**Program/Module Overview:**

Main script that is triggered whenever a csv file is uploaded in the required S3 Buckets

**Input File(s):**

Dimension Table CSV’s:

1. Entity table
2. School table
3. Student table
4. Teacher table
5. Parent table

Fact Tables CSV’s:

1. StudentPerformance.csv
2. StudentDiscipline.csv
3. StudentHealth.csv
4. StudentAttendance.csv
5. StudentFeedback.csv
6. TeacherPerformance.csv
7. TeacherFeedback.csv
8. ParentFeedback.csv

**Processing Steps:**

1. The bucket name and name of the CSV file that has been uploaded to the S3 bucket from the event information

(101).Using the AWS-SDK for python (boto3) the uploaded CSV file object is loaded into the lambda script

1. An array of records is created using the CSV file object
2. An SQL transaction is started and the generated transaction id is used to associate the executed queries with the transaction
   1. A connection is not established when first interacting with the database as the database cluster is inactive when idle, but this first interaction puts the cluster in a active state so the script is coded to retry 3 times to make successful connection to the database.
3. Based on the file name different scripts are run, if the file name was of a Dimension table then sf\_Dimension\_table\_aurora\_load.py is run else if the file name was of a Fact table then sf\_Facts\_table\_aurora\_load.py us run
4. Transaction is committed if insertion and deletion were successful otherwise all changes are rolled back

**Output File(s):**

*-*

**Design / Development Comments:**

The encoding had to be changed from ***‘utf-8’*** to ‘***utf-8-sig’*** to ignore the BOM character when reading a file.

**Program/Module Overview:**

Contains common functions for building parameter sets to for SQL query and functions to create the queries too

**Input File(s):**

-

**Processing Steps:**

1. Build\_parameters\_final: Creates parameter sets and does string processing to convert the values into appropriate format and datatypes based on the column types. Parameter sets are built according to the boto3 syntax (<https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/rds-data.html>)
2. build\_insert\_query : creates the insert query using the table name and the column headers

**Output File(s):**

-

**Design / Development Comments:**

ProcessDate column parameter is added separately because the value for it means the timestamp when the records were added to the database. This timestamp is set in “build\_parameters\_final” function using the datetime library

**Program/Module Overview:**

Contains common functions for running the SQL queries on the database using AWS SDK for python (boto3)

**Input File(s):**

-

**Processing Steps:**

1. runbquery: Runs the batch\_execute\_statement method of RDS data service client from boto3. Used to execute all batch insert queries and delete queries ([https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/rds-data.html#RDSDataService.Client.batch\_execute\_statement](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/rds-data.html%23RDSDataService.Client.batch_execute_statement))
2. runquery: Runs the execute\_statement method of RDS data service client from boto3,

used for running some delete queries and the show query ([https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/rds-data.html#RDSDataService.Client.execute\_statement](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/rds-data.html%23RDSDataService.Client.execute_statement))

1. count\_all\_records: returns the number of records in particular table

**Output File(s):**

-

**Design / Development Comments:**

-

**Program/Module Overview:**

Module is used when a file containing records of a Dimension table is uploaded and then inserts the records into respective tables

**Input File(s):**

-

**Processing Steps:**

1. Column types and Column headers are stored from the response of the “SHOW columns from {table\_name}” query.
2. Uses build\_parameters\_final and build\_insert\_query from sf\_Build\_parameters\_aurora\_load.py module to create parameter sets and insert query for the boto3 batch execute statement.
3. A delete query is run to delete old records of same primary keys that are to be inserted to update the records with the latest data
4. runbquery function from the sf\_Run\_query\_aurora\_load.py are used to execute batch insert and delete queries.
5. count\_all\_records function from the sf\_Run\_query\_aurora\_load.py is used to calculate the number of records of a table to check if records inserted after deletion of duplicates is equal number of records extracted from the CSV

**Output File(s):**

-

**Design / Development Comments:**

Python’s boto3 client is being used to connect with RDS Database Cluster. The Cluster ARN and the Secret ARN with the stored database credentials are used to gain access and query the database. After connecting with a RDS through boto3 we can use ‘batch\_execute\_statement’ to run a batch of SQL statements on the Aurora DB and is more efficient than the ‘execute\_statement’ which can only query 1 SQL statement.

**Program/Module Overview:**

Module is used when a file containing records of a Fact table is uploaded and then inserts the records into respective tables

**Input File(s):**

-

**Processing Steps:**

1. Very similar to sf\_Dimension\_table\_aurora\_load, main difference is how the deletion of records is handled.
2. Current month’s records if present, are deleted before inserting new records from the same month. As data in these table is only added once a month any duplicate addition like this means the current month’s records in table needs to be updated by deleting old records and adding new.

**Output File(s):**

-

**Design / Development Comments:**