ETL Patterns Surrounding DEI

Contents

[Patterns within DEI with connection environments 1](#_Toc128137157)

[Load Patterns in DEI for writing to Hadoop 2](#_Toc128137158)

[Dynamic Patterns 2](#_Toc128137159)

[Notes 9](#_Toc128137160)

[How PowerCenter interacts with DEI to bring it all together 11](#_Toc128137161)

# Patterns within DEI with connection environments

* 1. FlatFile to Hadoop, HDP to FF, HDP to HDP, HDP to SQL/DB2, SQL/DB2 to HDP
     1. Configuration for larger data sets/complex mappings
        1. Validation Environments: Hadoop – Spark
           1. \*With the current cluster, ‘Blaze’ should not be set here
        2. Execution Environment: Blaze/Blaze\_CDP(tez)
     2. Configuration for simple mappings and small data sets (FF/HDP to HDP/FF)
        1. Validation Environments: Native
        2. Execution Environment: Native
           1. These settings are used when the mapping is read/writing < 100,000 records with insert/insert truncate or FF reading/writing. Spark takes a couple minutes to spin up. Mappings that meet this criterion can complete in less than 30 seconds because they do not have to wait for Spark to start up/shut down.
  2. Jobs that previously ran on the validation environment: Hadoop – Both Spark and Blaze now fail and have been changed to remove ‘Blaze’ from the validation environment only

# Load Patterns in DEI for writing to Hadoop

* 1. Insert/Insert Truncate
     1. Single Mapping with a swap script.
        1. Mapping: Writes source data to a stage table (\_load), then swaps the load with the main with a script (the correct method for insert and truncate)
        2. \*Insert is being done directly on the main table in some older mappings
           1. \*Insert and truncate should never be done directly on the main table.
  2. Upsert/Update
     1. At least 2 Mappings: \_01, \_2 … and so on, plus scripts for snapshots
        1. Mapping \_01: Staging of the data in a load or stage table (aedwdb.’table\_load’, etlworkdb.’table\_stg’
        2. Mapping \_02+: Reconcile the data between the stage and the main table
        3. A script ***should*** be in the flow to remove the old snapshot of the data and add the snapshot back when the data has changed.
           1. \*Some of the old flows may not have snapshots for historical tables and should be reworked at some point to include them. If the tables get dropped/truncated there are no backups of this data.

# Dynamic Patterns

**ALL DYNAMIC JOBS CALLED FROM POWERCENTER**

* 1. **Dynamic\_Command\_ACID**
     1. Purpose: To write to Hadoop with data already existing in Hadoop (no other sources outside of Hadoop)
        1. *Hadoop to Hadoop only*
     2. Using 4 Scripts, No mapping (no execution environment can be sent since there is no mapping. It is using the beeline connection to impala)
        1. Using a script to stage the data from existing Hadoop tables into a separate table
        2. Remove current snapshot on the target (main) table
        3. Reconcile the data in the stage table to the main table through an HQL script executed in impala
        4. Create a snapshot of the target (main) table
  2. **Dynamic\_File\_To\_Hive\_ACID** 
     1. Purpose: Perform insert/update/upsert with data pulled with PowerCenter from any source/server/connection
     2. Uses 1 mapping and 3 scripts
        1. Mapping: Dynamic\_file\_to\_hive\_load
           1. Reads in a flat file on the DEI share drive and writes it to a stage table (‘table’\_load)
           2. Validation Environments: Hadoop – Spark
           3. Execution Environment: Blaze/Blaze\_CDP(tez)

Execution environment is set by the user in the param file

* + - 1. Using scripts (4 scripts)
         1. Remove current snapshot on the target (main) table
         2. Reconcile the data in the stage table to the main table through an HQL shell script executed in impala
         3. Swap the \_load with the main table
         4. Create a snapshot of the target (main) table
    1. Table/View setup required for this flow:
       1. A main table in aedwdb – aedwdb.’table’
       2. A load table in aedwdb – aedwdb.’table\_load’
       3. A stage table in etlworkdb – etlworkdb.’table\_stg’
          1. \*When the job loads the incoming data from the flat file into this table, all the fields are loaded as strings. \*\*Every field in the stage table is set to String even the date fields. The casting of these fields is done in the view of the table NOT the mapping
       4. A view of the stage table in etlviewsdb – etlviewsdb.’table\_stg\_v’
          1. \*The view of the table will cast any string from the stage table to a date\_time field
  1. **Dynamic\_File\_To\_Hive\_Command\_With\_Dates**
     1. A similar flow as Dynamic\_File\_To\_Hive\_Command. It was created to reduce complexity. Created to have DEI convert string date fields to time\_stamps, eliminating the need for a view of the stage table with logic in the view to convert timestamps
        1. Similar to the previous job, *EXCEPT* inside the mapping there is an expression that will convert any field that has appended to it – ‘\_dt’, ‘\_date’, ‘\_datetime’, ‘\_tstamp’, ‘\_ttstamp’, from a string to a timestamp. Again, this is done because we are taking a flat file with the incoming data and passing it into DEI, so this data all comes in as strings.
     2. Tables required:
        1. A main table in aedwdb – aedwdb.’table’
        2. A load table in aedwdb – aedwdb.’table\_load’
        3. A stage table in etlworkdb – etlworkdb.’table\_stg’
           1. This job does not require a view of the stage table since the mapping casts the dates in the file as time\_stamps. This stage table is the same as the load and main table. The date fields are not set to String type like in the previous dynamic job.
  2. **Dynamic\_File\_To\_Hive\_ACID\_Pipe\_Delim\_With\_Dates**
     1. The exact same flow as Dynamic\_File\_To\_Hive\_Command\_With\_Dates
     2. EXCEPT it uses a Pipe delimiter and preserves double and single quotes
        1. Everything else is the same
     3. The use case is for jobs that have double quotes in the data that the existing Dynamic jobs are set to remove
  3. **Dynamic\_File\_To\_Hive\_KUDU\_Stg**
     1. The purpose is to have a separate dynamic job that writes to tables in Hadoop that are KUDU
        1. Kudu tables are stored differently (they are ‘stored as kudu’, opposed to ‘stored as parquet’ and ‘stored as textfile’ etc). They are in their own database – aedwdbk
        2. Unlike regular Hadoop tables, Kudu tables declare a primary key
           1. What this does for us is it allows us to scan the table quickly, But more importantly it makes the data relational. Which allows us to run commands like upsert/update/delete directly onto the table. These things cannot be done on regular Hive tables since they aren’t relational (they don’t declare a primary key structure)
     2. 1 Mapping and 2 Scripts
        1. Mapping: Dynamic\_File\_To\_Hive\_Load
           1. Takes the incoming deltas from a flat file and writes them to a stage table
        2. Scripts
           1. 1st Script calls a script on the share drive that will perform a

‘upsert into’, ‘update into’, ‘delete\_from’ directly on the table with the data from the staging table previously loaded.

* + - * 1. 2nd Script does a refresh and compute stats on the table
    1. Tables required:
       1. A main table in aedwdbk – aedwdbk.’table’
       2. A stage table in etlworkdb – etlworkdb.’table\_stg’
       3. A view of the stage table in etlviewsdb – etlviewsdb.’table\_stg\_v\_k’
  1. **Dynamic\_File\_To\_Hive\_KUDU\_Stg\_With\_Dates**
     1. Everything is the same as the previous job
     2. EXCEPT the mapping converts date\_fields that come in as strings to time\_stamps
        1. Meaning the stg table can have the date fields set to time\_stamps
        2. Use case is to reduce complexity of the setup of stage table and views of the stage table
     3. Tables Required
        1. A main table in aedwdbk – aedwdbk.’table’
        2. A stage table in etlworkdb – etlworkdb.’table\_stg’
           1. There is no need for a stg view since the mapping casts the string dates to time\_stamps
  2. **Dynamic\_File\_To\_Hive\_Load\_With\_Dates**
     1. The exact same flow as - Dynamic\_File\_To\_Hive\_ACID\_Pipe\_Delim\_With\_Dates
        1. I am unsure if any job is currently using this duplicate job.
  3. **Dynamic\_File\_To\_Hive\_TRUNC\_LOAD**
     1. 1 mapping and 2 Scripts
        1. Mapping: Dynamic\_File\_To\_Hive\_Load
           1. Writes to a staging table
        2. Scripts
           1. 1st script reconciles the data

Reconciles the data between the stage table and the main table

* + - * 1. 2nd Script refreshes the table
        2. \*\*This job does not create or remove any snapshots

Meaning there is no backup of the data and if another job comes into existence that writes to this table with an update/upsert that job would create a snapshot and the truncate job would fail trying to truncate a table that has a snapshot.

* + 1. Tables required:
       1. Main table in aedwdb
       2. Stage table in etlworkdb
  1. **Dynamic\_File\_To\_Hive\_TRUNC\_LOAD\_With\_Dates**
     1. The purpose of this job is to perform insert truncates
     2. 1 mapping and 1 Script
        1. Mapping: Dynamic\_File\_To\_Hive\_Load
           1. Writes directly the main table in aedwdb
        2. Scripts
           1. Refreshes the table
     3. This job was used heavily but should not be used going forward since we should not be writing directly to any table.

* + - 1. Instead, Dynamic\_File\_To\_Hive\_TRUNC\_LOAD\_TO\_STAGE, should be used going forward
    1. \*\*This job also does not create or remove any snapshots
  1. **Dynamic\_File\_To\_Hive\_TRUNC\_LOAD\_TO\_STAGE**
     1. Dynamic job to be called from PowerCenter to perform an insert truncate on a Hadoop table
     2. 1 Mapping and 4 Scripts
        1. Mapping: DYNAMIC\_FILE\_TO\_HIVE\_LOAD\_WITH\_DATES\_TO\_STAGE
     3. 4 Scripts
        1. Remove current snapshot on the target (main) table
        2. Reconcile the data in the stage table to the main table through an HQL shell script executed in impala
        3. Swap the \_load with the main table
        4. Create a snapshot of the target (main) table
  2. **Dynamic\_File\_To\_Hive\_Partition\_Cmd\_With\_Dates**
     1. Use case: A dynamic job to write to partition tables in Hadoop.
     2. 1 Mapping and 3 Scripts
        1. Mapping: stages the incoming data into an \_incr table
        2. 1st Script gets the partitions of the table that will be updated and loads into a \_stg table. Then reconciles the data between the \_incr table (incoming new data) and \_stg table (existing partition data from the table that will be added/updated)
        3. 2nd Script drops the partitions that will be updated in the main table. Inserts the data from the \_stg table into the main table
        4. 3rd Script refreshes the table

# Notes

* + 1. In all the dynamic jobs the execution environment variable is set by the user for each individual job in the parameter file for ‘Hadoop\_connection’ (or something named similar)
       1. Which means the user sets whether to use Blaze or Blaze\_CDP (tez) for the execution environment.
       2. **Our current setup for jobs is to use Tez (Blaze\_CDP) as our backend execution environment. Spark is used for our validation environment**
          1. Tez is the newer option that was recommended for us to use by Cloudera services. Tez saves resources. In many cases it seems to have a performance increase with jobs completing faster.
          2. The Tez engine has better execution-time than MapReduce
       3. The jobs are all set for the validation environment as
          1. Hadoop – Spark

Spark being the engine architecture used on the Hadoop cluster to run our mappings.

It takes a couple minutes to spin up and shut down. Its job as an executor is to submit the mapping to the Hadoop cluster to run

* + - 1. Kudu tables are relational tables. They have a primary key declared and can perform commands through a SQL script in a shell script or inside of PowerCenter to update/upsert/delete/insert directly onto tables.
         1. These tables are much more computationally expensive
         2. They allow for advanced partitioning

Combining Hash and Range partitioning

* + - * 1. Limitation

Updates/inserts/deletes are non-transactional. If the SQL fails partway, it cannot be rolled back.

# How PowerCenter interacts with DEI to bring it all together

* + 1. (Most) of the dynamic jobs take in its source data from a flat file on the DEIShare drive (with the exception of the dynamic\_command job which takes data from existing sources in Hadoop, so there is no need for a source flat file with the data/deltas)
       1. These delta/source files get created inside of PowerCenter and then are placed on the DEI share drive in this directory
          1. //adwde01/DEIShare/PC\_TO\_DEI

Dev example

* + - 1. This solution was made as a workaround to continue to use PowerCenter due to the difficulty/problems of using DEI. Instead of creating DEI mappings and workflows the solution is to develop the mappings inside of PowerCenter and write the desired output to a flat file which gets put on the DEI share drive for the dynamic jobs to use to write to Hadoop.
         1. This gave us the flexibility to continue to use PowerCenter for our development and keep most of our jobs housed there while having a pattern in place to follow that eliminates complex DEI work.
         2. To be able to use PowerCenter to call the dynamic jobs the ‘plink’ scripts were created (by Tommy Dato). These are bat scripts on the informatica share drive that can SSH into the DEI server and kick of shell scripts which can run jobs.
         3. The dynamic job is passed the parameter file containing what source file and what target table is getting read/loaded.
         4. The DEI job uses this information to dynamically get the fields for the table it Is loading and the fields from the flat file.
         5. These scripts return logs and an exit code to let PowerCenter know if the script succeeded or failed and log the error/success.
  1. **Downsides/Things to consider with the Dynamic Jobs**
     1. Very hard to monitor
        1. In DEI you must search through the dynamic jobs and their logs to find the one you are looking for. Many instances of the same dynamic job are running around the same time. In PowerCenter all you can see if your command task running, no other details.
     2. Dependent on logging and the process of passing and accessing information back and forth from the INFA server and the DEI server.
     3. Staying on PowerCenter and having two separate places for our ETL jobs
        1. How long do we plan to stay on PowerCenter and what are the limitations we are facing not utilizing Hadoop to its full potential in doing so?
     4. We are storing lots of data for our source files on the DEI share drive.
        1. Every table we want to write to for Hadoop has to get written to a flat file first and placed on the DEI share drive.
           1. \*We do have some purge criteria for cleaning up files.