CentralNic Group PLC

Private and Confidential

Data Engineer: Test Assignment

Hello,

The next step in our hiring process is to complete and submit coding for evaluation. This is one of the most important steps in our hiring process because we are a technology-driven company that takes pride in what we create.

Submission instructions

1. Submit a zip file with all your code and documentation to pawel.rzeszucinski@centralnic.com, mirco.pyrtek@centralnic.com and asterios.merkos@centralnic.com.
2. Include a README.md with explanations on how the files are organized, how to install the module and how to run the logic included. Also, include a short write-up in the same README.md explaining your thought process in producing the scripts. List possible problems you can foresee in this kind of workflow and if you have further considerations that you had no time to implement, please explain.

Feel free to ask clarification questions on the desired requirements within the three days period.

Task description

- The project has to be written in the Python programming language, version 3.
- In summary, this project is about creating a CLI module that allows the user to perform a simple ELT process based on multiple database tables.
- For simplicity we will be using SQLite[1] as a database, so the test is self-contained.
- Attached you can find a database file (*'source.db'*) with data originating from an UCI Machine Learning Repository dataset called "Online Retail Data Set[2] that can be

[1] See https://docs.python.org/3/library/sqlite3.html for further reference

[2] Dr Daqing Chen, Director: Public Analytics group. chend '@' lsbu.ac.uk, School of Engineering, London South Bank University, London SE1 0AA, UK., https://archive.ics.uci.edu/ml/datasets/online+retail

imported into SQLite. Within the database, there are 3 different tables that have to be joined together in order to extract sample files for further processing.

- The first table is called `transactions` and includes information about transactions occurring between 2010-12-01 and 2011-12-09 for a UK-based and registered non-store online retail. All of those transactions are in the same currency. The respective columns are:
    - transaction_id → Unique identifier of the transaction.
    - invoice_id → The invoice identifier this transaction belongs to.
    - stock_code → The stock code.
    - description → Product description.
    - quantity → The amount that was purchased.
    - invoice_date → The datetime the transaction was invoiced as an ISO string (YYYY-MM-DDThh:mm:ss.000Z).
    - unit_price → The price of a single unit of the product with a given description.
    - customer_id → The customer that has purchased the given quantity of the product.

transaction_id	invoice_id	stock_code	description	quantity	invoice_date	unit_price	customer_id
1	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01T08:26:00.000Z	2.55	17850
2	536365	71053	WHITE METAL LANTERN	6	2010-12-01T08:26:00.000Z	3.39	17850
3	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01T08:26:00.000Z	2.75	17850
4	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01T08:26:00.000Z	3.39	17850
5	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01T08:26:00.000Z	3.39	17850

- The second table is called `customers` and maps the customer_id from the transactions table to the customer's country name. However, the country still needs to be mapped to its ISO 3166-1 alpha-2 code representation.
    - customer_id → The customer's unique identifier.
    - customer_country_name → Country of the customer.

customer_id	customer_country
17850	United Kingdom
13047	United Kingdom
12583	France
13748	United Kingdom
15100	United Kingdom

- The third table is called `countries` and maps the country name to the ISO 3166-1 alpha-2 code:

- country_2_alpha → The ISO 3166-1 alpha-2 code of a specific country.
- country_name → The written name of the country.

country_2_alpha	country_name
AF	Afghanistan
AX	Åland Islands
AL	Albania
DZ	Algeria
AS	American Samoa

- The final extracted features should look like the following:
  - transaction_id → A unique transaction identifier
  - customer_id → A unique customer identifier
  - customer_country → The country of the customer as an ISO 3166-1 alpha-2 code
  - datetime → Datetime in ISO format as (YYYY-MM-DDThh:mm:ss.000Z)
  - product_description → Description of the product being sold
  - quantity → The amount that was purchased.
  - unit_price → The revenue of the transaction is quantity * unit_price
- Your assignment should include all the files of your final result (target database, extracted files, etc.) as well as your code.

Part 1: Create extraction logic

The first part of your assignment is to design and implement the extract logic of your module. Given an SQL query stored in a file, and some specific params that are passed as named arguments to that query, the module should apply the query to the given SQLite database and save the resulting data to a CSV file given as the target. In this specific task, the module should extract all transactions of a given date, so the date has to be a parameter to be passed to the SQL.

- Command name: `eltcli extract`
- Parameters:
  - database: The file where the source database is located
  - sql: The query to be executed

  - params: The params for the SQL file.
  - target: The target file where to store the resulting data in CSV format.
- An example call in the CLI could look like the following:

```
eltcli extract --database source.db --sql transactions.sql
--params '{"date" : "2010-12-01"}' --target
transactions_2010-12-01.csv
```

- **Finally**, create the SQL query to extract data for 2010-12-01 and 2010-12-02 using your module and save the resulting files.

Part 2: Create load logic

The second part of the assignment is to create a script that imports the transactions of the CSV files that were extracted in the previous step into an SQL table.

- Command name: `eltcli load`
- Parameters:
  - input: Path to the file with the input data in CSV format.
  - database: The database file where to put the given data.
  - target: The target table of the data to be loaded.
- An example call in the CLI could look like the following:

```
eltcli load --input transactions_2010-12-01.csv --database
target.db --target transactions
```

- It should put the information into the DB, in a table that is specified as an argument. All data from CSVs imported by the script will be placed in this same table.
- It should handle duplicate rows (i.e. same CSV imported twice by mistake or update existing data by processing a new file).
- Import the two files extracted before to a new database (e.g. target.db) using your module.

Part 3: Create transformation logic

The third part of the assignment is to create a module that is able to transform the imported data by passing an SQL query as a file. The SQL query that you have to specifically create should read from the SQL table from the previous part and aggregates data into two new DB tables that allow getting the following information:

- Aggregated by user/day: Number of operations and revenue per User per Day
- Aggregated by user/hour: Number of operations and revenue per User per Hour
- Command name: `eltcli transform`
- Parameters:
  - database: The file where the database to operate on is located.
  - sql: The sql file to execute in order to perform the transformation task.
  - params: The params for the sql file.
  - target: The target table in the same db where to put the resulting data.
- Running the script should look like:

  ```
  eltcli transform --database target.db --sql
  aggregation-user-day.sql --params '{"date" : "2010-12-01"}'
  --target transactionsdaily
  ```
- Create both SQL queries and aggregate transactions for dates 2010-12-01 and 2010-12-02 as well as for all hours of a day (0 - 23). Store the aggregated data in two separate tables within your target database.

Part 4: Module testing & deployment

The final part of your test assignment is about writing appropriate unit tests for your code (e.g. using pytest) and applying code quality tools such as mypy for static type checking. The following outcomes are expected from you in this final exercise:

- Build the script as a python CLI project including tests with coverage >= 80%.
- Add at least one module to analyze the code quality of your module.
- Properly document how to install and execute your module in markdown.