Gary Griswold
November 9, 2019

DBPv4 Load

Software Design Description

After struggling to find ways to validate the data being loaded into ETL, the earlier design document written on Aug 16, 2019 is extended by the following:

The ideal program could be event driven using the movement of files in directories or buckets. Possibly, it would use queues to control multiple processes or subprocesses. However, the initial deployment should be a command-line synchronous process. So, it should be developed as a set of standalone executables whose functionality is consistent with the processes of a multiprocessing program. And the first deployment will simply have a controller class that ties these together into one synchronous process.

The program will consist of the following executable modules that can be run together or independently.

1) Input validation
2) Upload & Calculate
3) Database Update
4) Bucket Cleanup

Each process will have input and output directories, and the files will be moved through these directories as processing proceeds. The directories are named so they will appear in reverse sequence.

1) validation
2) upload
3) database

4) cleanup

User Feedback will be one line per module and fileset. The module names: validate, upload, database will be first on the line, followed by the fileset being processed, followed by the first letter of the books of the Bible. Errors summaries will be displayed also, but the entire error file will be in a files with the date and time in the filename so that they are not overwritten by successive runs.

Input Validation Module

The program accepts multiple types of inputs

1) input files in the above mentioned directories

2) fileset id's referring to objects in a bucket

3) file listings of objects in a bucket

The program can process one fileset, many filesets, one bucket, or many buckets. There is no capability to process sets of files less than an entire fileset because of the difficulty of identifying corresponding records to delete. However, the user will be able to update the bucket with specific files and then reprocess the fileset by inputting the filesetId.

The InputReader class would contain a set of iterators or generators for each kind of input so that the remaining processing uses the same logic for processing an entire bucket that it uses to process a set of files. There will be an iterator for all files, and one for just those that will be loaded into bible_files. That is, filenames with 3 slashes whose types are html, mp3, and mp4. The iterator would iterate at two levels, i.e. fileset and file so that there is a natural way to organize logic around the beginning and ending of a fileset.

1) The validation will be done first for all filesets to be processed, so that the errors can be communicated quickly back to the admin running the program.

2) It uses the FilenameParser class to find the best parse, and keeps a record of the parser that succeeded.

3) It performs a set of checks, including that chapters are complete and in range.

4) For each fileset, the validator looks up all of the required LPTS data. Required LPTS data that is absent is also an error.

5) The validator puts correctly parsed files that have no errors into a CSV file for each fileset, and stores those filesets in a single working directory.

6) Errors go to an error report, whose name includes date and time of the run.

7) Filesetids with errors go into a quarantine CSV file that later populates a quarantine table during the database update phase.

8) Bible Verses are handled differently, in an incremental load, the checking of text filesets will catch most errors in the Bible Verses, so they will not be extensively validated on their own. In a full load, they will be loaded from the current DBP. [I need to do further study of this data to be certain that is a good idea.]

9) Most text filesets have duplicates, and we want to accept the most recent. The most recent can be identified by the parser that successfully parsed it.

10) The files identified as duplicates should have their full name inserted into a duplicates CSV file.

11) The process should compare the coverage of the fileset duplicates and report a warning if the older has better coverage.

12) The process continues through the next phase for filesets that have no errors. When physical files are processed, they are moved to the upload folder.

Upload and Calculate

The upload iterates over all files in the filesets that passed validation if physical files are processed. If a listing of bucket files is being processed then there is nothing to do by this module.

1) Files will be uploaded and as the upload succeeds the file is moved to the database directory.

2) Files that fail in the upload will be automatically retried based upon their presence in the upload directory.

5) Video files are submitted to transcoding, meta data from the transcoding is collected and entered into a CSV file.

6) Audio files are processed with ffprobe to obtain a duration. This is also stored in a CSV file.

7) Once the last file upload has succeeded, and the last transcoder result is processed, there is some event that indicates that all files have been moved so that the database module can begin.

Database update

1) The typical use of the app will be to process the files of one or a few filesets. It will complete one fileset at a time. Although, the validation module is completed for all filesets being processed before any other modules are started.

2) The entire load will be one transaction, possibly excluding the bible-verses table, which will have its own transaction.

3) The load will be prepared as a text file of insert, replace and possibly update statements with all tables to be updated in a sequence that is correct for their foreign key references.

4) The batch will begin with deletes of the filesets presented in reverse order.

5) Each table insert that auto generates a key will be followed by a select statement that captures the key valve in an SQL variable. e.g. "select @file-id = lastrowid();" Then @file-id will be used as a value in other tables that have a column referencing the primary key.

6) This batch will be executed as a process, or subprocess, and as a single transaction per fileset that will rollback and produce errors if any unique key, or foreign key, or other SQL error occurs.

7) For the largest tables, during a full load, this module might also drop all non-unique secondary indexes before the load and recreate them at the end of the process.

8) The data to be loaded will mostly exist in CSV files created during validation and upload. These CSV files will iterate as an array of dictionaries, which should make the logic to populate the tables fairly simple.

Bucket Cleanup

The ideal database is one that contains all of the valid resources, none of the invalid resources, and contains complete and correct data about those resources. But the buckets should also be correct containing all of the resources described in the database with no invalid or duplicate resources. If that were true then it would always be possible to reconstruct the database from data in the buckets.

During the validation step, bucket data that was determined to be duplicate was recorded in a CSV file named duplicates. This post load process should move the objects in that table from the dbp-prod and dbp-vid buckets to a repository such as, dbp-duplicates, or possibly they would be moved to a bucket named dbp-removed-{year}.