[STADVDB] Hands-On Activity 1
Building a Data Warehouse with an ETL Tool

Daniel III Ramos

January 2024

1 Summary

This activity is my first exposure to the whole Extract-Transform-Load (ETL) pipeline. The process involves extracting data from heterogeneous sources, transforming them to be fit for further analysis, and loading them into a single database known as a data warehouse.

The project specifications require Apache NiFi as the ETL tool. However, using the tool poses some problems than initially realized. Some issues were minor like the archaic, confusing installation and unavailable databases. These issues were resolved relatively quickly. However, the most glaring issue would be the fact that I had to learn a new tool with all its intricacies in only a week.

The FAQs offered an alternative approach–to code your own ETL tool. Gauging the pros and cons of the two methods, I opted for the latter. The learning curve for coding your own ETL tool is way lower than learning new software since, at this point, I already have the prerequisite knowledge to code while I have zero knowledge of Apache NiFi. The method also exercises skills in my major, Software Technology.

For my software solution, I opted for Python due to its rich library ecosystem and ease of use. The software design is explained in the following section. The software is uploaded to a public repository: `https://github.com/dhannn/minipytl`.

The activity exposed me to what I assume are standard processes. It makes me see how data flows in my everyday life. I might even use some concepts I learned in my non-academic projects as explained in a future section.

2 Background

In this project, I gathered data from three different sources to a central repository, known as a data warehouse, with the aim of further analysis. The sources include a flat CSV file, a relational (MySQL) database, and a document-oriented (MongoDB) database. To create our data warehouse, we might need to explore some concepts that might help us.

First, we need a process called **Extract-Transform-Load** (ETL). In this process, we first *extract* our data sources by accessing them separately. The *data sources* get loaded to a transient storage known as the **staging area**. Then, we *transform* the data to prepare them for analysis in our data warehouse. Finally, we load our transformed data to a singular database known as the *data target*. The data warehouse gets updated periodically using the same process.

There exist many tools for the ETL process. The project suggested Apache NiFi for our ETL tool.

3 Problems

Our goal of gathering data for a data warehouse was met with problems. Most problems arise from the tool prescribed, specifically, its outdated nature and, more importantly, the learning curve required.

In this section, I quickly discuss the outdated nature. Then, I discuss how the learning curve for the tool poses a problem and the general approach I took instead.

3.1 Archaic Tool

As stated earlier, Apache NiFI is suggested as the ETL tool. However, the tool proved to be hard to set up. Most modern software is configured to be seamlessly installed. But Apache NiFi requires a lot from the user such as going to a log file to get access to a username and password. Such information will be used to access an HTTP file from the web which I find odd. Again, this is minor and only served as a lesson for software usability.

3.2 Learning Curve Problem

My major issue for this project is the learning curve for using Apache NiFi. We only have a week to gather the data sources and load them into the data warehouse. Thus, learning a whole new tool with its components might not be feasible without cramming. Fortunately, an alternative approach is offered on the FAQ page.

Instead of using the Apache NiFi, we can code our own ETL tool. While this may seem like more work, I suppose that I have the prerequisite skills to do the task at hand. I know how to interface with both relational and document-oriented databases from my Information Management and Web Application Development courses. My Modeling and Simulation course taught me how to work with data from CSV and manipulate them through `pandas DataFrame`s.

Unlike with Apache NiFi, I was not starting with zero knowledge. I just have to combine all the disparate skills I acquired to achieve our objectives. It also aligns with my major as a Software Technology student.

So for this project, I decided to create a minimal Python library that emulates the ETL process.

4 MiniPyTL: My Software Solution

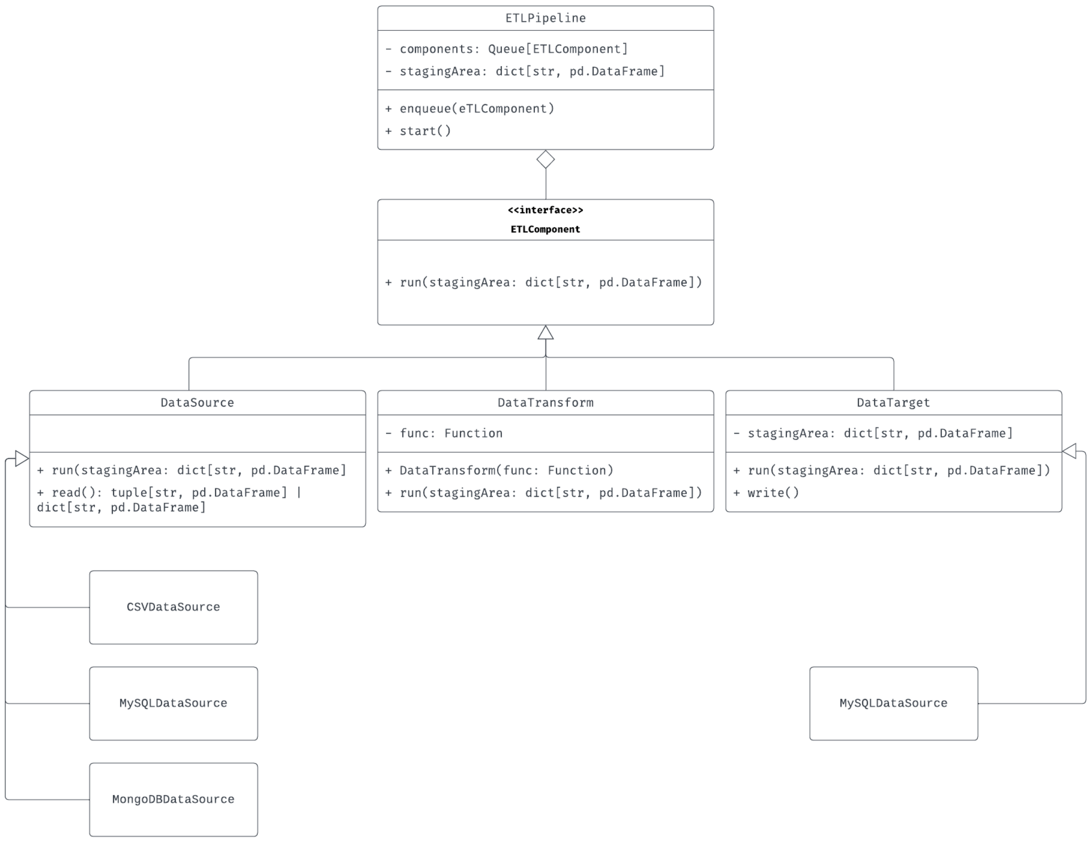

ETLPipeline
- components: Queue[ETLComponent]
- stagingArea: dict[str, pd.DataFrame]
+ enqueue(eTLComponent)
+ start()
<<interface>>
ETLComponent
+ run(stagingArea: dict[str, pd.DataFrame])
DataSource
+ run(stagingArea: dict[str, pd.DataFrame]
+ read(): tuple[str, pd.DataFrame] | dict[str, pd.DataFrame]
DataTransform
- func: Function
+ DataTransform(func: Function)
+ run(stagingArea: dict[str, pd.DataFrame])
DataTarget
- stagingArea: dict[str, pd.DataFrame]
+ run(stagingArea: dict[str, pd.DataFrame])
+ write()
CSVDataSource
MySQLDataSource
MongoDBDataSource
MySQLDataSource

Figure 1: Class diagram of MiniPyTL

In this chapter, I outline my software solution and its components. Figure 1 shows the class diagram of the software. It has the following components:

- `ETLPipeline`
- `ETLComponent`

- `DataSource`
- `DataTransform`
- `DataTarget`

4.1 Components

4.1.1 `ETLPipeline`

The `ETLPipeline` class handles the whole ETL process. It enqueues the components to be run and goes through each component to process the data. It contains the staging area which holds the temporary collection of `DataFrame`s.

4.1.2 `ETLComponent`

The class represents operations on data such as extraction (`DataSource`), transformation (`DataTransform`), or loading (`DataTarget`). The `run()` method is overridden based on the exact operation. The method modifies the staging area by adding a `DataFrame`, transforming an existing DataFrame or dumpng the staging area to the target.

4.1.3 `DataSource`

The class encapsulates the extraction of data. It has a `read()` function that returns a tuple containing the name of a table and its corresponding `DataFrame`, or a dictionary whose keys are the name and values are their `DataFrame`.

The `read()` method is meant to be implemented based on what source we are extracting from. Currently, we have the following implementations: `CSVDataSource`, `MySQLDataSource` and `MongoDBDataSource`.

4.1.4 `DataTransform`

The class is responsible for transforming our current data. For flexibility, the class is designed such that, upon instantiation, we pass a function that modifies the staging area. When the `ETLComponent.run()` is called, the object simply calls the function.

4.1.5 `DataTarget`

The class encapsulates the loading of the data. It has the `write()` class which takes the staging area and dumps everything into the specified database. Like the `DataSource` class, it should be overridden based on the type of database of the data warehouse.

For this example, I only implemented a `MySQLDataTarget` subclass since the project only requires MySQL for the data warehouse. However, this should be pretty extensible to other database types.

4.2 Sample Usage

In the GitHub repository, the `./src` directory contains a sample script (`sample.py`) on how to use it.

First, we initialize our ETL pipeline.

```
pipeline = ETLPipeline()
```

Then, we create an instance of a `DataSource` subclass. In this case, we are instantiating a `CSVDataSource`.

```
filename = './data/26k-consumer-complaints.csv'
csvDataSource = CSVDataSource('complaints', filename, 'Complaint ID')
pipeline.enqueue(csvDataSource)
```

Let us create a data target to load our CSV data.

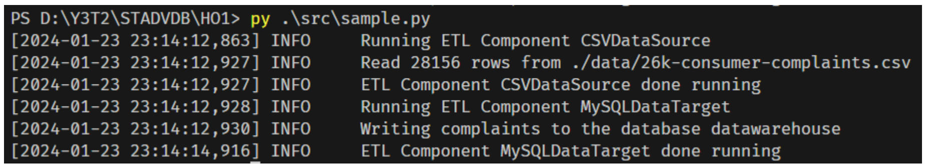
```
config = {
    'user': os.getenv('MYSQL_USER_DST'),
    'password': os.getenv('MYSQL_PASS_DST'),
    'host': os.getenv('MYSQL_HOST_DST'),
    'database': os.getenv('MYSQL_DB_DST')
}

mysqlDataTarget = MySQLDataTarget(config)
pipeline.enqueue(mysqlDataTarget)
```

Simply call the `run()` method of the pipeline to activate the process.

```
pipeline.start()
```

Running the snippet yields the following log results (Figure 2).



```
PS D:\Y3T2\STADVDB\HO1> py .\src\sample.py
[2024-01-23 23:14:12,863] INFO     Running ETL Component CSVDataSource
[2024-01-23 23:14:12,927] INFO     Read 28156 rows from ./data/26k-consumer-complaints.csv
[2024-01-23 23:14:12,927] INFO     ETL Component CSVDataSource done running
[2024-01-23 23:14:12,928] INFO     Running ETL Component MySQLDataTarget
[2024-01-23 23:14:12,930] INFO     Writing complaints to the database datawarehouse
[2024-01-23 23:14:14,916] INFO     ETL Component MySQLDataTarget done running
```

Figure 2: Logs of a sample program

Note that the code snippets here are simply excerpts. Should you be interested in looking at how the other data sources work and the data transformation, simply refer to the GitHub repository.

4.3 Limitations

While the software successfully extracts, transforms and loads the data sources to the target, there are several limitations that hinder the library from rivaling established tools such as Apache NiFi. This includes the following:

- **Lack of mechanism for periodic updates.** As stated in the Background section, ETL process should ideally update data from time to time. However, for this implementation, there is not enough time to include this feature.

  Out of all the limitations, this is the easiest to implement as Python, or even in operating systems, has plenty of mechanisms that allow for task scheduling.

- **Type safety and data integrity.** Ideally, we should enforce typing and data integrity in our data warehouse. Such features include foreign key constraints and explicit.

- **Processing data of large volume.** While I used `pandas DataFrame` and the implementation of `DataFrame` operations are optimized, there could be some room for improvement.

  One specific instance of this is in our transform function which flattens the MongoDB collection to several tables in a relational database. I do not have time to make it performant so I decided to log in to signify that it's an expensive calculation. There could also be potential to harness parallel computing.

- **Error handling.** Good software is defensive and so, it ought to consider edge cases through error handling. Like the past points, I simply do not have time to add error handling and test-driven development.

With enough time, it is possible to develop robust features for the library. Despite the limitations, however, designing and implementing the library still serves as a valuable experience in handling data. Doing this project gives me a higher-level idea of how data could flow in an analytics environment.

5 Insights and Conclusion

After doing the activity, I gain more knowledge and skills in data warehousing and extracting data from heterogeneous sources. This activity shifts the way I see data and how to manage them, especially in contexts where I did not initially consider these concepts.

For example, I get tasked by my relative to analyze what factors contribute to overtime in their workplace. The lessons covered gave me a framework to follow. It provides me with the necessary vocabulary to discuss the problem and possible solutions in depth.

With this newly acquired knowledge, I understand that we might consider data warehousing of employee work logs, information about employees (such as what "Tower" or Career Level) and possibly time-related information (since they follow a different cyclical process each month). Due to their heterogeneous nature, these data sources should be extracted, transformed and loaded into our data warehouse for further analysis. Taking it further, I might consider retrofitting the schema of the current database to follow the star or snowflake schema depending on their needs.

In conclusion, I can recontextualize what I learned in this activity to novel situations that does not involve the current context.

6 Declaration of AI Use

In accordance with the directives of the Provost, which encourages the transparent use of AI technology to enhance projects, I hereby declare that I use AI technology to augment this hands-on activity. The following use cases illustrate my application of AI:

- Brainstorming on possible software design and architecture
- Clarity and guidance on implementation details (for example, to normalize or not, to use multithreading or not, how customizable the design should be)
- Feedback on writing
- Automating adding code documentation

For transparency, the following link shows the scope of AI use: https://chat.openai.com/share/e4280f73-f1e0-4b0c-8364-14c439203b95.

I guarantee that all aspects of the project beyond the aforementioned scope are entirely original and free from external influence.