De La Salle University – Manila
Term 2, A.Y. 2023-2024

In partial fulfillment of
the requirements for the course
Advanced Database Systems

H01. Building a Data Warehouse with an ETL Tool

Submitted by:

Co, Shuan Noel S.

Dela Cruz, Sebastien Michael V.

Gregorio, John Marc A.

Ramos, Ronn Patrick B.

Section:
STADVDB - S13

Submitted to:

Professor Valdez Eduardo

January 24, 2024

In this paper we will discuss our procedure of building a data warehouse which involves several crucial steps, including preparing the environment, setting up the system, initializing the data warehouse, scripting the ETL data modeling processes, and implementing quality assurance to determine if the automated process works as intended when querying the data warehouse. The overall workflow encompasses setting up Apache NiFi, gathering the required data, overcoming setbacks, and ensuring the reliability of automated processes.

The initial step involved setting up the necessary dependencies to run Apache NiFi locally. Initially, the team missed this important step which resulted in various errors during installation, including the JAVA_HOME environment variable not being found and LinkageError that prevented the web app from running locally. To fix these issues, the team had to install the most recent version Java Runtime Environment (JRE), as well as Java Database Connectivity (JDBC) driver class to allow Apache NiFi to access the data warehouse built on MySQL. During the installation process, some members found out that the Java version must match the Apache NiFi version, with the latest version only accepting Java 21. Properly configuring the environment enabled the team to address complications that appeared later on, such as failures caused by the incorrect installation of the Java driver class when parsing transformed data to the warehouse.

After the completion of the setup, the team configured and prepared the automated scripts with Apache NiFi. With its help, the entire process of inputting, transforming, and loading data into the warehouse was streamlined. This efficiency arises from the ability to abstract away intricate coding details and focus on the conceptual processes required for transforming raw data. Once Apache NiFi was launched, the ETL process can be performed through a locally hosted web server at *https://localhost:8443/nifi*. To access this website, you need to retrieve the unique login credentials generated by the batch execution script which you can find within the logs folder. We have also found that the generated username and password will also be shown when you first launch apache-nifi. Before delving into the application of ETL Data Modeling, let's provide an overview of the concept and usage of Apache NiFi for ETL.

Moving onwards, the next procedure involves building the data warehouse. To meet our specific needs, we utilized MySQL Workbench and configured the necessary services and established security credentials for the root administrator. As admins, we created new schemas that amalgamated different database systems into tables. Through the help of the workbench, the majority of the workflow did not consist of any coding. Furthermore, the team did not encounter any difficulties in preparing the warehouse, schema, and individual tables that translated as to how the raw dataset files were accessed. For example, with the .CSV formatting, we transformed column headers and converted it into the corresponding attributes in the tables while modifying the naming convention to exclude spaces and special characters. Throughout the process, the team made small errors, mistakenly assigning float column values as integers in the database. Though a simple mistake on the part of the team, one can really see how proper preparation of the data warehouse and its configurations can really cause major setbacks due to incompatibility as the data flows from the ETL tool towards the data warehouse. In our case we had a relatively small database with only it being hosted locally, but imagine the nightmare of the data flow being interrupted by an outage due to a simple error of incompatible data types as a result may cause Data ingestion bottleneck, loss of data and a costly endeavor to fix as data is extremely precious.

Another potential challenge when utilizing ETL occurred during the inclusion of the transformation process. Uploading a file directly into the database is straightforward, requiring fewer steps. However, the introduction of the transformation step adds another layer of complexity as it requires you to maintain the data integrity throughout the transformation process. The manipulation and adaptation of data to fit the target schema can introduce errors and inconsistencies due to the fundamental differences between file types. Naturally, validation and error-checking becomes essential to ensure that the data remains unchanged during the transformation process.

Now that the data warehouse is prepared, the next step is to migrate the different types of data from various sources to the warehouse. This process can be executed using Apache NiFi, a Java-based ETL tool. The fundamental concept of ETL data modeling is simple: we initially extract the necessary data from different sources, which could include the local directory or the cloud. This is followed by simple or complicated transformations so that each data source conforms to the warehouse's standard to create consistency. Afterwards, the transformed data is loaded into the specified warehouse, which would now be compatible with the warehouse's schema and tables. Essentially, the goal is to abstract the tedious process and establish consistency and standardization processes. This approach proves beneficial for business intelligence, as individuals needing this information often lack the time to process, merge, and query the data themselves. As software engineers, our goal is to abstract the tedious process and streamline them for efficiency.

The team decided to work on the .CSV data sources first, namely GoSales and Customer Complaints, as this file type was the most common among the raw data. To start off, we established a connection between ApacheNiFi and MySQL server. This involved creating a process group capable of accommodating multiple controller services. For our needs, we employed a DBCPConnectionPool to link the hosted MYSQL Server to ApacheNiFi as these services required several parameters, including the database connection URL, JDBC driver, the driver installation location, and the unique login credentials. The only problem we encountered regarding the connection setup was how to configure the driver and determine the correct directory, which led to corresponding errors. Nevertheless, the team resolved these issues by having the appropriate driver class name (com.mysql.jdbc.Driver) for the specific driver version we had utilized. Following that, we decided to import the .JSON file and utilized MongoDB for this task. To accomplish this, we established a local MongoDB server and incorporated the MongoURI as a parameter in the controller services to connect it to ApacheNiFi.

Finally, the team now possesses all the necessary tools for implementing ETL. Our initial focus was on the GoSales dataset, which consists of 4 CSV files. We configured the GetFile processor to retrieve these datasets from a specified directory. Afterwards, we monitored its ability to read raw data by observing the file disappearance and the information displayed by the processor during the extraction process. Since it was working, we proceeded with the transformation phase. This phase holds utmost importance as the project's goal is to extract data from various sources into the warehouse. Therefore, we tried to keep the transformation simple by simply formatting the column headers to match the schema of our databases. For example "Retailer code" was transformed into "*Retailercode*" as spaces in attribute names in MySQL did not work when we were doing tests. For this case, we utilized a replace text processor which uses regex for selection values and replacement values, with our selection values targeting only column headers as attributes and replacement values varying depending on each data source. For some .CSV files, we only needed a single replacement text processor in removing spaces. However, some contained special characters which we opted to remove to remain consistent. Next, we created a connection between the GetFile process and ReplaceText processor to transfer read data from one processor to another. Afterwards, we use a PutFile processor to test if the intended transformations on the column header matched the table attributes from the schema. The PutFile simply returned the file with the changes made to a directory specified in the configuration panel. While implementing the process, we encountered two problems. For the first one, the file did not appear in the specified directory after running the processors. This meant that there was an error in the configuration or the replacement process. The second case was when the file appeared in the corresponding directory but either the changes were incorrect or there were no changes at all, which implied the regex statement was incorrect. We fixed the regex statement and removed the PutFile processors, as it was only temporarily added for testing purposes. However, we shall be using this again later on.

The following step was the most tedious and time-intensive aspect of the project–loading the data into the warehouse. Setting up the PutDatabaseRecord processor involves several steps. Firstly, we need to prepare a new service, specifically a CSVReader, as the data is in CSV format. Within this service, we have to set the properties to interpret the header as string fields for the attributes in our table schema. The next steps depend on the delimiters that separate each value. If it follows the standard CSV format where it is separated by commas, then we simply have to select the Microsoft Excel as CSV format. However, that is not the case in different data sources as some will have different standards in the way they record their data. This is where the team got stumped as we did not know how to deal with semicolon delimited CSVs so we played a bit with the configuration and selected the CSV format. We discovered the option to set a custom value for the separator and specify it as a semicolon (;), and similarly select true to treat the first line as header. We would have two different CSVReaders for different PutDatabaseRecords, one where it was comma separated while the other was semicolon separated. Going back to the PutDatabase record there will be a drop down menu when selecting a record reader we simply selected the one corresponding to the data source we were processing. Naturally the database type would be MySQL, the type of operation to be is insert, utilize the DBCP connection pool we had previously prepared, set schema name as the name of our database warehouse and table name as the table we want to pool that specific data source into. Lastly, the unmatched column behavior property was set to either fail or ignore depending on the behavior of the transformed data, as one of the data sources contains a column without a name that is observed to be simply a counter which was not needed in the loading of data into MySQL.

Next, the team integrated MongoDB and used additional processors to handle JSON data. The entire workflow is very similar to the previous process. We begin with the GetFile processor to acquire the JSON format. Afterwards, we performed the similar procedure to ensure that the JSON data containing information on the sale transactions are actually being read. Afterwards, we used the EvaluateJsonPath processor to extract specific fields from the JSON file. We decided to keep the customer and item attribute as a nested attribute as it felt more natural and intuitive. It also led to fewer properties in the EvaluateJsonPath configuration. Next, we utilized the SplitJson processor to break down the JSON content into individual records, aligning each record according to the supplies schema. We used the Attribute ToJson and the UpdateAttribute processors for converting and manipulating the attribute data. These processors can ensure that the attribute data is properly formatted in JSON while also providing the flexibility to perform necessary transformations before the data is loaded into MongoDB. One challenge we encountered was obtaining and managing the unique identifiers, such as the ID field, for the supplies_orderItems schema. The extraction and preservation of the unique identifiers for each record became complicated, especially with the nested attributes. We attempted to utilize the UpdateAttribute processor to create attributes that represent unique identifiers based on specific conditions or existing data. It allowed for the dynamic creation and adjustment of the identifiers during the transformation phase which helped maintain the consistency and distinct identifiers across the schemas. Some parts of the difficulty came from data type mismatches as well, but this was more of an issue with validation than anything else. Once the proper fields have been extracted from the JSON file and we were able to break down the JSON content into individual records, putting it into the database using ConvertJSONtoSQL and PutSQL was then just a matter of making sure that we did not make mistakes regarding data types/names.

Assuming everything worked, if we attempt to query the warehouse then we should be able to find the corresponding data to be in their corresponding attributes. However, we encountered many failures in terms of migrating the data for some of the data sources. The biggest issue is for the failure of the processors where it does not prompt what the error was nor if it failed. However, we figured out a way to notify us of failure or success. This is where PutFile comes into play as it shows the failure or success status of the processor. We used two PutFile Processors and connected it to the PutDatabaseRecord. One to indicate failure and the other for success. It also makes sure to terminate the processor for failure. This is because every connection in a processor has a prompt that shows a queue and the name of the connection, so we can track if there is a queue in either failure or success. There was an instance where we kept failing after repeatedly checking the configurations and making sure it was working. Apparently, the problem was the warehouse having conflicting data types. We mistakenly set the wrong data type while the other attribute names had typos. This was very eye opening to the team as sometimes we should not tunnel vision on specific parts of the project, but rather consider all possibilities. We wasted a lot of time thinking our approach was wrong, but in reality it was already correct and the error was located in another part of the project we had already considered to be complete. With that being said, after multiple trial and errors, testing and making each group of processors work, we attempted to run it all at the same time. Note that each GetFile has a separate group of processors so we can decide whether to run it all at the same time or one by one. After the long hours of configuration of PutDatabaseRecord, we queried all of the tables and it contained the data from various data sources. Apache NiFi can run indefinitely given circumstances so we can automate any changes or pushes of files to a directory, showcasing real time changes. However, this can be configured as well based on the needs such as for larger data sources real time may require lots of processing power and time causing bottlenecks therefore a daily update can also work.

Overall, the experience of using Apache NiFi and low-code tools to create conceptual processes to work was really fun and challenging. The concept of ETL makes more sense as companies scale into larger products and industries. For example, Google has multiple products, including Google Drive, Gmail, Youtube and many more, that work independently most of the time. Nevertheless, streamlining the process becomes crucial if they aim to have extensive data analytics and automate the combination and processing of millions of data seamlessly. Data comes in various forms and by extracting it from different sources, we use the ETL data model to extract different sources of data, apply transformation techniques as needed to be able to standardize and make sure it is compatible once it reaches the warehouse for Business Intelligence. ETL tools help us automate the process and save the time of manually doing a process that is repetitive and as the scope of the company increases data flow also increases considerably and to be able to scale data analysis properly automation will always be faster than letting a person manually work on it.

Contribution:
CO, SHUAN NOEL SANGALANG - Performed tests A-D and paper writing

DELA CRUZ, SEBASTIEN MICHAEL VISTAL - Performed tests A-D and paper writing

GREGORIO, JOHN MARC AQUINO - Performed tests A-D and paper writing

RAMOS, RONN PATRICK BICERA - Performed tests A-D and paper writing

References:
https://stackoverflow.com/questions/15171622/mongoimport-of-json-file
https://stackoverflow.com/questions/65120452/convert-json-to-sql-and-insert-data-in-db2-using-nifi
https://www.youtube.com/watch?v=v2ZLu_ykBr0&t=207s