Implementation and verification of a data vault

Storing and Managing Data(Module 06-32245),Autumn 2022

Wandiwash Krishnakumar, Arjunkumar , 2457663
axw263@student.bham.ac.uk

November 29,2022

1 ABSTRACT

Aim-Creation and Implementation of a data vault system with complete Staging layer, Enterprise layer and Information layer for 2 neuroimaging datasets acquired using fNIRS namely Visuomotor dataset and Pre-autism dataset. Background- We are provided with a helpful ER diagram with all the table's information and field names along with their datatypes. ***Methods*** -Instructions and demonstration on creating a database, tables followed by loading the datasets after completing necessary transformations on them is shown. Staging layer shows creating a database, schema and 27 tables according to ER diagram on project statement. The python side of staging layer integrates with staging SQL through PostgreSQL for automation purposes in project. Python reads the datasets, performs transformations and loads them into the tables and give verification on each table insertions. ***Results***-The end result would be all 27 tables loaded with appropriate data and ability to query them for further studies. Automated completion of the data vault is achieved by having python, SQL and datasets integrations while on the same directory. A full implementation of data vault system is possible with high degree of automation and ease of use for further querying on the loaded data for future uses, here we have tried to implement one such instance.

2 INTRODUCTION

Unprecedented large data volumes are generated by advanced observatory instruments and demand efficient technology for science harvesting [1]. Here we have such a scenario Where we have 2 datasets of completely different formats and types requiring different transformations. First, we have Visuomotor dataset where there are 75 comma separated files. Here, the file follows the format of having the metadata in the first K columns followed by having the data across 2801 rows and 30 columns. There are seven unique subjects in dataset 1 of which, 6 undergo 12 different groups of studies and 1 of which undergoes 3 different group of study. The metadata provides personal information on the subject's Name, Age , Sex and ID , It also gives the technical information of the study such as Sampling period, Wave[nm], StimType etc. Secondly, we have Preautism dataset, here we have 20 different groups which have 20 files for metadata of extension (.hdr) and each of this metadata is further accompanied by 3 other data files of varying file formats. The challenge includes reading the metadata files, identifying proper separators between field names and values and transforming them for loading into the tables. Total files taken into consideration here are 80 files. 20 files of the extension (.evt) are not taken into consideration as they launch an application on opening which is presumed out of scope in this scenario.

2.1 SQL AND POSTGRESQL

Creation of a database named smdvault, schema and tables inside the database is completed. A potential automation opportunity was noticed here, when running the python script multiple times for loading data, we will need to drop the tables or the entire database depending on the scenario for reloading of datasets. Thus, the PostgreSQL application was integrated with python script using psycopg2 module to drop database and recreate them each

time on new build for smooth reloads. The SQL script includes the commands for database creation, schema creation and table creation.

2.2 Python Script

Seven modules in total were used for this implementation, which are numpy, pandas, os, glob, datetime and itertools. The numpy module was used for the numpy arrays which help in array operations used in data transformation and loading. Pandas was used for reading the files into a dataframe using the built in method pandas.read_csv, the dataframes are an excellent tool for easy data transformations. OS and glob methods were used in combinations for paths and filenames retrieval to avoid file path hard coding issues. Datetime was used for date and time operations for various fields and finally itertools module was used for the zipping of lists to make them run in conjecture and iterate through them in parallel. Script is roughly split into three main sections, Firstly we read the files into a list, dataframe or any format based on method used, this is achieved by fetching all the filenames and paths and reading them using python inbuilt methods. Secondly we convert the read files into usable format, *For example:* we change the list containing each lines of a file as an entry to a list of dictionaries where each dictionary belongs to one file and the keys are the field names and values are the field values. This change allows for ease of use later down the line. Finally , all the converted files are loaded into the SQL tables using insert statements from python script itself, this is done with the help of cursor and connection objects to PostgreSQL and using string formatting for value substitutions.

3 STATE OF THE ART DATA VAULTS

The data vault keeps the data in its original format and place, while at the same time enables transparent (meta)data access and analysis using a query language.[2]Traditional data vaults are the ones revolving around fetching and loading of the data to facilitate for further business analytic and data science operations. There are innovations on this landscape with newer technologies building upon the existing methods and allowing for further data warehousing, data science and analytics operations at the centre of it with increased security to boast. Some latest innovations include SciDB where, SciDB is a new open-source data management system intended primarily for use in application domains that involve very large (petabyte) scale array data; for example, scientific applications such as astronomy, remote sensing and climate modeling [3]

4 METHODS

Filenames reading functions- These functions are using the 'os' method to fetch the current working directory's path and then use the 'glop' method to fetch the names of all the files matching the criteria. These filenames are all stored in a list and returned when these functions are called. Example method-'filename()'

4.1 DATA READING

These functions take the filename function's output as one of their arguments and traverse through the filename list, opening each one of those files and storing their data into another list or dataframe. They create one list or dataframe per file and will return a list of lists or list of dataframe as their function call return. This list is still in its raw format and it's still unusable. Example method-'metadatalist(),dataframelist()'

4.2 DATA TRANSFORMING

These functions take the data reading function's output and performs the transformation logics on them. In Visual Motor files , they would read the file output line by line and take only the metadata part and store them into a dictionary with the key being the field name and value being the field value, Similarly for the data part of the Visual motor file, it would take the data part alone and store them into a list. For Preautism files, it looks for the '=' as separator and takes all the metadata values from the '.hdr' files and for the data part , it looks at the four files per subject and stores them into a list for all 20 subjects.

4.3 DATA LOADING

These are the bulk of this project's python script. There are a total of 27 loading functions, one per table based on the ER diagram. Each of this function is unique based on the individual table's specific requirements. Some similar functionality among them is the use of the hash method for the creation of the primary keys. The source field and timestamp field are also generated in a similar way across the board. Some challenges faced were when one table is taking the primary key of another table as its foreign key. In this scenario we will need to take on of the two approach, either create the primary key in a reproducible way such that they can be locally generated inside the method or to append the primary key in a list and return that list as the function return. I have gone with the second method here. Another challenge was faced when one table takes the primary key of another table as its own primary key. In this case the number of records on both tables should match as the primary key cannot be duplicate thus, we need to understand the table's intent and the purpose behind the data to a higher degree to match the record counts. Once the decision is made on what to insert , then we would generate or fetch the required data and load them inside the table with appropriate datatype.

4.4 DECISIONS AND METHODS

For the assignedTo table, we observe it takes the primary keys form two tables as its foreign key, namely Hubgroup table and HubexperimentalUnit table. Here the Hubgroup table has a total of 12 groups and HubexperimentalUnit has 8 records, the decision was made to load the records into assignedTo table for only the units which were part of the groups. Another unique decision was made when in the ER diagram there is a clause stating HubSubject is a HubExperimentalUnit. This statement was interpreted to both tables having the same number of records.

4.5 CHALLENGES AND METHODS

In the sessionmetadata insertion method, The difficulty was in looping the foreign keys from two tables hubsession and hubmetadata in such a way that we get (number of metadata key value pair * number of files) records for both the VM and Preautism files. The insertion had to be done in parallel because of the NOT NULL constraint in the columns and this lead to the challenge and ultimately unique work arounds being used. This was a very educative scenario leading to new learnings.

5 RESULTS

The filenames were successfully read and loaded inside a list for further use, a snapshot is provided as a proof of it.

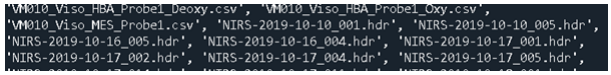


Figure 1

Data was successfully read and transformed, a snapshot is provided as a proof of it.

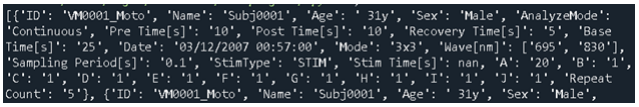


Figure 2

The data was successfully inserted into the tables, a snapshot is provided as a proof of it.

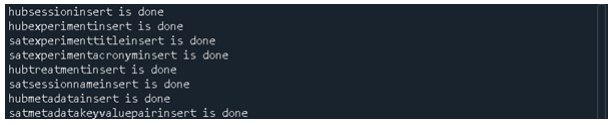
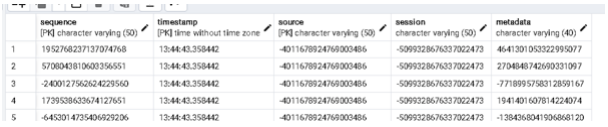


Figure 3

The data inserted using the scripts are being succesfully returned while querying from PostgreSQL



	sequence [PK] character varying (50)	timestamp [PK] time without time zone	source [PK] character varying (50)	session character varying (50)	metadata character varying (40)
1	1952768237137074768	13:44:43.358442	-4011678924769003486	-5099328676337022473	464130105332995077
2	5708043810600356551	13:44:43.358442	-4011678924769003486	-5099328676337022473	2704848742690331097
3	-2400127562624229560	13:44:43.358442	-4011678924769003486	-5099328676337022473	-7718995758312859167
4	1739538633674127651	13:44:43.358442	-4011678924769003486	-5099328676337022473	1941401607814224074
5	-6452014735406929206	13:44:43.358442	-4011678924769003486	-5099328676337022473	-1384368041906868120

Figure 4

The information layer queries were successfuly read and executed, the resulting records from tables were fetched and written to an excel file which was auto generated through the scripyt.

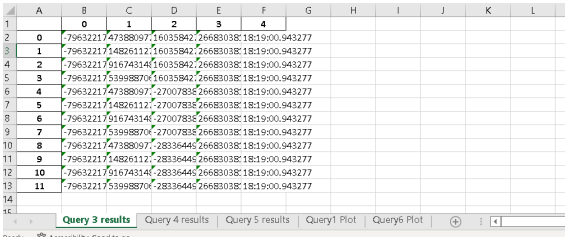


Figure 5

All these snapshots are being provided in the results subfolder along with the source code, sql files and any other supporting files necessary. A README.txt file with instructions on how to replicate the results on any machine is provided The code is devoid of any bugs and runs off the shelf without any modifications from th user.

6 DISCUSSION

Further improvements can be made on this model to allow for easier data insertions and accessibility. Currently , we have all the data inside a folder and we load them, but the facility to load new data as they arrive inside the database as an update would be better. APIs would be an excellent tool for this facility. We can allow for the new data to be inserted through a website as they are generated using APIs and it would allow for an up-to-date database where we don't need to bulk load the data each time. We can also consider JSONS as an option, the new data can be transformed into a JSON and loaded as part of batch loads. Furthermore, increased security features could be another improvement with access restrictions on who can edit the data and who can view them. Facility for iterative data loads without database drops and recreation each time seems to be the sensible next move with facility to keep a record of data load history.

7 CONCLUSION

A complete, efficient and state of the art implementation of the enterprise layer, staging layer and information mart was achieved which adheres faithfully to the project guidelines and the requirements with additional facilities and modifications added solely with the intent on improving the accessibility and sensibility of the model. The model achieves its original goal of having a complete database model with all the tables loaded with their necessary data in its entirety and being readable through SQL for further analytics. In conclusion ideally We aim to develop a data warehouse system with the ability to query relational and NoSQL data as well as to integrate, store and preserve history of all the corporate data and their changes into a single system of records [4-6]

8 REFERENCES

[1] Hey, T., Tansley, S., Tolle, K.: The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft Research (2009)

[2] M. Ivanova, M.L. Kersten and S. Manegold, "Data Vaults: A Symbiosis between Database Technology and Scientific File Repositories", Proc. 24th

[3] Overview of SciDB: Large Scale Array Storage Processing and Analysis", Proc. 39th Ann. Sigmod, pp. 963-968, 2010.

[4] V. Jovanović, I. Bojicic, C. Knowles and M. Pavlic, "PERSISTENT STAGING AREA MODELS FOR DATA WAREHOUSES", Issues Inf. Syst., vol. 13, no. 1, pp. 121-132, 2012.

[5] D. Jakšić, Metadata repository model for data warehouse schema evolution and integration with master data management system, Sveučilište u Rijeci, 2016.

[6] D. Subotic, "Data Warehouse Schema Evolution Perspectives", New Trends in Database and Information Systems II - Selected papers of the 18th East European Conference on Advances in Databases and Information Systems and Associated Satellite Events ADBIS 2014 Ohrid Macedonia September 7–10 2014 Proceedings, vol. 312, pp. 333-338, 1, 2014.