# **Title**: Step by Step document to the ETL Process using S3 files

## **Author**: Imanpreet Singh

## **Date**: 27th Jan 2022

## **Change Logging:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version |  | Author | Date | Description |
|  |  |  |  |  |
|  |  |  |  |  |

## **Scenario**:

Every weekday stock open and close price is provided.

Stock open price is available in a database. (Table = **int.openstockprice**)

Stock close price file is provided at the end of the day in a csv file (name = **closestockprice.csv**) and loaded in the landing area.

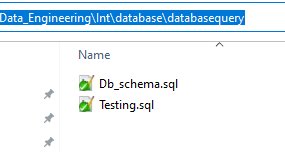
## **Requirement**:

At the end of the day, open stock price and close stock price needs to be loaded into one table in the database over which data analysis can be done. (Table = int.stockprice)

## **Pre-requisites:**

Before using the scripts and modules please follow below step to set up the system.

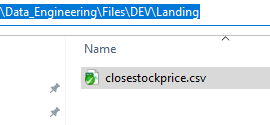
* Create Database Schema. Use ‘**Db\_schema.sql’** from below path.



Insert Stock open price data in the table **int.openstockprice** for current date using **Testing.sql** script

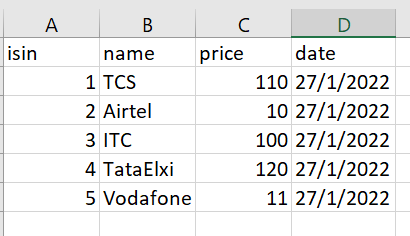


* Update below file in the landing area with the current date data.:

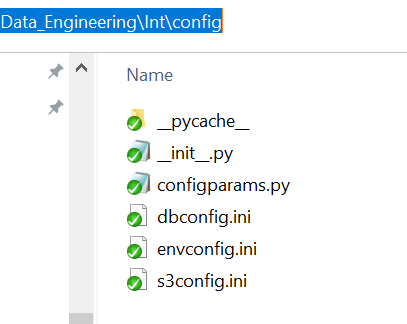


For 27th Jan below is the example.

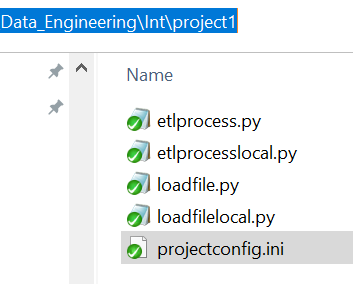
*As it is a stock price data, at weekend there will be no file.*



* Update config files as per your system:
  + **dbconfig.ini:** Update configuration about your source and target database. In our case, both are same.
  + **envconfig.ini:** Update configuration that will change as and when moved to different environments. e.g., archive path, landing path, file path etc.
  + **s3config.ini:** Update configuration about your S3 credentials.



* Update config files related to the project you are working.
  + **projectconfig.ini:** For our **stock price** solution, we need to provide source and target tables, file name etc. that can be provided here and will not be specific to other projects.



## **Testing:**

Once pre-requisites are met, we are ready with our process.

### **Load file from Landing to S3 bucket:**

#### Script - loadfile.py environment project

* + Input variable
    1. environment 🡪 DEV
    2. project 🡪 stockprice
  + **python C:\Git\Data\_Engineering\Int\project1\loadfile.py DEV stockprice**
  + This script will accept environment name and project name. This script can be used across different projects.

This will perform following tasks:

1. Load the file from landing area to S3 bucket.
2. This will check with the S3 bucket existence and create it if not there.
3. If there is already a file in S3 bucket, then it will archive that file first to Archive bucket and add stock price date as suffix to the file. e.g., if file contains 27Jan 2022 data then that file will be archived as filename\_27012022.
4. If Archive Bucket is not there it will create that bucket
5. Once file is archived then file is moved from landing to process Bucket

### **Perform ETL process:**

#### Script - etlprocess.py environment project

* + Input variable
    1. environment 🡪 DEV
    2. project 🡪 stockprice
  + **python C:\Git\Data\_Engineering\Int\project1\etlprocess.py DEV stockprice**
  + This script will accept environment name and project name.

This will perform following tasks:

##### Get the date for which data need to be processed.

* + 1. For first time, it will be current date
    2. Post that, maximum date from the target tables - int.stockprice is fetched and a day is added to it.
    3. If new date is weekend, then accordingly dates are added to get next weekday. (this is to make sure we are not loading same day into target table)

##### Load the data from database into the data frame – **open stock price data.**

1. Create WHERE condition to get the data from the database based on the date expected (as per step 1)
2. Create the select query based on the configuration and where condition (only columns defined in config will be fetched)
3. Use the query to get data from the database into data frame.
4. Check if data is there or not
5. Standardize DATE columns and PRICE columns

##### Load the data from csv file in the process bucket into data frame. – **close stock price data.**

1. Check bucket and file exit
2. Check file is not empty
3. Check the date present is for the date expected (as per step 1)
4. Check the columns for mandatory columns i.e., columns matching the open price data frame.
5. Standardize DATE columns and PRICE columns

##### Join both the data frames to create a final data frame containing both open and close stock price.

1. Derive day, month, year part from the date column and add it as separate column in data frame
2. Sort the column as per the target table column list. This target column list is defined in the config.

##### Load the data into the target table **int.stockprice**

1. Generate the dynamic insert query based on the schema, table, and column list.
2. Insert the data using this query

### **Repeat the process for next day:**

* Load the Open price data in the database for next day.
* Using Testing.sql. In our case next day data is 28th Jan 2022.
* Load close price file in the Landing area for next day manually
* Load the file from Landing area to S3 Bucket by running script
  + loadfile.py environment project script to
* Load the open price and close price data with all required transformation into target table by running below script.
  + etlprocess.py environment project

## **Package and Underlying Module Details:**

*(\*\* are for Python Dictionary Arguments, where we pass Dictionary as input parameter)*

### **configparams**

#### ConfigParams

* + - * + To get parameters from configuration file and return it them as Python Dictionary

Input parameters - Config file, Section

Returns – Python dictionary

### **database**

#### GetData

* + - * + To get data from the database in the form of Python Data Frame

Input parameters - query, logfile, \*\*dbparams

Returns – Python data frame

#### GetInsertQuery

* + - * + To get data from the database in the form of Python Data Frame

Input parameters - schemaname, tablename, columnlist, logfile

Returns – String

#### PutData

* + - * + To get data from the database in the form of Python Data Frame

Input parameters - query, data frame, logfile, \*\*dbparams

Returns –

### **localfile**

#### PutFile

* + - * + To move file from one folder to another and archive the file if already present in the target path.

Input parameters - filesourcepath, filetargetpath, filearchivepath, filename, logfile

Returns -

### **awsS3**

#### ArchiveFileS3

* + - * + To archive an S3 file to Archive bucket. Archive bucket for any bucket will be - bucketname+'archive'.

Input parameters - bucketname, filename, logfile, \*\*s3params

Returns –

Dependency on Module - **CreateBucketS3, CopyFileS3, DeleteFileS3, GetFileDatas3**

#### CopyFileS3

* + - * + To copy file from one S3 bucket to another S3 bucket.

Input parameters - sourcebucket, targetbucket, sourcefilename, targetfilename, logfile, \*\*s3params

Returns –

Dependency on Module -

#### CreateBucketS3

* + - * + To move file from one folder to another and archive the file if already present in the target path.

Input parameters - bucketname, logfile, \*\*s3params

Returns –

Dependency on Module -

#### DeleteFileS3

* + - * + To delete file from a S3 bucket.

Input parameters - bucketname, filename, logfile, \*\*s3params

Returns –

Dependency on Module -

#### GetFileDatas3

* + - * + To get csv file data in S3 bucket and return it in Python Data frame.

Input parameters - bucketname, key, logfile, \*\*s3params

Returns –

Dependency on Module - **CreateBucketS3**

#### PutFileS3

* + - * + To load file from local folder to S3 bucket.

Input parameters - bucketname, filepath, filename, logfile, \*\*s3params):

Returns –

Dependency on Module - **CreateBucketS3, ArchiveFileS3**