Gary Griswold
August 16, 2019

DBP v4 Load from LPTS

with modifications from Aug 16 meeting

Objectives

The objectives of this project include improving the data correctness of the LPTS data loaded into the DBP database; improving the performance of the load; improving maintainability of the program; reducing the effort required to use the program; implement the latest specification of permissions; and to provide an easy to understand program that can later become the basis of a new load when LPTS is replaced.

Revise vs. Rewrite

If it were possible to correct the problems by revising the current program, that would be ideal. However, the current program updates one row in the database at time, as it processes each fileset, and this must be changed to updating batches of transactions both for performance and reliability reasons. Hence, rewriting is simpler than modifying the current program. It will be best to write the new program in Python, because there are many other DBP utility programs written in that language.

Load Process

The manual process of running the program should be improved in order to reduce the amount of effort required. It would be nice if the program could be run automatically, when files are placed in a directory, but this might not be practical if the program is frequently restarted after problems are corrected. The following processing is suggested:

- The load processed is organized by a set of directories that describe the status of the load. The directory names are defined in the config file, but their expected names are: 1. validate, 2. upload, 3. database, 4. done.

- Typically, a load is started by placing the files for one or more damId's in the validate directory.
- When the load program is started, it scans the files in the 'validate' directory to get the damId/fileset names.
- It reads the access database to see if the damIds are already present. If one does not exist, the program prompts for the stockNumber for each damId and stores those damIds in the LPTS.
- It reads the access database to see if there is a bible_id for the LPTS record. If not, it prompts for the bible_id, and updates it into the LPTS record.
- When these initialization steps are completed for all damId's, a message is output to the screen to let Alan know that it has started the run for all damIds.
- It performs the LPTS extract for the selected stockNumber, and generates a file. These LPTS extract files are moved into an archive directory for storing all past extract files.
- It then runs all validation functions for all LPTS extract fields that require a validation.
- If there are any validation problems, the process stops for that damId. Errors are put into an error log, and those damId files are left in the validate directory.
- If there are no errors for a damId, all of those files are moved to the 'upload' directory.
- Alan, Taylor or others can correct the problems, and if there are problems that are going to be accepted, the program can be restarted with an error-override option.
- The program, now uploads each file to AWS and as each file upload succeeds, the file is moved to the 'database' directory to indicate upload success.
- For text files the text processing programs that generate text upload files and SQL statements will be run and then the load to AWS will be performed.
- When all of the files have been uploaded, and moved to the 'database' directory, the program updates the DBP database with all of the information in the LPTS extract performing data transformations and cleanup as needed.
- Errors are not expected during the update, because verification should have caught problems that will cause failure, but I will organize the updates so that the database is left in a coherent state on failure.
- After a failure, the program can be restarted, and because the files are in the 'database' directory, it will restart the DBP update.

- When the DBP update completes successfully, it will update the DBP status in the LPTS database to be 'live'. It also updates the load date in LPTS.
- If there are other LPTS data fields that should be updated, please let me know their identity.
- When the update of the DBP database completes successfully, the files that have been uploaded are moved from the 'database' directory to the 'done' directory, which should be the repository where they are permanently stored.
- While the load program is running it displays status information on the console for each bible_id/fileset_id to identify the stage it has reached.

There is a second case, where one wants to rerun the load program from LPTS to DBP without uploading data. There will be an option of the program that allows this, and does not require files to be moved in the directories. It will be possible to do this run for a specific damId, or for the entire database.

Validation

A validation program will be written that, as much as possible, compares the data in the DBP database, LPTS database, AWS S3 bucket. This is essential to testing the new DBP load program, but it will also be useful later for revalidating the system any time an unexpected failure occurs during processing.

Load Program Design

The load program will be written using a design common to ETL (Extract Transfer Load) programs. That is, it will read an external file that defines the data fields to be read from Access, the validations to be perform, and transformations to be performed, and the data columns to receive the result. However, this external file will not contain any procedural programming instructions, rather, it will contain the names of the functions to be performed for each validation, cleanup or transformation of a data item.

The load specification file, will be completed with the rest of the load program, but the file as currently defined is an attached document.

The purpose of this design is to produce a program that could be easily extended to contain additional data columns or even tables. And to provide a readable specification of the load that can be used by the future load of the new LPTS.

All validation functions will be together in one module so they can be called by name, and all cleanup function will also be in a different module, and transformation functions in a third module. These functions will not use any shared state, but only process parameters passed into them. This is done to insure that the all logic for each specific validation, cleanup, or transformation is all in one place.

Implementation Steps (3 to 4 months)

The load program will be developed in a number of steps:

- Write code to read in the external ETL specification, and build internal data objects that will be used for all subsequent processing.
- Write the code to extract fields from Access, but initially ignore validation.
- Write proof-of-concept code that will take un-transformed LPTS data, and build internal data objects that contain this data, and then use that data to generate SQL.
- Write the validation functions for LPTS data.
- Write the cleanup functions for LPTS data.
- Write the transform functions for LPTS data.
- Write the process that converts all of the transformed data in SQL statements, and organizes these into batches.
- Write the process that updates the DBP database.
- Add S3 Upload to the program.
- Write the above mentioned validation program, and fix any problems.
- Write the all of the process steps that automate the processing for Alan.
- Come to Albuquerque to deploy the new program.