Crisp Take Home Project Summary

Brenda Kao
07/31/2023

Deliverables

- Running code and test suite provided through online code repo or in a tar-ball

  => tar-ball file name: etl-project.tgz
  => GitHub repository: https://github.com/bkao000/etl-project
  => One application etl-app and one library etl-lib are included in the delivery with tests suite and test cases.

- Instructions on how to build and run the code with example data

  => 1. Uncompress the tar ball or the downloaded project zip from GitHub URL provided.
  2. Set up environment variable ETL_PROJECT to the path of project folder (eta-project).
  3. See "README.md" file in "${ETL_PROJECT}/etl-app" folder.

- *Short* architectural overview and technology choices made

  => See "etl-project-design.pdf" in ${ETL_PROJECT}/etl-app/doc

- (Basic) documentation, unless it's completely self-documenting (to a fellow software developer)

  => Requirements and specification: "Crisp - Engineering Manager Take Home Test.pdf"
  => Application design document: "etl-project-design.pdf"
  => Comments in code.

- List of assumptions or simplifications made

  => A row is considered as good when all the target fields can be published based on the definitions in yaml file. Other unused source fields contents will not be checked.
  => A row precess will stop when the first field transformation error is hit. The rest of fields will not be checked.
  => If a target field should be set up with a default value but it is not defined in the yaml file, the whole job will stop when this error is hit for the first time.

=> Each target field in yaml file should contain required attributes: src (source field), tgt (target field), type(target field type).

=> Source formats of numeric data are not checked. Rely on conversion functions to capture errors.

=> A test yaml file "order-product-name-no-src-format-check.yaml" is created for the following scenario. How to run it is documented in "README.md".

Since the source data contains leading lowercase character in "Product Name" I assume the data doesn't compliant with the source format definition "[A-Z]+". Instead of filtering out the invalid source data I transform the data to uppercase to meet the requirement in target.

- List of the next steps you would want to do if this were a real project

=> Convert "etl_order.py" to a framework to handle all similar type of data sets by taking table name/data set name such as 'order' or 'user' as a parameter. This parameter can be used to locate the source file, configuration file, output file and error log file in the predefined folder structure.

=> Config jobs to run in a scheduler such as Airflow. Set up dependencies between a job and its upstream and/or downstream. Set up job monitors and alerts.

=> Add data quality checks and alerts.

=> Create a dashboard/report to show daily production jobs/pipelines and data quality status.

=> Write a tool to review and validate yaml file to ensure it is in the correct format and contains all required definitions such as src field, target field, target type.

=> Create a configure file or table to register ETL pipelines. Scheduler can pick up jobs to run from the registry.

=> Add dependency check to ETL pipelines to ensure each stage of a pipeline starts with good upstream data.

Requirements (See "Crisp - Engineering Manager Take Home Test.pdf" for details.)

- The transformations should be configurable with an external DSL (like a configuration file)

=> Source fields, target fields and source to target mappings are defined in a yaml file - "order.yaml".

- The functionality should be implemented as a library, without (significant) external dependencies

=> "etc-lib" library is created as a package. It contains the etl transformation functions.

- Invalid rows should be collected, with errors describing why they are invalid (logging them is fine for now)

=> Invalid rows are collected in an error file - "err-order-output.csv" with error description, error field, error row data.

- The data tables can have a very large number of rows

=> If source row volume is big consider to break it into smaller chunks and to process it more frequently. Rely on one task to process a big volume of data has some drawbacks as below.

- The task might hit the resource limit as data grows over time,
- If the job fails it will take a long time to rerun it.
- Big volume of source data and target data could make the troubleshooting cumbersome.