Data Warehouse Setup (Task A)

This task entails the initialization of the data warehouse in particular, the creation of the warehouse schema and the tables based from the GOSales dataset. To perform the construction of the data warehouse schema, the group utilized MySQL Workbench, a MySQL development tool to facilitate a multitude of MySQL database system features. Additionally, this tool is also used in the creation of the table schemas themselves. The overall process took an insignificant amount of time as this task only encompasses the formation of schemas which can be done by an existing database tool.

In doing this task, no major challenges were encountered due to the familiarity of every member in the database and its tools. It is to be noted that the group however read through online resources such as the MySQL official documentation [1] to discern minor particulars and subtleties needed to accomplish this task.

Apache Nifi Preparation (Task B)

The preparation of Apache Nifi includes the installation of the tool itself and the retrieval of data from the GOSales MySQL database to the data warehouse. With regard to the installation of Apache Nifi, some members needed to perform the task differently due to being in a Mac environment. On the other hand, in terms of importing the GOSales data to the warehouse, the group needed to transfer the data from the provided CSV file to another MySQL database as every member of the group is unable to access the remote resource provided. After importing GOSales to a MySQL database, the group began developing the process to transfer this data to the warehouse itself.

The initial process is to retrieve the data from the MySQL database which is done through a `QueryDatabaseTable` processor. This processor generates and executes a SQL statement based on its configuration and returns the result in an Avro format. The result is then processed in a `PutDatabaseRecord` processor configured to receive data in Avro format and transform this into an SQL statement which will be executed. Both of these processors need a `DBCPConnectionPool` service to enable data communication between Nifi and the databases.

As the process flow is relatively simple, no major challenges are encountered in doing this task aside from grasping the different processors, services, and setting up their proper configurations. To facilitate the connection between the Nifi process and the MySQL database however, a database driver namely the JDBC driver is required by the `DBCPConnectionPool` service used by the processors and thus presented the need for an additional installation separate from the installation of Nifi itself.

CSV Data Sourcing (Task C)

This task involves retrieving data from a CSV file and loading it into the data warehouse using Apache Nifi. To fulfill this task, the dataset is first downloaded from an online repository of datasets [2]. Having a local copy of the dataset, the group then proceeded to design the process and dataflow to import the data from this file to the data warehouse using an article by ProjectPro [3] as a blueprint.

The first step identified to accomplish this data loading task is the retrieval of data from the source file which is made possible by the `GetFile` Processor. This processor allows the fetching of files in the specified directory. Following this processor, the `UpdateAttribute` processor is used to facilitate the addition, deletion, and updating of file attributes. This allowed

the group to add a `schema.name` property with the value of the name of the table in the warehouse where the data will be stored in. The data will then be processed in a `ConvertRecord` processor to format the data to JSON. The final processor used for this task is the `PutDatabaseRecord` processor which is responsible for interacting with the data warehouse. This processor also requires a `DBCPConnectionPool` service which handles the connection of the process to the warehouse. Similar to the previous task, the aforementioned service is also used and requires JDBC which is the driver application for MySQL.

The development of this workflow to source data from a file in CSV format to a data warehouse is met with several challenges; however, the most significant error the group faced in performing this task is data format handling. In particular, the existence of the index column without a header name in the source resulted in incorrect data format. As the group determined that this column is unnecessary due to the existence of another data attribute which uniquely identifies each record present in the data, the members agreed to consider this index column in the processes; however, it will be ignored and will not be included in the loading of data to the warehouse itself.

MongoDB Data Sourcing (Task D)

The steps involved in this task are setting up the database connection, defining the schema for the record reader, and transforming the data to match the tables in the data warehouse. The record reader uses a specific avro-formatted schema based directly on the data from MongoDB. After converting the record to a JSON file, it was processed with several Jolt transformations, converted to sql, then stored in the data warehouse.

The main challenge for the task is transforming the JSON file. It was due to the file having nested attributes and the complexity in following the structure of the Supplies schema. Flattening the data for `supplies_orders` was relatively easy. However, the other tables require more complex operations to complete.

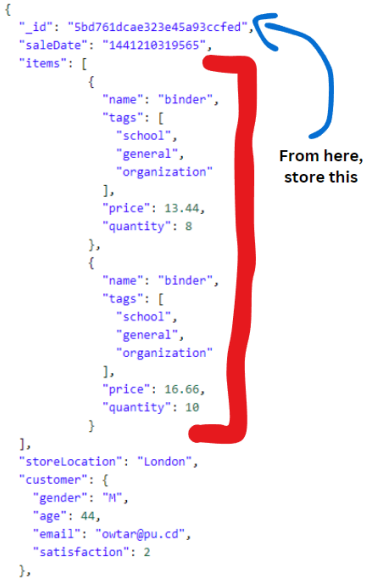

Figure 1. Sample record JSON file to be transformed.

One problem with xupplies_order_items and supplies_order_item_tags is separating array elements in ‘items’ and ‘tags’ to their own JSON record. Complicating matters further, we must also ensure that the orderid and supplies_order_items_key are stored accordingly in their record. As shown in figure 1, this was challenging as these fields are a level above their scope and I have zero knowledge of transforming data.

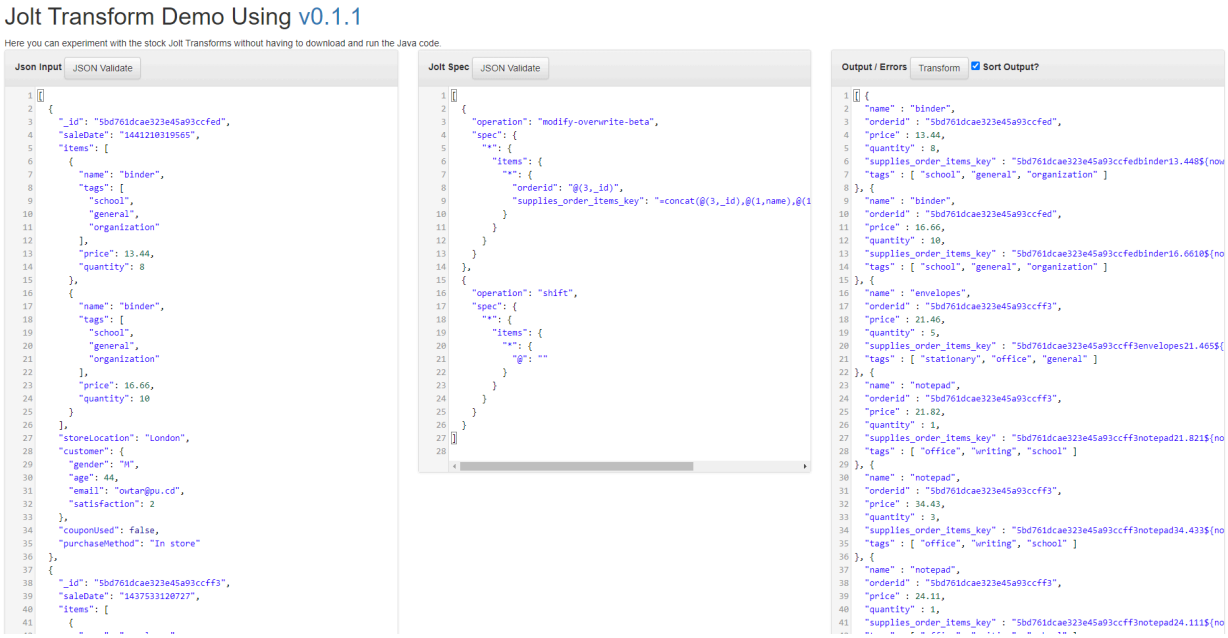

Figure 2. Successful demo of transforming records to supplies_order_items table format

The problem required an “until-5-am” trial and error and understanding jolt specification which was efficiently facilitated by a Jolt transform demo website [4].

First, it was important we manually specify the schema on the record reader to obtain a desired JSON format we can easily transform thereafter. For supplies_order_items, the solution was to add the orderid and a UUID to each item first with ‘modify-overwrite-beta’. Here, we specify several steps: “*”, traverse all records; “items”, go to the ‘items’ array; then, “*”, traverse all items object itself. The orderid was added using “@(3,_id)” to look up 3 levels where the _id value is stored, and a UUID was added with some string concatenation.

Afterwards, the ‘shift’ operation was used to flatten the data and extract each item object as their own record. ‘shift’ transforms the overall structure of the record, maintaining the values of the original JSON. “@”: “” was used to store all item data. Results are shown in Figure 2.

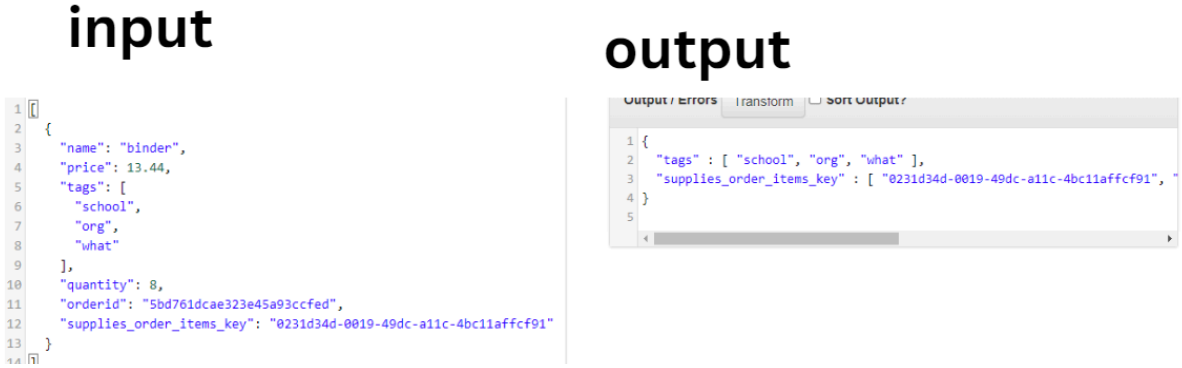

Figure 3. Jolt transformation using the structurally same spec from supplies_order_items

The same spec was also used for supplies_order_items_tags. However, since elements in ‘tags’ was not an object but instead strings, a huge problem was encountered where the record is transformed into arrays instead of records shown in figure 3.

An additional shift was required. The output arrays were traversed making a record for each index with "@": "[&].tagname", "@(2,supplies_order_items_key[#])": "[&].supplies_order_items_key". Inside the ‘tags’ scope, the first specification targets the current element being processed in the source array and stores it in the tagname field. The latter spec was required to go up 2 levels to supplies_order_items_key and get the elements from there.

Overall, Jolt Transformations, a powerful JSON manipulation library, excels in efficiently and elegantly formatting JSON data. Its strength lies in its ability to define concise, reusable transformation specifications that can be applied to complex JSON structures. By employing a declarative approach, Jolt simplifies the process of reshaping and restructuring JSON, making it particularly adept at handling diverse data sources and transforming them into desired formats.

Conclusion

When it comes to trying out tools for the first time, mistakes are expected to happen. Many minor errors were encountered throughout every step of this activity, and anyone else using these tools for the first time would be bound to make similar mistakes. Analyzing error messages, reading through the documentation of whatever tool is being used, as well as looking at what others have done are all instrumental to resolving these minor issues.

After experiencing first hand how much processing needed to be done in order to have the data be in a usable format, we can see how important it is to have this centralized clump of data. While we weren’t able to see how the data would be handled from that point on for business or analytical purposes, we could already see how the data warehousing aspect of the OLAP process that we were building would be vital for accessibility, consistency, and even efficiency.

Data warehousing can offer a medium for a streamlined process for any kind of operation on the data that simply wouldn’t be possible if the data was still in its heterogeneous form, coming from all different kinds of sources. It would be impractical to try to generate insights from data if different segments were in json and csv formats, for example.

References

[1] MySQL. “12.24.2 DECIMAL Data Type Characteristics.” [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/precision-math-decimal-characteristics.html

[2] Plotly Sample Datasets [Online]. Available: https://github.com/plotly/datasets/blob/master/26k-consumer-complaints.csv

[3] How to read data from local and store it into MySQL table in NiFi [Online]. Available: https://www.projectpro.io/recipes/read-data-from-local-and-store-it-into-mysql-table-nifi

[4] “Jolt transform demo.” https://jolt-demo.appspot.com/

Contribution

Name	Contribution
Corpuz, Joshua	Accomplished each task and contributed to writing the MongoDB sourcing section in the report.
Dimagiba, Rafael	Accomplished each task and contributed to writing the Conclusion section in the report.
Escopete, Steven Errol	Accomplished each task and contributed to writing the Data warehouse and Apache Nifi Preparation section in the report.
Pineda, Ralph	Accomplished each task and contributed to writing the CSV Data sourcing section in the report.