblSource with the columns ColID, ColA, ColB, and ColC; make ColID is a primary unique key:  USE SSISIncrementalLoad\_Source  GO  CREATE TABLE dbo.tblSource  (ColID int NOT NULL  ,ColA varchar(10) NULL  ,ColB datetime NULL constraint df\_ColB default (getDate())  ,ColC int NULL  ,constraint PK\_tblSource primary key clustered (ColID))  2. Create a Destination table named tblDest with the columns ColID, ColA, ColB, ColC:  USE SSISIncrementalLoad\_Dest GO CREATE TABLE dbo.tblDest (ColID int NOT NULL  ,ColA varchar(10) NULL ,ColB datetime NULL ,ColC int NULL)  3. Let's load some test data into both tables for demonstration purposes:  USE SSISIncrementalLoad\_Source GO  -- insert an "unchanged" row INSERT INTO dbo.tblSource (ColID,ColA,ColB,ColC) VALUES(0, 'A', '1/1/2007 12:01 AM', -1)  -- insert a "changed" row INSERT INTO dbo.tblSource (ColID,ColA,ColB,ColC) VALUES(1, 'B', '1/1/2007 12:02 AM', -2)  -- insert a "new" row INSERT INTO dbo.tblSource (ColID,ColA,ColB,ColC) VALUES(2, 'N', '1/1/2007 12:03 AM', -3)  USE SSISIncrementalLoad\_Dest GO  -- insert an "unchanged" row INSERT INTO dbo.tblDest (ColID,ColA,ColB,ColC) VALUES(0, 'A', '1/1/2007 12:01 AM', -1)  -- insert a "changed" row INSERT INTO dbo.tblDest (ColID,ColA,ColB,ColC) VALUES(1, 'C', '1/1/2007 12:02 AM', -2)  4. You can view new rows with the following query:  SELECT s.ColID, s.ColA, s.ColB, s.ColC  FROM SSISIncrementalLoad\_Source.dbo.tblSource s LEFT JOIN SSISIncrementalLoad\_Dest.dbo.tblDest d ON d.ColID = s.ColID WHERE d.ColID IS NULL  This should return the "new" row - the one loaded earlier with ColID = 2 and ColA = 'N'. Why? The LEFT JOIN and WHERE clauses are the key. Left Joins return all rows on the left side of the join clause (SSISIncrementalLoad\_Source.dbo.tblSource in this case) whether there's a match on the right side of the join clause (SSISIncrementalLoad\_Dest.dbo.tblDest in this case) or not. If there is no match on the right side, NULLs are returned. This is why the WHERE clause works: it goes after rows where the destination ColID is NULL. These rows have no match in the LEFT JOIN, therefore they must be new.  This is only an example. You occasionally find database schemas that are this easy to load. Occasionally. Most of the time you have to include several columns in the JOIN ON clause to isolate truly new rows. Sometimes you have to add conditions in the WHERE clause to refine the definition of truly new rows.  Incrementally load the row ("rows" in practice) with the following T-SQL statement:  INSERT INTO SSISIncrementalLoad\_Dest.dbo.tblDest (ColID, ColA, ColB, ColC) SELECT s.ColID, s.ColA, s.ColB, s.ColC  FROM SSISIncrementalLoad\_Source.dbo.tblSource s LEFT JOIN SSISIncrementalLoad\_Dest.dbo.tblDest d ON d.ColID = s.ColID WHERE d.ColID IS NULL  5. There are many ways by which people try to isolate changed rows. The only sure-fire way to accomplish it is to compare each field. View changed rows with the following T-SQL statement:  SELECT d.ColID, d.ColA, d.ColB, d.ColC FROM SSISIncrementalLoad\_Dest.dbo.tblDest d INNER JOIN SSISIncrementalLoad\_Source.dbo.tblSource s ON s.ColID = d.ColID WHERE ( (d.ColA != s.ColA) OR (d.ColB != s.ColB)  OR (d.ColC != s.ColC) )  This should return the "changed" row we loaded earlier with ColID = 1 and ColA = 'C'. Why? The INNER JOIN and WHERE clauses are to blame - again. The INNER JOIN goes after rows with matching ColID's because of the JOIN ON clause. The WHERE clause refines the resultset, returning only rows where the ColA's, ColB's, ***or*** ColC's don't match ***and*** the ColID's match. This is important. If there's a difference in any ***or*** some ***or*** all the rows (except ColID), we want to update it.  *Extract-Transform-Load (ETL) theory has a lot to say about when and how to update changed data. You will want to pick up a good book on the topic to learn more about the variations.*  To update the data in our destination, use the following T-SQL:   UPDATE d SET d.ColA = s.ColA ,d.ColB = s.ColB ,d.ColC = s.ColC FROM SSISIncrementalLoad\_Dest.dbo.tblDest d INNER JOIN SSISIncrementalLoad\_Source.dbo.tblSource s ON s.ColID = d.ColID WHERE ( (d.ColA != s.ColA) OR (d.ColB != s.ColB)  OR (d.ColC != s.ColC) )  **Incremental Loads in SSIS**  Let's take a look at how you can accomplish this in SSIS using the Lookup Transformation (for the join functionality) combined with the Conditional Split (for the WHERE clause conditions) transformations.  Before we begin, let's reset our database tables to their original state using the following query:  USE SSISIncrementalLoad\_Source GO  TRUNCATE TABLE dbo.tblSource  -- insert an "unchanged" row INSERT INTO dbo.tblSource (ColID,ColA,ColB,ColC) VALUES(0, 'A', '1/1/2007 12:01 AM', -1)  -- insert a "changed" row INSERT INTO dbo.tblSource (ColID,ColA,ColB,ColC) VALUES(1, 'B', '1/1/2007 12:02 AM', -2)  -- insert a "new" row INSERT INTO dbo.tblSource (ColID,ColA,ColB,ColC) VALUES(2, 'N', '1/1/2007 12:03 AM', -3)  USE SSISIncrementalLoad\_Dest GO  TRUNCATE TABLE dbo.tblDest  -- insert an "unchanged" row INSERT INTO dbo.tblDest (ColID,ColA,ColB,ColC) VALUES(0, 'A', '1/1/2007 12:01 AM', -1)  -- insert a "changed" row INSERT INTO dbo.tblDest (ColID,ColA,ColB,ColC) VALUES(1, 'C', '1/1/2007 12:02 AM', -2)  Next, create a new project using Business Intelligence Development Studio (BIDS). Name the project SSISIncrementalLoad:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_00.png  Once the project loads, open Solution Explorer and rename Package1.dtsx to SSISIncrementalLoad.dtsx:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_01.png  When prompted to rename the package object, click the Yes button. From the toolbox, drag a Data Flow onto the Control Flow canvas:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_02.png  Double-click the Data Flow task to edit it. From the toolbox, drag and drop an OLE DB Source onto the Data Flow canvas:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_03.png  Double-click the OLE DB Source connection adapter to edit it:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_04.png  Click the New button beside the OLE DB Connection Manager dropdown:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_05.png  Click the New button here to create a new Data Connection:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_06.png  Enter or select your server name. Connect to the SSISIncrementalLoad\_Source database you created earlier. Click the OK button to return to the Connection Manager configuration dialog. Click the OK button to accept your newly created Data Connection as the Connection Manager you wish to define. Select "dbo.tblSource" from the Table dropdown:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_07.png  Click the OK button to complete defining the OLE DB Source Adapter.  Drag and drop a Lookup Transformation from the toolbox onto the Data Flow canvas. Connect the OLE DB connection adapter to the Lookup transformation by clicking on the OLE DB Source and dragging the green arrow over the Lookup and dropping it. Right-click the Lookup transformation and click Edit (or double-click the Lookup transformation) to edit:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_08.png  When the editor opens, click the New button beside the OLE DB Connection Manager dropdown (as you did earlier for the OLE DB Source Adapter). Define a new Data Connection - this time to the SSISIncrementalLoad\_Dest database. After setting up the new Data Connection and Connection Manager, configure the Lookup transformation to connect to "dbo.tblDest":  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_09.png  Click the Columns tab. On the left side are the columns currently in the SSIS data flow pipeline (from SSISIncrementalLoad\_Source.dbo.tblSource). On the right side are columns available from the Lookup destination you just configured (from SSISIncrementalLoad\_Dest.dbo.tblDest). Follow the following steps:  1. We'll need all the rows returned from the destination table, so check all the checkboxes beside the rows in the destination. We need these rows for our WHERE clauses and for our JOIN ON clauses.  2. We do *not* want to map all the rows between the source and destination - we only want to map the columns named ColID between the database tables. The Mappings drawn between the Available Input Columns and Available Lookup Columns define the JOIN ON clause. Multi-select the Mappings between ColA, ColB, and ColC by clicking on them while holding the Ctrl key. Right-click any of them and click "Delete Selected Mappings" to delete these columns from our JOIN ON clause.  3. Add the text "Dest\_" to each column's Output Alias. These rows are being appended to the data flow pipeline. This is so we can distinguish between Source and Destination rows farther down the pipeline:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_10.png  Next we need to modify our Lookup transformation behavior. By default, the Lookup operates as an INNER JOIN - but we need a LEFT (OUTER) JOIN. Click the "Configure Error Output" button to open the "Configure Error Output" screen. On the "Lookup Output" row, change the Error column from "Fail component" to "Ignore failure". This tells the Lookup transformation "If you don't find an INNER JOIN match in the destination table for the Source table's ColID value, don't fail." - which also effectively tells the Lookup "Don't act like an INNER JOIN, behave like a LEFT JOIN":  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_11.png  Click OK to complete the Lookup transformation configuration.  From the toolbox, drag and drop a Conditional Split Transformation onto the Data Flow canvas. Connect the Lookup to the Conditional Split as shown. Right-click the Conditional Split and click Edit to open the Conditional Split Editor:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_12.png  Expand the NULL Functions folder in the upper right of the Conditional Split Transformation Editor. Expand the Columns folder in the upper left side of the Conditional Split Transformation Editor. Click in the "Output Name" column and enter "New Rows" as the name of the first output. From the NULL Functions folder, drag and drop the "ISNULL( <<expression>> )" function to the Condition column of the New Rows condition:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_13.png  Next, drag Dest\_ColID from the columns folder and drop it onto the "<<expression>>" text in the Condition column. "New Rows" should now be defined by the condition "ISNULL( [Dest\_ColID] )". This defines the WHERE clause for new rows - setting it to "WHERE Dest\_ColID Is NULL".  Type "Changed Rows" into a second Output Name column. Add the expression "(ColA != Dest\_ColA) || (ColB != Dest\_ColB) || (ColC != Dest\_ColC)" to the Condition column for the Changed Rows output. This defines our WHERE clause for detecting changed rows - setting it to "WHERE ((Dest\_ColA != ColA) OR (Dest\_ColB != ColB) OR (Dest\_ColC != ColC))". Note "||" is used to convey "OR" in SSIS Expressions:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_14.png  Change the "Default output name" from "Conditional Split Default Output" to "Unchanged Rows":  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_15.png  Click the OK button to complete configuration of the Conditional Split transformation.  Drag and drop an OLE DB Destination connection adapter and an OLE DB Command transformation onto the Data Flow canvas. Click on the Conditional Split and connect it to the OLE DB Destination. A dialog will display prompting you to select a Conditional Split Output (those outputs you defined in the last step). Select the New Rows output:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_16.png  Next connect the OLE DB Command transformation to the Conditional Split's "Changed Rows" output:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_17.png  Your Data Flow canvas should appear similar to the following:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_18.png  Configure the OLE DB Destination by aiming at the SSISIncrementalLoad\_Dest.dbo.tblDest table:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_19.png  Click the Mappings item in the list to the left. Make sure the ColID, ColA, ColB, and ColC source columns are mapped to their matching destination columns (aren't you glad we prepended "Dest\_" to the destination columns?):  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_20.png  Click the OK button to complete configuring the OLE DB Destination connection adapter.  Double-click the OLE DB Command to open the "Advanced Editor for OLE DB Command" dialog. Set the Connection Manager column to your SSISIncrementalLoad\_Dest connection manager:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_21.png  Click on the "Component Properties" tab. Click the elipsis (button with "...") beside the SQLCommand property:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_22.png  The String Value Editor displays. Enter the following parameterized T-SQL statement into the String Value textbox:  UPDATE dbo.tblDest SET ColA = ? ,ColB = ? ,ColC = ? WHERE ColID = ?  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_23.png  The question marks in the previous parameterized T-SQL statement map by ordinal to columns named "Param\_0" through "Param\_3". Map them as shown below - effectively altering the UPDATE statement *for each row* to read:  UPDATE SSISIncrementalLoad\_Dest.dbo.tblDest SET ColA = SSISIncrementalLoad\_Source.dbo.ColA ,ColB = SSISIncrementalLoad\_Source.dbo.ColB ,ColC = SSISIncrementalLoad\_Source.dbo.ColC WHERE ColID = SSISIncrementalLoad\_Source.dbo.ColID  *Note the query is executed on a row-by-row basis. For performance with large amounts of data, you will want to employ set-based updates instead.*  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_24.png  Click the OK button when mapping is completed.  Your Data Flow canvas should look like that pictured below:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_25.png  If you execute the package with debugging (press F5), the package should succeed and appear as shown here:  http://vsteamsystemcentral.com/images/ext/SSISIncrementalLoad_26.png  Note one row takes the "New Rows" output from the Conditional Split, and one row takes the "Changed Rows" output from the Conditional Split transformation. Although not visible, our third source row doesn't change, and would be sent to the "Unchanged Rows" output - which is simply the default Conditional Split output renamed. Any row that doesn't meet any of the predefined conditions in the Conditional Split is sent to the default output.  That's all! Congratulations - you've built an incremental database load! [:)] |