CDW: Guide to Incremental Extracts

Issue 2; Draft D

Thursday, September 07, 2017

Presented by:

Adams, Joseph K.(Ken)

Department of Veterans Affairs

Business Intelligence Service Line (BISL)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Major Changes** |
| Issue 1; Final | Fackler, David O. | 9/13/13 | Original Document(section in [CDW Guide: Projects and Development](https://vaww.cdw.va.gov/support/Shared%20Documents/Best%20Practices/CDW%20Guide%20Projects%20and%20Development.docx)) |
| Issue 2; Draft A | Adams, Joseph K. | 6/16/15 | Carved out of Dave’s original document and expounded on it |
| Issue 2; Draft B | Adams, Joseph K. | 6/26/15 | Made changes based on group feedback |
| Issue 2; Draft C | Adams, Joseph K. | 7/13/15 | Added complex scenario |
| Issue 2; Final | Adams, Joseph K. | 7/29/15 | Add valid SharePoint links |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[Revision History i](#_Toc425934692)

[Table of Contents ii](#_Toc425934693)

[Introduction 1](#_Toc425934694)

[Methods of Extracting Data 1](#_Toc425934695)

[Concept of DW Batches 1](#_Toc425934696)

[ExtractBatch View 2](#_Toc425934697)

[Incremental Load Scenarios 3](#_Toc425934698)

[Local Project Database Example 3](#_Toc425934699)

[Setup 3](#_Toc425934700)

[Script 4](#_Toc425934701)

[OpCode 5](#_Toc425934702)

[Remote Project Database Example 5](#_Toc425934703)

[Setup 6](#_Toc425934704)

[Script 7](#_Toc425934705)

[SSIS Package 7](#_Toc425934706)

[Complex Load Scenario 11](#_Toc425934707)

[Script 13](#_Toc425934708)

[Exclusions 16](#_Toc425934709)

[Recommendations 16](#_Toc425934710)

# Introduction

The recommended method of extracting data from data warehouse (DW) databases, from any given DW server, is in an incremental fashion. This means that a downstream development team pulls only rows that have changed or are new since the last extraction. Extracting the data in an incremental fashion means that a team is not pulling the same data repeatedly and reduces the overall load on the server(s). This document will discuss how to extract data incrementally across all levels of the enterprise - VISN data warehouses (VDW); regional data warehouses (RDW) and corporate data warehouses (CDW).

# Methods of Extracting Data

Downstream development teams will either be extracting and loading data on the same server or extracting data from one server and loading to another server.

* If the destination database is on the same server as the source database, the extract can be done using T-SQL commands. The preferred method is to ingest the data in smaller batches using a cursor. Later in the document, we will chronicle how this should be done and provide actual snippets of code. [This code is available from a CDW SharePoint link.](https://vaww.cdw.va.gov/support/Training/Shared%20Documents/Forms/AllItems.aspx?RootFolder=/Support/Training/Shared%20Documents/All%20Day%20Training%20Presentations/2015-07-29_Region03_OptimizingCDW_RDW_Processes/Incremental%20Load%20Samples&FolderCTID=0x01200004315B0DA48E3047A544EE49A5DBCA2E&View=%7bB66924E8-353B-4D57-BA74-539F63FCF067%7d)
* If the source and destination databases are on two separate servers, the extract can be done using an SSIS package. The preferred method is to again ingest the data in smaller batches using a ForEach Loop inside of SSIS. Later in the document, we will chronicle how this should be done and provide both snippets and screenshots of the code. [This code is available from a CDW SharePoint link as well.](https://vaww.cdw.va.gov/support/Training/Shared%20Documents/Forms/AllItems.aspx?RootFolder=/Support/Training/Shared%20Documents/All%20Day%20Training%20Presentations/2015-07-29_Region03_OptimizingCDW_RDW_Processes/Incremental%20Load%20Samples&FolderCTID=0x01200004315B0DA48E3047A544EE49A5DBCA2E&View=%7bB66924E8-353B-4D57-BA74-539F63FCF067%7d)

# Concept of DW Batches

Data is extracted on a regular basis from medical centers and other sources. That data is transformed and loaded into a consistent data model. The initial data ingestion takes place at the regional level, or RDW. When that data is loaded into the RDW environments in each region, changes to any existing table (inserts and updates) are loaded in batches, with each batch of data assigned a unique identifier called the ETLBatchID. Every row of data affected when a batch of changes is applied to a table and tagged with the ETLBatchID value assigned to the batch. Based on this process, any given batch of data from any given table can be retrieved by querying for the particular ETLBatchID value assigned to that batch.

Within each region, ETLBatchID values are assigned within specific ranges, just like the surrogate ID values (or SID values) assigned within each table. The following shows the ranges used in each region:

|  |  |  |
| --- | --- | --- |
| **Region** | **Start of ETLBatchID Range** | **End of ETLBatchID Range** |
| R01 | 800000000 | 999999999 |
| R02 | 1000000000 | 1199999999 |
| R03 | 1200000000 | 1399999999 |
| R04 | 1400000000 | 1599999999 |

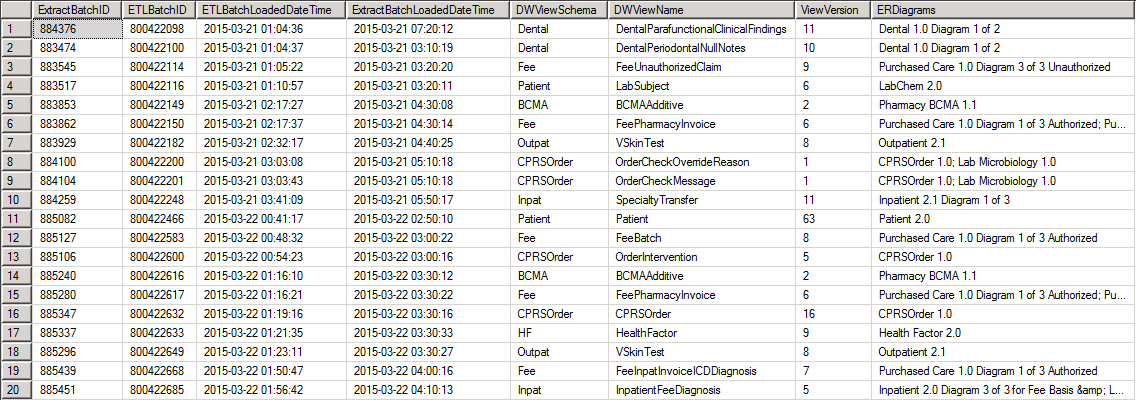
Data loaded into the RDW environments is then distributed to all CDW environments and to the VDW environments within that region. As batches of data are distributed, the ETLBatchID values are kept in place so that rows tagged with a given ETLBatchID are the same in every environment.

# ExtractBatch View

At the CDW level, an additional batch ID value is assigned to better support incremental extracts. If projects developing downstream systems have to rely on using ETLBatchID values only, they would be required to keep up with the last ETLBatchID value processed from each region. To make that easier, at the CDW level, the regional ETLBatchID values are “coalesced” into a single range of values. This new identifier is called the **ExtractBatchID**. A cross-reference view available in the SPV database (named EB.ExtractBatch) can be used to translate ETLBatchID values into ExtractBatchID values.

Previously, this view was present on the CDW level only. Now it has been pushed out to the various other DW environments in order to synchronize how teams are pulling data incrementally. In addition, the newest version of EB.ExtractBatch now contains the name of the view. Having that information allows a team to segregate batches related to specific views e.g., Appt.Appointment, Dim.Location, Outpat.Visit, and so on. Prior to having this available, an expensive join between the EB.ExtractBatch view and the source view was required.

The key to extracting data incrementally is to exploit the ExtractBatchID and DWViewName values found in the ExtractBatch view. Here is a sample of that view:



When extracting data incrementally, the general idea is to extract data where ExtractBatchID values fora view are greater than what has previously been extracted for that view. This requires some mechanism for keeping track of the last maximum ExtractBatchID value that was successfully loaded during the prior extract. A simple table design that contains this information can be easily implemented and updated as each incremental extract is completed. This type of extract is best explained via a few examples contained in this document.

# Incremental Load Scenarios

## Local Project Database Example

In this example, we will assume that a project database has been set up on one of the CDW instances. The team needs to extract data from the TIU.TIUDocument view inside of the SPV database on that CDW server. This example reflects the method to do an incremental extract using the ExtractBatchID value.

### Setup

The first step for the team is to determine what columns are needed from TIUDocument. After doing so, they will need to create a production table named Dflt.ProdTIUDocument. In addition, they will create a table to keep track of the last ExtractBatchID processed during an incremental extract. The T-SQL for creating that table is below:

CREATE TABLE [Dflt].[ExtractBatchLog](

[DWViewName] [varchar](100) NULL,

[LastExtractBatchID] [int] NULL,

[LastExtractDateTime] [datetime] NULL DEFAULT (getdate())

) ON [DefFG]

Once the team has created the Dflt.ProdTIUDocument and Dflt.ExtractBatchLog tables they are ready to create the extract script. The T-SQL script contains a cursor that will be used to loop over the ExtractBatchID value for TIUDocument. The flow of the script is as follows:

* Declare a cursor named BatchCursor to house the ExtractBatchID values and related ETLBatchID values of interest for TIUDocument
  + Populated by querying EB.ExtractBatch view in the SPV database
    - Filter for DWViewName = ‘TIUDocument’
    - Filter for ExtractBatchID > the LastExtractBatchID processed for TIUDocument
  + Loop over each ExtractBatchID and grab TIUDocument records for each associated ETLBatchID
* For each ExtractBatchID value returned:
  + Query and merge the TIUDocument records for a given ETLBatchID into the Dflt.ProdTIUDocument table
  + Finally, update the Dflt.ExtractBatchLog table recording the ExtractBatchID as the last handled by the extract process

### Script

The cursor example is seen below. (Please note this does not include business specific logic such as filtering based on a range of dates, list of Sta3ns, etc.)

**T-SQL has been condensed in order to have it fit here – it stretches over the next couple of pages:**

--declare batch variables and cursor

DECLARE @ETLBatchID AS INT, @ExtractBatchID AS INT;

DECLARE @DWViewName AS VARCHAR(100) = 'TIUDocument';

DECLARE BatchCursor CURSOR STATIC

FOR

--get batch ids

SELECT EX.ETLBatchID, EX.ExtractBatchID

FROM SPV.EB.ExtractBatch AS EX

INNER JOIN Dflt.ExtractBatchLog AS EL ON

EX.DWViewName = EL.DWViewName

WHERE EX.DWViewName = @DWViewName

AND EX.ExtractBatchID > EL.LastExtractBatchID

ORDER BY EX.ExtractBatchID;

OPEN BatchCursor;

FETCH NEXT FROM BatchCursor INTO @ETLBatchID, @ExtractBatchID;

WHILE @@FETCH\_STATUS = 0

BEGIN

--merge data from view into production(destination) table

MERGE

Dflt.ProdTIUDocument AS D

USING

(

SELECT TIUDocumentSID, TIUDocumentIEN, Sta3n --and so on...

FROM SPV.TIU.TIUDocument

WHERE ETLBatchID = @ETLBatchID

) AS M

ON

D.TIUDocumentSID = M.TIUDocumentSID

WHEN NOT MATCHED THEN

INSERT

(

TIUDocumentSID, TIUDocumentIEN, Sta3n --and so on...

)

VALUES

(

TIUDocumentSID, TIUDocumentIEN, Sta3n, --and so on...

)

WHEN MATCHED THEN

UPDATE SET

[TIUDocumentSID] = m.[TIUDocumentSID]

,[TIUDocumentIEN] = m.[TIUDocumentIEN]

,[Sta3n] = m.[Sta3n]

--and so on...

;

--after merging batch then update the ExtractBatchLog table

UPDATE Dflt.ExtractBatchLog

SET LastExtractBatchID = @ExtractBatchID, LastExtractDateTime = GETDATE()

WHERE DWViewName = @DWViewName;

--grab the next batch for processing

FETCH NEXT FROM BatchCursor INTO @ETLBatchID, @ExtractBatchID;

END

--close and deallocate the cursor once all batches for a given table have been processed

CLOSE BatchCursor;

DEALLOCATE BatchCursor;

The script above has been vetted by various groups within BISL. Each group weighed in as far as an approach and processing on an ExtractBatchID by ExtractBatchID basis is the recommended methodology. Coding standards were also put forth such as the use of a cursor, use of the T-SQL MERGE as well as syntax and formatting best practices. [To recap this code is available from the CDW SharePoint link.](https://vaww.cdw.va.gov/support/Training/Shared%20Documents/Forms/AllItems.aspx?RootFolder=/Support/Training/Shared%20Documents/All%20Day%20Training%20Presentations/2015-07-29_Region03_OptimizingCDW_RDW_Processes/Incremental%20Load%20Samples&FolderCTID=0x01200004315B0DA48E3047A544EE49A5DBCA2E&View=%7bB66924E8-353B-4D57-BA74-539F63FCF067%7d)

### OpCode

Sometimes after records have been loaded into the data warehouse they are deleted from the source system. When that happens those records are not deleted from the data warehouse but marked with a value of “D” in the field OpCode (visible only in the SPV views) so that customers can mark those records likewise in their downstream data marts. By doing so the processing is kept simpler in that only UPDATES and INSERT scenarios are required in the MERGE statement, no DELETES. The full list of OpCode values is below:

* **D** - Indicates the record, identified by the business key fields, was deleted from Vista.  Can be used by systems extracting from CDW to remove those records.
* **I** - Used internally by the load process.  Customers should not use this value in their processes.
* **P** - Indicates that this record was generated by CDW as just a placeholder (no values other than the business key fields).  This occurs when there was a reference to this record in the source system but this record has not been loaded into CDW.  Usually it is a matter of timing and the values for this record will be populated during the next load cycle.  At that time “P” would be removed from OpCode. Generally customers should take no special action on records with this value and just load them as any other records.
* **S** - Used internally by the load process.  Customers should not use this value in their processes.
* **U** - Used internally by the load process.  Customers should not use this value in their processes.

## 

## Remote Project Database Example

In the second example, we will assume that a project team needs to pull data from an RDW instance but their project database sits on a separate server. This is a remote project scenario meaning the source data resides on one server and the target data is on another. The extract team needs to extract data from the TIU.TIUDocument view inside of the SPV database on that RDW server. This example reflects the method to do an incremental extract using SQL Server Integration Services (SSIS) and the ExtractBatchID value.

(Please note that in cases like this, the project team should request ETL services and an ETL application account using the CDW Project Request Form – [link to Project Request Form](https://vaww.cdw.va.gov/Support/SitePages/ProjectRequestForm.aspx).)

### Setup

The first step for the team is to determine what columns are needed from TIUDocument. After doing so, they will need to create the following tables: a staging table named Dflt.StageTIUDocument and a production table named Dflt.ProdTIUDocument. In addition, they will create a table to keep track of the last ExtractBatchID processed during an incremental extract. Similar to the Local Project Database Example the project team needs to keep up with the ExtractBatchID values processed so that a new iteration of the data load knows where to start. Furthermore, the project team will capture additional measures such as counts by type – stage count and final count. The stage count is the number of rows queried from the source for a particular ETLBatchID and the final count is the number of rows in the destination table containing that ETLBatchID. The level of detail for capturing these counts will be the ExtractBatchID. That being the case, an additional view is created to return the maximum ExtractBatchID per table. The T-SQL for creating those objects is below:

CREATE TABLE [Dflt].[ExtractBatchSSISLog](

[DWViewName] [varchar](100) NULL,

[ExtractBatchID] [bigint] NULL,

[ETLBatchID] [int] NULL,

[ExtractDateTime] [datetime] NULL DEFAULT (getdate()),

[CountStage] [int] NULL,

[CountFinal] [int] NULL

) ON [DefFG]

GO

CREATE VIEW [Dflt].[LastExtractBatch]

AS

SELECT DWViewName, MAX(ExtractBatchID) AS LastExtractBatchID

FROM Dflt.ExtractBatchSSISLog

GROUP BY DWViewName

GO

Once these objects are in place, the team will create a new SSIS package to load the TIUDocument production table. This example will use the one package per table approach. In this approach, a single SSIS package will focus on loading one production table. This is the recommended methodology for developing and organizing SSIS packages.

The team designs each package so that it can be used to do an initial load as well as a nightly incremental load. The package design flow is as follows:

* Fetch the batches to be processed using a stored procedure call by passing in:
  + @DWViewName
  + @ETLType(Initial or Incremental)
* Uses a For Each Loop container to loop over the set of ETLBatchID values, executing the following tasks for each:
  + Using a Data Flow task, extracts the records needed from the source view, looking for those records that belong to the current ETLBatchID; these records are loaded into the appropriate staging table
  + Using a simple Execute SQL task, merges the data in the staging table into the appropriate production table
  + Finally, update the logging table with the batch IDs and counts

### Script

The T-SQL for the stored procedure Dflt.FetchBatchesPerTable mentioned above is seen below:

CREATE PROCEDURE [Dflt].[FetchBatchesPerTable]

@DWViewName VARCHAR(100),

@ETLType VARCHAR(20)

AS

BEGIN

SET NOCOUNT ON;

DECLARE @LastExtractBatchID BIGINT;

IF @ETLType = 'Initial'

BEGIN

SET @LastExtractBatchID = 0;

END;

IF @ETLType = 'Incremental'

BEGIN

SELECT @LastExtractBatchID = LastExtractBatchID FROM Dflt.LastExtractBatch;

END;

--SELECT @LastExtractBatchID

--get batch ids

SELECT EX.ETLBatchID, EX.ExtractBatchID

FROM SPV.EB.ExtractBatch AS EX

WHERE EX.DWViewName = @DWViewName

AND EX.ExtractBatchID > @LastExtractBatchID

ORDER BY EX.ExtractBatchID;

END

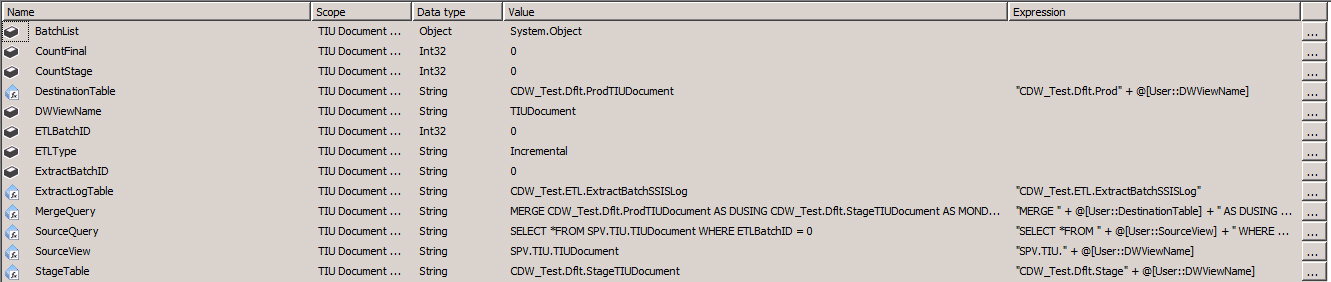
(Please note that in the case of an initial load the LastExtractBatchID is set to 0 and thus the procedure would return all ExtractBatchID values that are present in the ExtractBatch table for the necessary view.)

### SSIS Package

The following images and paragraphs describe the design of the package for loading the Dflt.ProdTIUDocument table.

##### Package Variables

The SSIS package contains several variables, some with expressions and some without, that perform various roles in the packages (source queries, tables name, etc.)

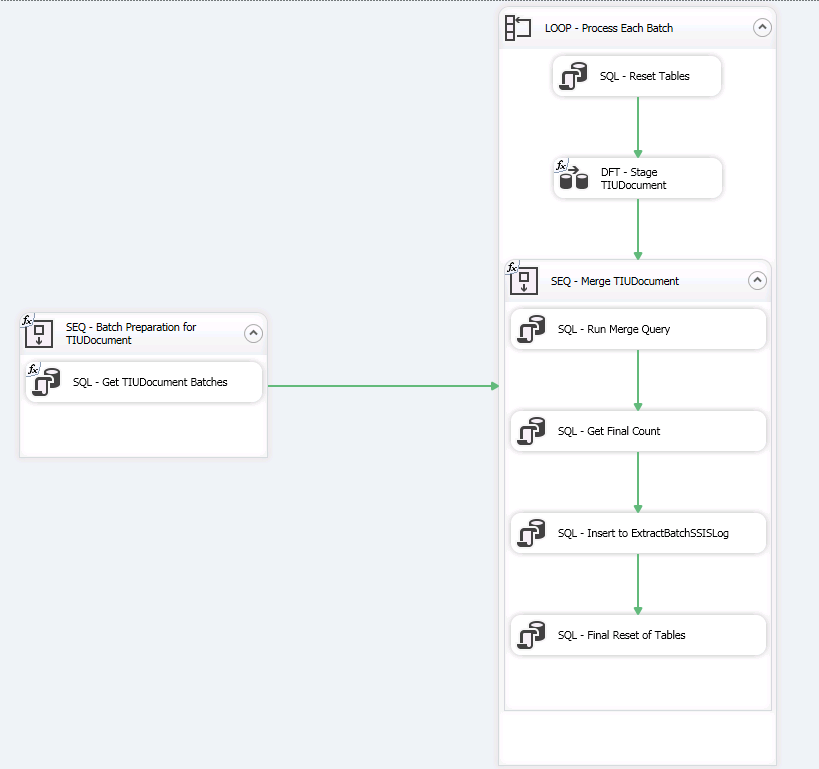


Below is a list of the variables that will require the project team’s attention when developing a new SSIS package:

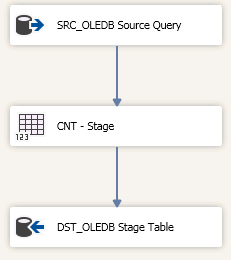
* DestinationTable (@[User::DestinationTable])
  + Value: CDW\_Test.Dflt.ProdTIUDocument
  + Meaning: This is the three-part table name of the production table – i.e., DatabaseName.SchemaName.TableName. The value has an expression and the team should modify the database and schema names to fit their needs.
* DWViewName (@[User::DWViewName])
  + Value: TIUDocument
  + Meaning: This is the root name of the view minus the schema.
* ETLType (@[User::ETLType])
  + Value: Initial or Incremental
  + Meaning: The variable value is passed as a parameter to DFLT.FetchBatchesPerTable in order to set the last used ExtractBatchID. Remember in the case of an initial load (ETLType = “Initial”) the LastExtractBatchID defaults to 0.
* MergeQuery (@[User::MergeQuery])
  + Value: T-SQL MERGE written as expression
  + Meaning: The T-SQL generated performs the merge of the Dflt.StageTIUDocument and Dflt.ProdTIUDocument tables. The project team should modify this expression.
* SourceView (@[User:: SourceView])
  + Value: SPV.TIU.TIUDocument
  + Meaning: This is the three-part name of the SPV source view – i.e., DatabaseName.SchemaName.ViewName. The value has an expression and the team should modify the database and schema names to fit their needs.
* StageTable (@[User::StageTable])
  + Value: CDW\_Test.Dflt.StageTIUDocument
  + Meaning: This is the three-part name of the SPV source view – i.e. ,DatabaseName.SchemaName.ViewName. The value has an expression and the team should modify the database and schema names to fit their needs.

##### Control Flow

The Control Flow of the package contains a handful of containers and all the objects therein.



* **SEQ - Batch Preparation for TIUDocument**: This sequence container holds tasks related to getting the batches needed for processing. *Note that the name of the object is based on an expression.*
* **SQL - Get TIUDocument Batches**: This Execute SQL task calls the stored procedure DFLT.FetchBatchesPerTable. It passes in the parameter values of @DWViewName and @ETLType. Note that those parameters are mapped from the variables @[User::DWViewName] and @[User::ETLType]. In addition, the list returned is stored as a recordset object in the variable @[User:: BatchList]. *Finally, note that the name of the object is based on an expression.*
* **LOOP - Process Each Batch**: This Foreach Loop container is used to loop over the values that are returned by the prior task into the @[User:: BatchList] variable. The container uses a Foreach ADO Enumerator to handle the looping and stores the current ETLBatchID and ExtractBatchID value while looping in the @[User:: ETLBatchID] and @[User:: ExtractBatchID] variables.
* **SQL - Reset Tables**: This Execute SQL task truncates the TIUDocument staging table. The staging table is truncated at the beginning of each batch so that the previous batch is flushed out and not reconsidered.
* **DFT - Stage TIUDocument:** This Data Flow task extracts the source data and loads it into the staging table. Also, a count of the records read from the source view is captured and loaded into a variable. The Data Flow task is shown below:



The source data is pulled from the RDW instance. The data is loaded into the TIUDocument staging table and in between an intermediary step counts the rows. The resulting count is stored in the following variable: @[User:: CountStage. *Note that the name of the object is based on an expression.*

* **SEQ - Merge TIUDocument**: This sequence container holds tasks related to completing the merge of the TIUDocument production table as well as the log table. *Note that the name of the object is based on an expression.*
* **SQL - Run Merge Query:** This Execute SQL task is used to run a simple MERGE command that merges the data from the staging table into the production table. The Execute SQL task makes use of the @[User::MergeQuery] which contains the actual MERGE logic. The team would modify the variable value rather than modifying the logic contained in the task’s SQL statement.
* **SQL - Get Final Count:** This Execute SQL task runs a query against the production table and counts the number of rows that contain the current ETLBatchID. The variable @[User::CountFinal] captures the count returned by the query.
* **SQL - Insert to ExtractBatchSSISLog**: This Execute SQL task is used to write back the processed batch information to the Dflt.ExtractLog table. This includes the following: @[User::ExtractBatchID], @[User::ETLBatchID], @[User::DWViewName], @[User::CountStage] and @[User::CountFinal]. These variables are passed along to a stored procedure. The stored procedure accepts the input and inserts it into the Dflt.ExtractLog table. Now all of the needed information for the current ExtractBatchID value is captured. The stored procedure, Dflt.UpdateExtractBatchSSISLog, is seen below:

CREATE PROCEDURE [Dflt].[UpdateExtractBatchSSISLog]

@DWViewName VARCHAR(100),

@ExtractBatchID INT,

@ETLBatchID INT,

@CountStage INT,

@CountFinal INT

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO Dflt.ExtractBatchSSISLog

(

ExtractDateTime,

DWViewName,

ExtractBatchID,

ETLBatchID,

CountStage,

CountFinal

)

VALUES

(

GETDATE(),

@DWViewName,

@ExtractBatchID,

@ETLBatchID,

@CountStage,

@CountFinal

);

END

* **SQL - Final Reset of Tables:** This final Execute SQL task truncates the TIUDocument staging table. The staging table is truncated at the end of each batch so that the unnecessary data is not backed up.

Once this package is defined then the project team can validate that the results will meet their needs. After verifying the results they are ready to use a SQL Server Agent job to load their table(s) on a nightly basis. The job will include a SQL Server Integration Services Package step and the source of that package will be the SSIS Catalog as they should be working in Project Deployment mode. (Note: More information about Project Deployment can be found on SP – [link](https://vaww.cdw.va.gov/support/_layouts/WordViewer.aspx?id=/TechTeam/ETL/ETL%20Documents/Guide%20to%20Using%20SSIS%20Project%20Deployment%20Model/Guide%20to%20Using%20SSIS%20Project%20Deployment%20Model.docx&Source=https://vaww.dwh.cdw.portal.va.gov/TechTeam/ETL/ETL%20Documents/Forms/AllItems.aspx?RootFolder=/TechTeam/ETL/ETL%20Documents/Guide%20to%20Using%20SSIS%20Project%20Deployment%20Model&FolderCTID=0x0120004A82BDD8D1F3024C98EF2C1859D1D874&View=%7bE7F24DA9-0EE8-43F7-96DF-80E0B22A172F%7d&DefaultItemOpen=1&DefaultItemOpen=1).) The job should run under the context of an SSIS Proxy using a valid ETL-specific utility account.

## Complex Load Scenario

The previous T-SQL and SSIS load examples were for extracting and loading a single table in an incremental fashion. Many development teams have project requirements that make it necessary to pull from multiple tables and load that comingled data into a single reporting table. This makes extracting incrementally more challenging but certainly possible. It does, however, require the team to identify a driving, or primary, view. This view is likely found in the initial join query.

Here is an example load query:

select a.sta3n

, a.patientsid

, c.patientssn, c.patientname

, a.visitdatetime

, a.visitsid

, a.primarystopcode

, a.secondarystopcode

, a.locationsid

, b.locationname

, a.PCEDataSourceSID

into #visits

from VDWWork.Outpat.Visit a

join VDWWork.Dim.Location b on a.locationsid = b.locationsid

join VDWWork.SPatient.SPatient c on a.patientsid = c.patientsid

join VDWWork.Dim.StopCode d on a.primarystopcodesid = d.stopcodesid

left outer join VDWWork.Dim.StopCode e on a.secondarystopcodesid = e.stopcodesid

join #FY f on dateadd(dd, datediff(dd, 0, a.visitdatetime), 0) >= f.CurrentFYStartDate

where a.sta3n = 538 and --a.ETLBatchID > g.MaxETLBatchID and

dateadd(dd, datediff(dd, 0, a.visitdatetime), 0) >= f.CurrentFYStartDate and

a.checkoutprocesscompletiondatetime is not null and b.noncountclinicflag = 'n'

and c.CDWPossibleTestPatientFlag = 'n';

-- get all providers for encounter

if object\_id('tempdb..#visitsProv') is not null

drop table #visitsProv

select a.visitsid

, a.visitdatetime

, c.staffien as providerien

, b.providersid

, b.primarysecondary

, c.staffname as EncounterProvider

, c.servicesection as EncounterProviderServiceSection

into #visitsProv

from #visits a

join VDWWork.Outpat.VProvider b on a.visitsid = b.VisitSID

join VDWWork.SStaff.SStaff c on b.providersid = c.staffsid;

In this example, the team is trying to gather Visit data for a particular Sta3n, which occur after a particular date, and so on. The query gathers columns from Outpat.Visit and related views. The script then goes on to combine the filtered Visit data and Outpat.VProvider. There are subsequent joins not seen above that touch other Outpat views like Outpat.VProcedure.

One solution could be to load each view incrementally into a staging table and then join the staging tables in order to create the final reporting table. Landing each view into a separate staging table is not advised because of the sheer volume of data returned from the views. For storage and performance reasons this method is not recommended. This method would also require some surgical merging of the final reporting table in order to have an end-to-end incremental process.

The recommended solution is to select a driving table and allow that to steer the process. An example of this approach in T-SQL will be shown in this document. The flow of the T- script is as follows:

* Declare a cursor named BatchCursor to house the ExtractBatchID values and related ETLBatchID values of interest for Visit
  + Populated by querying EB.ExtractBatch view in the SPV database
    - Filter for DWViewName = ‘Visit’
    - Filter for ExtractBatchID > the LastExtractBatchID processed for Visit
  + Loop over each ExtractBatchID and grab Visit records for each associated ETLBatchID
* For each ExtractBatchID value returned:
  + Query SPV.Outpat.Visit records for a given ETLBatchID into a temporary table
  + Delete the SID records from Dflt.ProdVisitDetail based on a join to the temporary table created in the previous step
  + Insert the records from the temporary table into the reporting table Dflt.ProdVisitDetail
  + Finally, update the Dflt.ExtractBatchLog table recording the ExtractBatchID as the last handled by the extract process

### Script

The cursor example is seen below. (Please note the filtering seen in the script can be tailored to fit the specific business needs of the project team.)

--declare batch variables

DECLARE @ETLBatchID AS INT, @ExtractBatchID AS INT;

DECLARE @DWViewName AS VARCHAR(100) = 'Visit';

--declare any other variables

DECLARE @FYStartDate DATETIME;

--set FYStartDate variable; in this case will be Oct. 1 of previous year

SET @FYStartDate = (SELECT CONVERT(DATETIME, CONCAT(YEAR(DATEADD(YEAR, -1, GETDATE())), '1001')));

DECLARE BatchCursor CURSOR STATIC

FOR

--get batch ids

SELECT EX.ETLBatchID, EX.ExtractBatchID

FROM SPV.EB.ExtractBatch AS EX

INNER JOIN Dflt.ExtractBatchLog AS EL ON

EX.DWViewName = EL.DWViewName

WHERE EX.DWViewName = @DWViewName

AND EX.ExtractBatchID > EL.LastExtractBatchID

ORDER BY EX.ExtractBatchID;

OPEN BatchCursor;

FETCH NEXT FROM BatchCursor INTO @ETLBatchID, @ExtractBatchID;

WHILE @@FETCH\_STATUS = 0

BEGIN

--gather visit info

--also get all providers for encounter(visit)

IF OBJECT\_ID('tempdb..#Visit') IS NOT NULL

DROP TABLE #Visit

SELECT VIS.Sta3n

,VIS.VisitSID

,VIS.PatientSID

,VIS.VisitDateTime

,VIS.PrimaryStopCode

,VIS.SecondaryStopCode

,VIS.LocationSID

,VIS.PCEDataSourceSID

,VIS.ETLBatchID

,CONVERT(DATE, VIS.VisitDateTime) AS Visit

INTO #Visit

FROM SPV.Outpat.Visit VIS

INNER JOIN SPV.Dim.VistaSite DVS ON --join to VistaSite can allow you to select whole VISNs

VIS.Sta3n = DVS.Sta3n

WHERE VIS.ETLBatchID = @ETLBatchID --filter for current ETLBatchID

--add any other filters to pare down number of rows

AND VIS.VisitDateTime >= @FYStartDate

AND VIS.CheckOutProcessCompletionDateTime IS NOT NULL

AND VIS.OpCode <> 'D'

AND VIS.Sta3n = 538;

--AND DVS.VISN = 10;

--\*\*this step will remove the SID value gathered in the previous step; the delete is based on a join to the temporary table

--\*\*PLEASE NOTE: the updated SID values will be updated in a subsequent step

--remove SIDs from target

DELETE D

FROM Dflt.ProdVisitDetail D

INNER JOIN #Visit S ON

D.VisitSID = S.VisitSID

IF OBJECT\_ID('tempdb..#VisitProvider') IS NOT NULL

DROP TABLE #VisitProvider

SELECT VIS.Sta3n

,VIS.PatientSID

,SPAT.PatientSSN

,SPAT.PatientName

,VIS.VisitDateTime

,VIS.VisitSID

,VIS.PrimaryStopCode

,VIS.SecondaryStopCode

,VIS.LocationSID

,LOC.LocationName

,VIS.PCEDataSourceSID

,SSTA.StaffIEN AS ProviderIEN

,VPROV.ProviderSID

,VPROV.PrimarySecondary

,SSTA.StaffName AS EncounterProvider

,SSTA.ServiceSection AS EncounterProviderServiceSection

,DPT.Classification

,VIS.ETLBatchID

INTO #VisitProvider

FROM #Visit VIS

INNER JOIN SPV.Outpat.VProvider VPROV ON

VIS.VisitSID = VPROV.VisitSID

AND VPROV.OpCode <> 'D'

INNER JOIN SPV.SStaff.SStaff SSTA ON

VPROV.ProviderSID = SSTA.StaffSID

AND SSTA.OpCode <> 'D'

INNER JOIN SPV.StaffSub.ProviderTypeAssignment PTA ON

VPROV.ProviderSID = PTA.StaffSID

AND PTA.OpCode <> 'D'

INNER JOIN SPV.Dim.Location LOC ON

VIS.LocationSID = LOC.LocationSID

AND LOC.OpCode <> 'D'

INNER JOIN SPV.SPatient.SPatient SPAT ON

VIS.PatientSID = SPAT.PatientSID

AND SPAT.OpCode <> 'D'

INNER JOIN SPV.Dim.ProviderType DPT ON

PTA.ProviderTypeSID = DPT.ProviderTypeSID

AND DPT.OpCode <> 'D'

--REMOVED JOINS TO Dim.StopCode; CAN BE ADDED BACK IF NEEDED

WHERE LOC.NonCountClinicFlag = 'N'

AND SPAT.CDWPossibleTestPatientFlag = 'N'

AND (CONVERT(DATE, VIS.VisitDateTime) >= PTA.EffectiveDate AND CONVERT(DATE, VIS.VisitDateTime) <= COALESCE(PTA.ExpirationDate, '12/31/9999'));

--get CPT codes

IF OBJECT\_ID('tempdb..#VisitCPT') IS NOT NULL

DROP TABLE #VisitCPT

SELECT DISTINCT VIS.visitsid

,CPT.CPTIEN

,CPT.CPTCode

,VPROC.Quantity

,CPT.CPTName

,VPROC.CPTSID

INTO #VisitCPT

FROM #Visit VIS

INNER JOIN SPV.Outpat.VProcedure VPROC ON

VIS.VisitSID = VPROC.VisitSID

AND VPROC.OpCode <> 'D'

INNER JOIN SPV.Dim.CPT CPT ON

VPROC.CPTSID = CPT.CPTSID

AND CPT.OpCode <> 'D';

--re-insert those same SIDs based on updates made

INSERT INTO Dflt.ProdVisitDetail WITH (TABLOCK)

(

Sta3n, PatientSID, PatientSSN, PatientName, VisitDateTime, VisitSID, PrimaryStopCode, SecondaryStopCode, LocationSID, LocationName,

PCEDataSourceSID, ProviderSID, PrimarySecondary, EncounterProvider, EncounterProviderServiceSection, Classification, CPTCode,

Quantity, CPTName, CPTSID, ETLBatchID

)

SELECT DISTINCT

A.Sta3n

,A.PatientSID

,A.PatientSSN

,A.PatientName

,A.VisitDateTime

,A.VisitSID

,A.PrimaryStopCode

,A.SecondaryStopCode

,A.LocationSID

,A.LocationName

,A.PCEDataSourceSID

,A.ProviderSID

,A.PrimarySecondary

,A.EncounterProvider

,A.EncounterProviderServiceSection

,A.Classification

,B.CPTCode

,B.Quantity

,B.CPTName

,B.CPTSID

,A.ETLBatchID

FROM #VisitProvider A

INNER JOIN #VisitCPT B ON

A.VisitSID = B.VisitSID

--after merging batch then update the ExtractBatchLog table

UPDATE Dflt.ExtractBatchLog

SET LastExtractBatchID = @ExtractBatchID, LastExtractDateTime = GETDATE()

WHERE DWViewName = @DWViewName;

--grab the next batch for processing

FETCH NEXT FROM BatchCursor INTO @ETLBatchID, @ExtractBatchID;

END

--close and deallocate the cursor once all batches for a given table have been processed

CLOSE BatchCursor;

DEALLOCATE BatchCursor;

Much like the local project example, the script above has been vetted by various groups within BISL. As previously mentioned, each group weighed in as far as an approach and processing on an ExtractBatchID by ExtractBatchID basis is the recommended methodology. The complex example is different in that it performs a DELETE and INSERT rather than a MERGE. In this case, gathering the new or updated SIDs from the driving view and removing those from the final table is acceptable and actually preferred in this scenario. [Much like the other examples, this code will be available from the CDW SharePoint link.](https://vaww.cdw.va.gov/support/Training/Shared%20Documents/Forms/AllItems.aspx?RootFolder=/Support/Training/Shared%20Documents/All%20Day%20Training%20Presentations/2015-07-29_Region03_OptimizingCDW_RDW_Processes/Incremental%20Load%20Samples&FolderCTID=0x01200004315B0DA48E3047A544EE49A5DBCA2E&View=%7bB66924E8-353B-4D57-BA74-539F63FCF067%7d)

## Exclusions

The tables for the following views are not loaded incrementally and therefore do not contain batch IDs. Therefore copying the whole view is the only practical option. These all represent small tables.

* Dim.Date
* Dim.Time
* Dim.VistaSite
* All of the DP views
* All of the Meta views

# Recommendations

It will be up to the individual project teams to improve upon the current SSIS example. The example does make use of variables, project parameters and some count logging. It does not, however, include event handlers, package logging or other useful package design features. This should serve as a base package for the teams to build upon and make into a viable template for future development.

The code in this document is also not intended to be a “on size fits all” solution. It is rather a starting point for incremental development. Each group has different needs and those needs will need to be reflected as logic inside of their T-SQL scripts and SSIS packages.