PyETL: ETL Pipeline with python **July 2020**

Database Design Amin Rezaei

This project, proposes an ETL Pipeline implemented in Python. An ETL Pipeline is pipeline getting the data from source datastore and storing them efficiently in another datastore called Data Warehouse. This project aims on a PostgreSQL to PostgreSQL ETL but it is easily extensible due to high abstraction units. Source Code is descriptive and well documented, for more clarification a brief overview of usage will be proposed:

- **SqlLoader class – package util**:

  This class reads SQL commands defined in a special format from a sql file and provides a pick method that gets the command from file with provided key and replaces the parameters given. It's a utility object and provides easy way to get SQL command without repeating it in code.

- **PostgresAccessor – package db**:

  This class handles connection to a PostgreSQL database instance and efficiently handles the connection.

- **Classes in package base**:

  These classes provide the abstract level declarations of ETL Components which make us capable to extend this project to work with almost any datastore.

- **Packages extractors, transformers, loaders:**
  These packages and corresponding classes are for extracting data from source and applying transformation then loading the result in destination.

- **Class PostgresConsistencyFixer - package fixer**:

  This class checks consistency between source and destination and makes sure that after the changes everything is consistent if not it fixes the inconsistencies.

- **class Pg2PgPipeline**:

  Implements a pipeline which its source and destination are PostgreSQL databases.

Implementation Details:

All implementations are highly flexible and no hardcoded procedure is included. Tables, relations and database structure are extracted using appropriate SQL commands.

```
--! ListTables:
SELECT
   tablename
FROM
   pg_catalog.pg_tables
WHERE
   schemaname != 'pg_catalog'
AND schemaname != 'information_schema';
--! TableColumns:
SELECT
   column_name, udt_name, is_nullable, character_maximum_length
FROM
   information_schema.COLUMNS
WHERE
   TABLE_NAME = '${table}' ORDER BY ordinal_position;
--! TableKeys:
SELECT key_column FROM (SELECT kcu.table_schema,
       kcu.table_name,
       tco.constraint_name,
       tco.constraint_type,
       kcu.ordinal_position AS position,
       kcu.column_name AS key_column
FROM information_schema.table_constraints tco
JOIN information_schema.key_column_usage kcu
     ON kcu.constraint_name = tco.constraint_name
     AND kcu.constraint_schema = tco.constraint_schema
     AND kcu.constraint_name = tco.constraint_name
WHERE tco.constraint_type = 'PRIMARY KEY'
ORDER BY kcu.table_schema,
         kcu.table_name,
         position) AS X WHERE table_name = '${table}';
--! ForeignKeyRelations:
SELECT DISTINCT
    tc.constraint_name,
    kcu.column_name,
    ccu.table_name AS foreign_table_name,
    ccu.column_name AS foreign_column_name
FROM
    information_schema.table_constraints AS tc
    JOIN information_schema.key_column_usage AS kcu
      ON tc.constraint_name = kcu.constraint_name
      AND tc.table_schema = kcu.table_schema
    JOIN information_schema.constraint_column_usage AS ccu
      ON ccu.constraint_name = tc.constraint_name
      AND ccu.table_schema = tc.table_schema
WHERE tc.constraint_type = 'FOREIGN KEY' AND tc.table_name = '${table}';
--! AllTablesForeignKeyRelations:
SELECT DISTINCT
        tc.table_name,
        ccu.table_name AS foreign_table_name
    FROM
        information_schema.table_constraints AS tc
        JOIN information_schema.key_column_usage AS kcu
          ON tc.constraint_name = kcu.constraint_name
          AND tc.table_schema = kcu.table_schema
        JOIN information_schema.constraint_column_usage AS ccu
          ON ccu.constraint_name = tc.constraint_name
          AND ccu.table_schema = tc.table_schema
    WHERE tc.constraint_type = 'FOREIGN KEY';
```

These are the SQL commands for extracting database structure (PostgreSQL).

One important issue is the order that we extract and load the data to destination because of relational nature of these databases operations should be in right order to avoid inconsistency or errors. To resolve this issue first we extract all Foreign Key relations using **AllTablesForeignKeyRelations** query, then we form a **DAG** from the dependencies at last we sort them in Topological Order that is the desired order we seek. We can consistently do Insert/Update/Delete operation with the obtained order.

Psycopg2 library is used for database connection and **networkx** library is used for obtaining topological sort. Time Machine is also implemented and can be used (Report in the other file).

Final Usage:

This code shows the usage example to run ETL for two PostgreSQL databases, connection parameters can be configured as proposed. Time Machine feature can optionally implement and create the corresponding structures on destination.

```
from pipeline.pg2pg import Pg2PgPipeline
from util.constants import ResourceConsts
from util.global_resources import GlobalResources
from util.sql_loader import SqlLoader

if __name__ == "__main__":
    resources = GlobalResources()
    sqlLoader = SqlLoader("data/commands.sql")
    resources.add(ResourceConsts.COMMANDS_LOADER, sqlLoader)
    pipeline = Pg2PgPipeline()
    # connection is connection params:
    '''
        - host: str = "localhost"
        - port: int = 5432
        - db: str = "postgres"
        - user: str = ""
        - pwd: str = ""
    '''
    pipeline.run(constructing=True, src_connection={}, target_connection={
        "db": "library"
    }, time_machine=True)
```

References:

- https://dataedo.com/kb/query/postgresql/list-all-primary-keys-and-their-columns
- https://stackoverflow.com/questions/1214576/how-do-i-get-the-primary-keys-of-a-table-from-postgres-via-plpgsql
- https://www.postgresql.org/docs/10/infoschema-columns.html
- https://www.postgresql.org/docs/9.1/infoschema-tables.html
- https://www.postgresql.org/docs/9.1/information-schema.html
- https://networkx.github.io/documentation/networkx-2.4/reference/algorithms/generated/networkx.algorithms.dag.topological_sort.html
- https://kb.objectrocket.com/postgresql