STADVDB H 01: Building a Data Warehouse with an ETL Tool

S11 & S12 Group 9: Jose Raphael Martinez, Jaeme Rebano, Wesly Samson, Miguel Villanueva

I. Tasks Performed, Challenges, and Resolutions

The tasks given in the activity explore ETL processes using various software tools such as Apache NiFi, MongoDB, and MySQL.

Task A requires the creation of a new schema for a local data warehouse. Initially, we established a database warehouse on our local machine and tested access to the GoSales database in MySQL. The required datasets for testing were located in the files tab on Canvas and were imported into a newly created schema in MySQL Workbench. During this task, no challenges were encountered, and the datasets imported from the CSV files underwent processing and cleaning. This involved removing tuples from the tables and adjusting the relationships and relational constraints within the tables. Notably, when the CSV files were initially imported into the workbench, the constraints were not immediately set. Upon closer analysis of the datasets, we identified attributes with relationships, enabling the establishment of primary and foreign keys for each table. We performed a reverse engineering of the scheme to create the Entity-Relationship (ERR) Diagram as a virtual representation of the current schema.

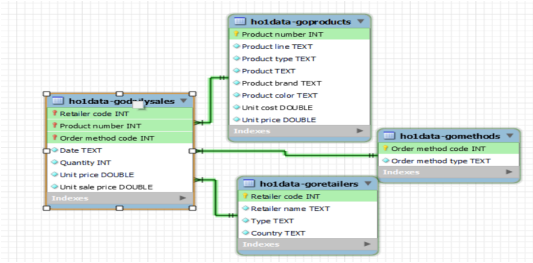


Figure 1. EER Diagram of the GoSales schema

Task B requires data transfer from a MySQL server onto a local data warehouse through Apache NiFi. The next step we took was to prepare the ETL Tool through the installation of NiFi. While installing the software, there was an error that we encountered during the installation of Apache NiFi. It was a JNI error in the installation of some members' local machines. While extracting the files of the zip file for the application, running the 'run-nifi.bat' file did not work due to the JDK being outdated with one of the machines. After updating the Java development kit and refreshing the environment variables, Apache was able to run and all the members were successfully connecting to the URL of Apache NiFi.

```
Microsoft Windows [Version 10.0.22631.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Miguel Villanueva>"C:\Users\Miguel Villanueva\Downloads\nifi-2.0.0-M1-bin\nifi-2.0.0-M1\bin\run-nifi.bat"
Error: A JNI error has occurred, please check your installation and try again
Exception in thread "main" java.lang.UnsupportedClassVersionError: org/apache/nifi/bootstrap/RunNiFi has been compiled b
y a more recent version of the Java Runtime (class file version 65.0), this version of the Java Runtime only recognizes
class file versions up to 52.0
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:756)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
        at java.net.URLClassLoader.defineClass(URLClassLoader.java:473)
        at java.net.URLClassLoader.access$100(URLClassLoader.java:74)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:369)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:363)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:362)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:418)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:352)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:351)
        at sun.launcher.LauncherHelper.checkAndLoadMain(LauncherHelper.java:621)
```

Figure 2. JNI Error while running run-nifi.bat

We also needed the MySQL Connector for NiFi to communicate with the server that was downloaded afterward-. A separate schema was created from the CSVs to act as our data source. QueryDatabaseTable and PutDatabaseRecord were used to transfer the contents from the source to the warehouse due to their simplicity. The column names had to be changed during the process as well since we realized that AVO does not like having whitespaces on it, and was adjusted accordingly. We originally encountered a problem that the data was being saved in the source instead of the warehouse but was quickly resolved by replacing the Database Connection Polling Service for PutDatabaseRecord.

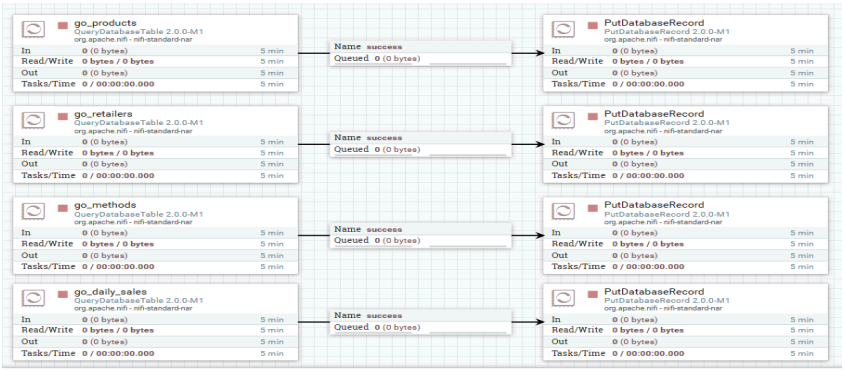


Figure 3. Successful Connection between the QueryDatabaseTable (source) and PutDatabaseRecord (destination)

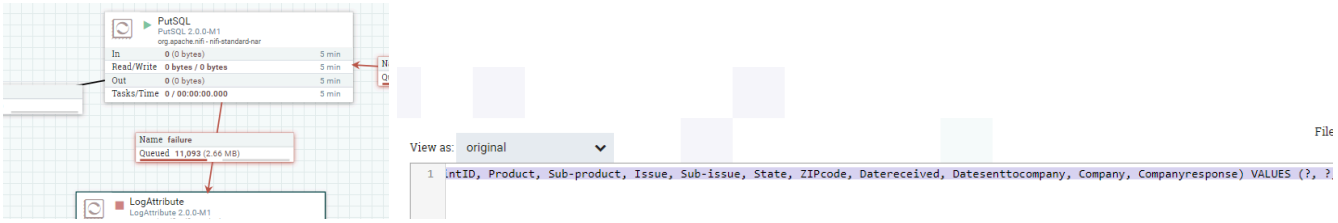
Task C involves creating a new table following the Customer Complaints dataset schema and loading the data into the data warehouse from a CSV file. The first step is to download the dataset from Github [https://github.com/plotly/datasets/blob/master/26k-consumer-complaints.csv]. After the source dataset was retrieved, we observed its attributes and data. Some attributes contained non-null values, leading us to set non-null attributes such as Complaint ID and Product when creating the table. The new table comprises of 14 attributes, including id, ComplaintID, Product, Subproduct, Issue, Subissue, State, ZIPcode, Datereceived, Datesenttocompany, Company, Timelyresponse, and Consumerdisputed.

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	☐	☑	☑	☐	☐	☐	☐	☐	
ComplaintID	INT	☑	☑	☐	☐	☐	☐	☐	☐	
Product	TEXT	☐	☑	☐	☐	☐	☐	☐	☐	
Subproduct	TEXT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Issue	TEXT	☐	☑	☐	☐	☐	☐	☐	☐	
Subissue	TEXT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
State	TEXT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
ZIPcode	DOUBLE	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Datereceived	DATE	☐	☑	☐	☐	☐	☐	☐	☐	
Datesenttocompany	DATE	☐	☑	☐	☐	☐	☐	☐	☐	
Company	TEXT	☐	☑	☐	☐	☐	☐	☐	☐	
Companyresponse	TEXT	☐	☑	☐	☐	☐	☐	☐	☐	
Timelyresponse	TEXT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Consumerdisputed	TEXT	☐	☐	☐	☐	☐	☐	☐	☐	NULL

Figure 4.1. Customer_complaints table

The next task is to create jobs in Nifi to load the data from the CSV file to the data warehouse. Several processors were used to accomplish the task based on a YouTube tutorial **[5]**. To retrieve the Customer complaints CSV file, the GetFile processor is added which created FlowFiles. We noticed that the first line of the CSV file contains attribute names with whitespaces, and the first column is empty. To solve the whitespaces, we used the search value and replacement value from a post in Stack Overflow **[6]**. Using the ReplaceText processor, we updated the content in the FlowFile to ensure the first cell contains 'id', and whitespaces are removed. Continuing with the processors, since the dataset is large, we thought of using the SplitText processor which divides the CSV file content into individual records for easier debugging when viewing the queue. Once the records are split, the ConvertRecord processor was added to transform the records into a JSON format, allowing us to specify the schema or structure of the data. Following this structuring, we included the ConvertJSONToSQL processor which generates the SQL INSERT statements based on the structured data from ConvertRecord. Finally, the PutSQL processor executes the generated SQL statements, inserting the data into the table in the data warehouse. Although, there is no connection to the server which is needed in the properties of PutSQL. We configured a **DBCPConnectionPool 2.0.0-M1** providing the properties such as the Database Connection URL, Database Driver Class Name, Database Driver Location(s), Database User and Password. There is no Driver that is installed, so MySQL Connector was downloaded and wrote the directory path in the configuration properties.

After running the processors individually, an error occurred towards the end: all the outputs from the *PutSQL* processor resulted in failure (Figure 4.2). After closely observing the entire flow and queues, it was identified that the insert statement after the *ConvertJSONToSQL* did not include the attributes *'Timelyresponse'* and *'Consumerdisputed'* (Figure 4.3) due to the question mark symbol being present at the end of it from the dataset (*Timelyresponse?*) making it not match the table attributes. To solve this error, another ReplaceText processor was added after removing the whitespaces which removed the question mark symbols. However, another error occurred stating that it could not distinguish the attributes after the hyphen in Sub-product meaning that it read 'Sub' only and the entire rest of the attributes as one ("-product,issue,sub-issue,...."). The ReplaceText processor was used to remove the hyphens. Once these processors were added, loading the data into the data warehouse was successful.



Figures 4.2 4.3. Failure results & Viewed file from list query with missing parameters

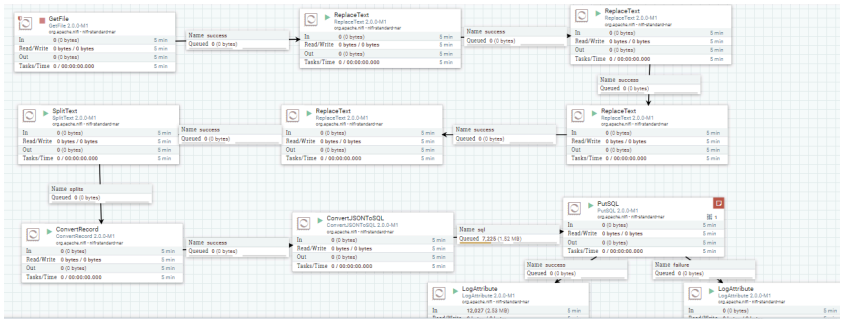


Figure 4.4. Flow Definition for loading data into Customer_complaints table

For **Task D**, the goal is to be able to insert data from a 'supplies' dataset in MongoDB into the MySQL data warehouse. In simple terms, the algorithm of the flow must be able to divide each original MongoDB document (with a total of 5000 documents) into 3 different JSON structures using different processors and services in Apache Nifi such that each JSON structure must correspond and be suitable for SQL insertion into a respective table in the data warehouse. With no prior knowledge of Apache Nifi, the first approach that I did was to try to establish a connection between Apache Nifi and MongoDB/MySQL. For MongoDB, I first tried to use the GetMongoRecord that made use of record writers to write the results. However, this was not needed, so I used the GetMongo processor instead and configured its **MongoDBControllerService** by providing the connection uri to the MongoDB database. For the MySQL connection, I first utilized the ExecuteSQL query and configured the **DBCPConnectionPool 2.0.0-M1** (which will be used throughout any SQL related processors) by providing the db connection URL, db driver class name (*com.mysql.cj.jdbc.Driver*), and the file location of the *mysql-connector-j-8.3.0* as these are crucial for establishing a connection to the MySQL database.

After ensuring that the proper connection to MongoDB and MySQL has been set, the next task, which was also the **most difficult**, is to determine the proper processors and relationships to be used for data extraction, reconstruction, and loading into the data warehouse. Since we need to transform the JSON document from the source database into a suitable structure for the data warehouse tables, I first dealt with the **Supplies_orders** table as it only requires a flat JSON. Furthermore, based on the requirements, the **JoltTransformJSON** processor was used because of its flexible *transformation* and *specification* feature for handling JSON transformations. For the first table, the following operation and specification was used based on a YouTube tutorial[3].

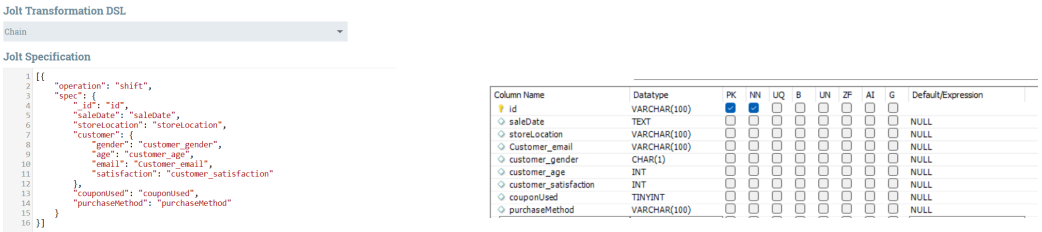


Figure 5.1 Sample Jolt Transformation Operation and Specification for Supplies_orders table

Figure 5.1 shows how the original JSON document from MongoDB can be transformed into a new JSON structure based on the given specification. It can also be seen that the data type of *salesDate* is in VARCHAR instead of *date time*. This was due to encounters involving unparseable date errors, so we agreed to change the data type to string since as of the time of this writing, we still haven't found a solution for the error. To put it simply, the specification trims the original JSON document by removing the items array key,, destructures the customer object,

and changes other key names with respect to the columns of the Supplies_orders table in the data warehouse. Afterwards, the resulting transformed JSON object is inputted into the **ConvertJSONtoSQL** processor, which also utilizes the DBCPConnectionPool service to establish a connection to the data warehouse and configures which schema and table will receive the incoming data. Next, the resulting SQL command from the ConvertJSONToSQL processor is inputted into the PutSQL Processor, which also utilizes the same connection pool service. The resulting output is also an SQL command, but signifies that the SQL query has been successfully executed if it goes through the success relationship.

After dealing with the appropriate JSON structures for the first table insertion, the remaining two tables were brought into focus. Both supplies_order_items and supplies_order_item_tags tables require deconstruction of nested objects For the *supplies_order_items* table, it requires flattening the **items** array from the original MongoDB documents so that each items with respective order IDs can be inserted into the data warehouse table. On the other hand, the *supplies_order_item_tags* requires a two-level deconstruction since the tags array is inside each element inside the items array. These requirements posed more issues later on because deconstruction/flattening of arrays/objects or transformation into JSON objects requires more complicated JOLTTransformJSON specifications and SplitJSON operations. In addition, the relationships of the data warehouse tables were also being taken into consideration, which also required more manipulations to the input JSON objects and its key values, such as converting each string element in the tags array into JSON objects so that they would be readable by other processors and would be able to retain key referencing information to different tables, assigning tag arrays into a key value for split convenience, etc. For a better visual of the flow for all the tables, refer to the following figure.

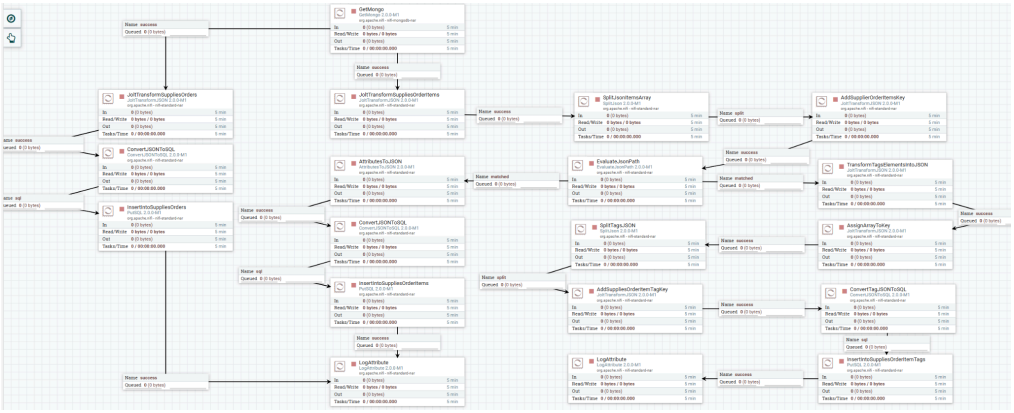


Figure 5.2 Whole Flow Definition for Transferring Data Into The Data Warehouse Tables

II. Additional Insights

Over the course of the task, we learned how to connect the database, and an important realization that we made while setting up the data warehouse and performing ETL is how powerful of a tool NiFi can be. From the formatting of the datasets and how different software can have different constraints regarding how a dataset should be fed, to creating processes that are automated, NiFi has a lot of capabilities that we haven't utilized yet. While performing the tasks, we also needed to make consistent adjustments to either the CSV directly or through the processor to handle problems such as the existence of whitespaces. Regarding the processors, there are several of them, which could also include pre-processing techniques that were required for us to use in order to successfully load the data into the warehouse. During this portion, we realized it was important to individually check in order the different processes in the flow. Looking at the logs to view the current activity allows us to know what went wrong if problems arise. Lastly, another insight to consider is that it is essential to understand your data and databases very well. Having proper understanding of how your data and databases are formatted, structured, processed, transformed, or sent not only reduces the amount of time needed to make trial-and-error efforts and changes when executing the flow processors and services, but it can also aid you in making efficient flow definitions or algorithms for better database performance and management.

III. Declarations

A. References

[1] EdbE, 2018, Is it possible to remove white spaces from the CSV files header name in NiFi?, https://stackoverflow.com/questions/52428510/is-it-possible-to-remove-white-spaces-from-the-csv-files-header-name-in-nifi

[2] Koon, S., 2020, Custom Insert Statement with PutSQL to Create mySql Upsert | BestBuy Dataflow | Apache Nifi | Part 7, https://www.youtube.com/watch?v=nO6eu9rmuUY&t=488s

[3] Koon, S., 2020, JOLT JSON Example Usage | Apache Nifi | Part 1, https://www.youtube.com/watch?v=85ECb9yEMwc

[4] Koon, S., 2021, MySql Database Connection Pool(DBCP) Service Setup | Apache Nifi, https://www.youtube.com/watch?v=0YROsMuqpFo

[5] Koon, S., 2020, Nifi - Ingest text file into flowfiles and put them into a mySQL table, https://www.youtube.com/watch?v=8KxcAiNdqvw&t=27s

[6] wcbdata, 2019, Jolt quick reference for Nifi Jolt Processors, https://community.cloudera.com/t5/Community-Articles/Jolt-quick-reference-for-Nifi-Jolt-Processors/ta-p/244350

[7] Koon, S., 2020, Optimized ETL's with QueryDatabaseTable and PutDatabaseRecord | Apache Nifi | Part 10, https://www.youtube.com/watch?v=9X8DJGXMra4

B. Contribution

Tasks were evenly distributed among team members, with each member assigned to one of the four parts. Specifically, Villanueva was assigned to Part A, Martinez to Part B, Rebano to Part C, and Samson to Part D. All members actively contributed to the writing of the activity report.