**Combining CDC and SSIS for Incremental Data Loads**

**New SSIS functionality makes the process straightforward**

As the DBA at the XYZ Company, you've been assigned the task of creating a report that shows sales order data. The data can be obtained from your order tracking application, but this system needs to be highly available and shouldn't be hit with large queries during the day when orders are being processed. The solution that makes the most sense is to create a copy of the database that can be used for reporting, but there are millions of records on these tables and even a nightly load won't complete in the available maintenance window.

**Related:** [Transforming Data Using SSIS and SQL](http://sqlmag.com/sql-server-integration-services/transforming-data-using-ssis-and-sql)

One challenge in feeding data from an OLTP database to an operational data store (ODS) is keeping the data in sync. The target tables in the ODS can be entirely reloaded from the source on a regular schedule, but this has performance implications on the source and target databases. Tracking incremental changes can be accomplished using triggers or marking records, but these solutions can also affect performance and are usually quite complex.

The change data capture (CDC) feature introduced in SQL Server 2008 provides an efficient framework for tracking inserts, updates, and deletes in tables in a SQL Server database. CDC has a performance cost, but it's less than other change-tracking solutions such as triggers. When CDC is enabled, the changes are written to tables in the database and can be accessed using CDC system stored procedures. Note that CDC isn't supported on system databases.

By using both SQL Server Integration Services (SSIS) and CDC, you can perform incremental loads to an ODS. This process has been available since SQL Server 2008, but with the release of SQL Server 2012, there are new SSIS tasks and components that make the process less complex. As in SQL Server 2008, data is retrieved from the CDC tables using log sequence numbers (LSNs), but the new CDC Control tasks in SSIS make it much easier to track the range of LSNs needed for processing the changes. Once configured, the CDC source transparently uses the CDC stored procedures to pull the changed data.

Given the complexity of what's being accomplished, using CDC and SSIS to migrate incremental changes is amazingly straightforward. You start by enabling CDC on the database and the tables you want to track. Next, you use an SSIS package to do a one-time load of existing data into your ODS table. Then, you use another SSIS package to bring over incremental changes using the new CDC task and CDC source. This package can be run at the desired interval to get changes into the ODS tables. Depending on your reporting needs, this could be every five minutes, once a day, or anything in between. Let's take a closer look at each of these steps.

**Enabling CDC**

Enabling CDC on a SQL Server database (Enterprise edition only) is done by executing sys.sp\_cdc\_enable\_db. Running this system stored procedure requires membership in the sysadmin server role. For example, if you want to enable CDC on the [AdventureWorks2012](http://msftdbprodsamples.codeplex.com/releases/view/55330) database, you'd run the code

USE AdventureWorks2012  
GO  
EXEC sys.sp\_cdc\_enable\_db;  
GO

When CDC is enabled on a database, is\_cdc\_enabled is set to True for that database in the sys.databases table.

After enabling CDC, a new cdc system schema is created containing the tables and stored procedures used to track changes. You can view the cdc schema items in the System Tables and System Stored Procedures folders in SQL Server Management Studio (SSMS).

Any tables you want to track need to be enabled, which you accomplish with the sys.sp\_cdc\_enable\_table stored procedure. For example, suppose you want to load changes for the SalesOrderHeader table in the AdventureWorks2012 database into an ODS table. To do so, you enable the SalesOrderHeader table with code such as

USE AdventureWorks2012  
GO  
EXEC sys.sp\_cdc\_enable\_table  
@source\_schema = N'Sales',  
@source\_name = N'SalesOrderHeader',  
@role\_name = NULL,  
@supports\_net\_changes = 1;  
GO

The sp\_cdc\_enable\_table stored procedure has many parameters you can use when enabling a table for CDC. For a basic setup, you only need to set three parameters: source\_schema, source\_name, and role\_name. The source\_name and source\_schema parameters specify the name of the table to track and the schema to which that table belongs, respectively.

The role\_name parameter assigns a database security role, which will be used to allow access to the change data. As this example shows, you can explicitly set this parameter to NULL when you don't want to use a role to limit access to the change data. You can also use the role\_name parameter to create a role if it doesn't already exist. If you specify a database security role that currently exists, it's used. If the role doesn't exist, SQL Server will create a database role using the specified name. Best practice is to create a role and explicitly allow access to the service account running SSIS.

In the code to enable the SalesOrderHeader table, note the inclusion of the supports\_net\_changes parameter. It allows querying only net changes using the cdc.fn\_cdc\_get\_net\_changes\_<capture\_instance> function. Enabling this parameter adds an additional nonclustered index to the CDC table, so there might be performance considerations.

Two other sp\_cdc\_enable\_table parameters -- captured\_column\_list and filegroup\_name -- are also worth noting, because they can affect performance. You can use the captured\_column\_list parameter to limit the columns being tracked. Fewer columns means that fewer changes need to be tracked and less storage space will be used. This also allows for less I/O when processing the change data to a new target. You can use the filegroup\_name parameter to assign a file group for the change data. If this parameter isn't used, the change data is stored on the default file group, which is typically PRIMARY. Microsoft recommends that the change data be kept on a separate file group from the data being tracked if possible.

After the first table is enabled, two new SQL Server Agents jobs are created: cdc.<databasename>\_capture (e.g., cdc.AdventureWorks2012\_capture) and cdc.<databasename>\_cleanup (e.g., cdc.AdventureWorks2012\_cleanup). The capture job runs every 5 seconds by default to capture changes. The cleanup job runs daily by default to clear out the tables so they don't grow too large. The default setting allows for three days of rolling change data before cleaning it out.

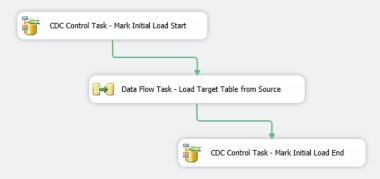
The change data for each table is stored in a separate system table. System tables follow the naming convention cdc.<capturename>\_CT (e.g., cdc.Sales\_SalesOrderHeader\_CT). Each table stores all the columns for each record that's changed (assuming you didn't use the captured\_column\_list parameter) along with five additional columns:

* \_\_$start\_lsn. This column stores the LSNs of the commit transactions that changed the records.
* \_\_$end\_lsn. The values in this column are always null. This column exists only for possible future enhancements.
* \_\_$operation. This column specifies the type of change (1 = delete, 2 = insert, and 4 = update) made to each record.
* \_\_$update\_mask. The values stored in this column are bit masks that indicate which columns have been changed. For an insert or delete record, the bit mask would be all ones because every column changed.
* \_\_$seqval. If more than one change occurred in same transaction, the values in this column are used to order the changes.

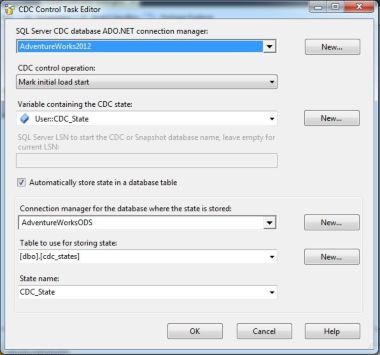
If you've ever tried to create a solution for tracking changes in a database, you can appreciate how powerful these few commands are to set up CDC on a SQL Server database. You can quickly enable CDC on all the OLTP database tables that you want to track. There are some performance considerations, but as I stated previously, they should be minimal compared with other solutions. The Microsoft article "[Tuning the Performance of Change Data Capture in SQL Server 2008](http://msdn.microsoft.com/en-us/library/dd266396%28v=sql.100%29.aspx)" discusses best practices that you can follow to ensure good performance when using CDC. Although it's written for SQL Server 2008, the best practices can be used with SQL Server 2012 as well.

**Performing the Initial Data Load**

After you enable CDC on the tables you want to track, you use an SSIS package to do a one-time load of your existing data into your ODS table. As Figure 1 shows, this package consists of three tasks. The first task is a CDC Control task, which is new in SSIS 2012. In this case, you configure the task to specify the CDC state for the initial load. The CDC state is stored within an SSIS variable, and the variable's value is persisted to a table in the target database, as shown in Figure 2. By persisting this value, you can source only the unprocessed changes on subsequent runs.

[[](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig1-lg.jpg)](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig1-lg.jpg" \o "Donnelly-SQL2338-Fig1-lg" \t "_blank)

**Figure 1:** Examining the tasks in the SSIS package for the initial data load

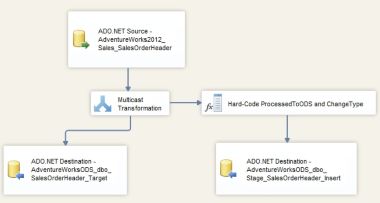
[[](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig2-lg.jpg)](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig2-lg.jpg" \o "Donnelly-SQL2338-Fig2-lg" \t "_blank)

**Figure 2:** Configuring a CDC Control task to specify the CDC state

A nice feature of the CDC Control task is that it lets you create a new user variable (CDC\_State) and a new table (cdc\_states) without leaving the task editor. The CDC\_State variable is a string variable that should be left blank. It will be automatically populated by the CDC Control task.

The second task is a Data Flow task that inserts all the records from the source table into the target table. For this example, I created a database named AdventureWorksODS and wrote a script that creates the SalesOrderHeader table in it. In this target table, I included all the fields I wanted to duplicate and removed the derived columns and the identity property from the key field. I also wrote a script that creates three staging tables: one for inserts, one for updates, and one for deletes. In addition to containing columns for the source data, the staging tables have the ChangeType, ChangeDate (default = GetDate()), and ProcessedToODS columns. You can download the scripts that create the target table (createTarget.sql) and the staging tables (createStaging.sql) by clicking the Download button.

When you perform the incremental load, you need to mark the records on the staging table to retain the changes for historical tracking. To keep the history for the initial load, the Data Flow task loads both the target table (SalesOrderHeader\_Target) and the insert table (Stage\_SalesOrderHeader\_Insert). As Figure 3 shows, this is accomplished using a Multicast transformation. I added a Derived Column transformation to hard-code the ChangeType value to 2 (insert operation) and hard-code the ProcessedToODS value to 1 for all records in the Stage\_SalesOrderHeader\_Insert table.

[[](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig3-lg.jpg)](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig3-lg.jpg" \o "Donnelly-SQL2338-Fig3-lg" \t "_blank)

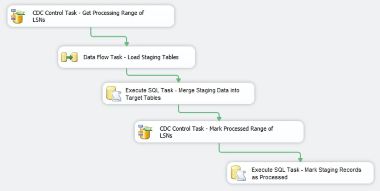
**Figure 3:** Using a Multicast transformation to mark the records

The last task is another CDC Control task. It updates the CDC state again to indicate that the initial load is complete.

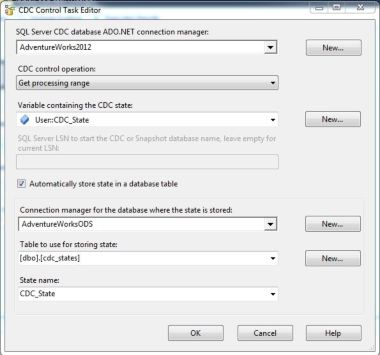
You're now done setting up the first SSIS package. When run, it will migrate all the data from SalesOrderHeader into the new ODS table.

**Performing the Incremental Load**

A second SSIS package is used to perform the incremental load. As Figure 4 shows, this package consists of five tasks. The first one is a CDC Control task that's used to get the processing range, as shown in Figure 5. The processing range isn't a time range but rather a range of LSNs that contain the changes to be loaded. In the initial load package, the maximum LSN was written into the CDC\_State variable and then persisted in the cdc\_states table created in AdventureWorksODS. This marks where the CDC Source task needs to start reading changes. When the CDC Control task runs, the end LSN is obtained using the same logic as the sys.fn\_cdc\_get\_max\_lsn function. The end LSN value is stored in the CDC state table and used for the CDC source in the Data Flow task.

[[](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig4-lg.jpg)](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig4-lg.jpg" \o "Donnelly-SQL2338-Fig4-lg" \t "_blank)

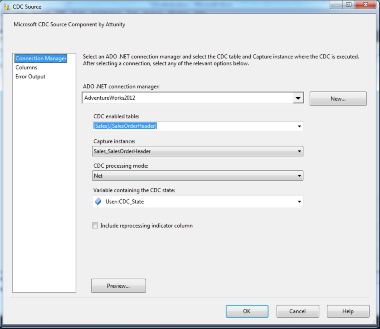
**Figure 4:** Examining the tasks in the SSIS package for the incremental load

[[](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig5-lg.jpg)](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig5-lg.jpg" \o "Donnelly-SQL2338-Fig5-lg" \t "_blank)

**Figure 5:** Configuring a CDC Control task to get the processing range

The second task in the incremental load package is a Data Flow task that loads the staging tables. It uses a CDC source as the first component. An ADO.NET connection to the source database (AdventureWorks2012) must be used with the CDC source.

After the connection is selected, you can add a CDC-enabled table to the source. To get only net changes, you must select Net from the *CDC processing mode* drop-down list, as seen in Figure 6. The Net option is only available if you used the *supports\_net\_changes = 1* parameter when running the sys.sp\_cdc\_enable\_table stored procedure to enable CDC on the table. The CDC source pulls the data in the same manner as the sys.fn\_cdc\_get\_net\_changes\_<capture\_instance> function.

[[](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig6-lg.jpg)](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig6-lg.jpg" \o "Donnelly-SQL2338-Fig6-lg" \t "_blank)

**Figure 6:** Configuring the CDC source to get only net changes

The from\_lsn and to\_lsn values are parsed out of the CDCState value and used to determine the range of changes that will be pulled from the cdc.Sales\_SalesOrderHeader\_CT table. (CDCState is a string value that's used throughout the package.)

The data retrieved by the CDC source is fed into a CDC splitter so that inserts, deletes, and updates are handled separately, as shown in Figure 7. The CDC splitter is a customized Conditional Split task that uses the \_$operation field to send different operations to different outputs. In this example, each of the splitter's outputs is sent to a separate ADO.NET data destination. If you use a different type of data destination (e.g., OLE DB, ODBC) you might run into data typing problems because the CDC source is required to use an ADO.NET connection manager.

[[](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig7-lg.jpg)](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig7-lg.jpg" \o "Donnelly-SQL2338-Fig7-lg" \t "_blank)

**Figure 7:** Using a Conditional Split task to load the staging tables

There's no configuration needed for the CDC splitter, because it automatically creates three separate outputs for the different operations (insert, update, and delete). Each of the data flow destinations points to the corresponding staging table in the AdventureWorksODS database. In addition to the source data, there are three fields available from the CDC source: \_\_$start\_lsn, \_\_$operation, and \_\_$update\_mask. You could record any of these columns for historical purposes, but for this example, let's map the \_$operation column from the CDC source to the ChangeType column in the staging tables (1=delete, 2 = insert, and 4= update). Once the staging tables are loaded, the Data Flow task is complete.

The next task in the incremental load package is an Execute SQL task. It executes the merge.sql script to merge the unprocessed staging records into the target table. You can download this script by clicking the Download button.

To finish the merge process, a CDC Control task marks the LSNs that have been processed. You configure this task by selecting the *Mark processed range* option in the *CDC control operation* drop-down list. This updates the persisted CDC state in the cdc\_states table so that when the package runs the next time, the starting LSN can be calculated.

The last task in the incremental load package is an Execute SQL task. It runs the following code to mark the staging records as processed in the ODS target table:

UPDATE Stage\_SalesOrderHeader\_Insert SET ProcessedToODS = 1 WHERE ProcessedToODS IS NULL;

UPDATE Stage\_SalesOrderHeader\_Update SET ProcessedToODS = 1 WHERE ProcessedToODS IS NULL;

UPDATE Stage\_SalesOrderHeader\_Delete SET ProcessedToODS = 1 WHERE ProcessedToODS IS NULL;

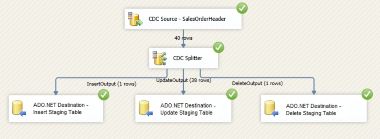
To test the incremental load package, follow these steps:

1. Run the initial load package.

2. Execute a query that inserts, deletes, and updates some records in the SalesOrderHeader table. You can view these changes by looking at the contents of the system table cdc.Sales\_SalesOrderHeader\_CT.

3. Run the incremental load package.

As Figure 8 shows, you'll be able to see that the changes were processed to the correct staging tables. In addition, in the last Execute SQL task in the incremental load package, you'll see that all the records are marked as processed in the ODS target table. This allows you to retain the history of your incremental load. The merge.sql script that ran in the other Execute SQL task uses a filter of ProcessedToODS = 0 on future runs to bring over only new changes.

[[](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig8-lg.jpg)](http://sqlmag.com/content/content/143311/Donnelly-SQL2338-Fig8-lg.jpg" \o "Donnelly-SQL2338-Fig8-lg" \t "_blank)

**Figure 8:** Reviewing the test results

**Solution Works for Tables of All Sizes**

The example I presented here is only a simple demonstration of how you can use CDC and SSIS together to load incremental changes into a target database. I've successfully implemented this solution on tables that have millions of records; every minute the changes are processed into an ODS database that's used for reporting and secondary applications. If you use CDC with SSIS 2012, the implementation of such a solution is remarkably straightforward.