[[1]](#footnote-1)

When does it make sense to use a Data Lake instead of a Data Warehouse?

Cameron Stewart, Nathan Deinlein, Cleveland Johnson

Data Lakes is a relatively new data storage framework, that like most cloud native technologies, is in the process of being adopted by large enterprise organizations. There is not a lot of clarity within Technology organizations on how to best create and leverage these Data Lakes. A company’s decision to invest in this new technology is expensive and requires significant thought into initial processes to ensure integrity and quality.

*Index Terms*— Computational and artificial intelligence, Computers and information processing …Computer Applications Big data applications.

# Data lake vs data warehouse problem

A data lake, much like a data warehouse, is meant for specific use cases. The use cases for a data warehouse could be considered common knowledge within Enterprise Technology organizations. There are high level statements and understandings of the use cases for data, but not many have implementation experience from which to draw. The goal of this paper is to address two foundational questions. The first question is “What is the difference between the data warehouse and data lake?”. Once grounded in the differences, we will then seek to address “When does it make sense to use a data lake instead of a data warehouse?”.

# methodology

To answer our two foundational questions, have chosen to address each question separately.

Our first question, “What is the difference between the data warehouse and data lake?”, will be answered by standing up a data warehouse and a data lake to observe the operation of each. We are taking two concepts into account when establishing our technology environments for learning. The first concept is around structured and unstructured data. The second concept is taking big data into account.

The data warehouse is traditionally a structured data platform that is sourced from operational systems, aggregated and/or transformed into it a target state to provide analytic value. Another interesting way to think of a data warehouse is schema-on-write, meaning the schema must be defined prior to loading the data. Since the data warehouse is typically an aggregation of source systems, it is not generally used in Big Data use cases.

The data lake on the other hand can be defined as an unstructured data platform that is essentially performing and extract from a source and loading into a target table. This extract then load begs the question that when is the data aggregated and/or transformed? The transform occurs when a use is querying the data, also known as schema-on-read. The schema-on-read and unstructured data attributes makes the data lake an excellent candidate for Big Data.

With these ideas in mind, we chose to use a Relational Database Management System for our data warehouse, specifically choosing MySQL. For the data lake, we chose to use a NoSQL Database Management System, specifically MongoDB. Mongo DB also has an Atlas Data Lake1 feature that enables the creation of a data lake with very low out of pocket costs. This product can connect to multiple data sources including Amazon S3, MongoDB Atlas clusters and HTTP URLs. To easily flush out our use cases, we will connect to MongoDB collections.

To incorporate Big Data into the analysis, Twitter was chosen as the primary data source for both the data warehouse and data lake. Two data sets were collected from twitter to perform the analysis. The first data set is common across the data warehouse and data lake. The common data set was generated using python and the Twitter API filtering on the keyword #covid-19. The second data set, specific to the data lake, used python and the Twitter API filtering the keyword #DataLake.

Our second question, “When does it make sense to use a data lake instead of a data warehouse?”, will be answered by observations during data ingestion, performing a query against the data warehouse and data lake to see how they are the same and then perform an additional query on the data lake to showcase how it is different from a data warehouse.

# Data Warehouse observations

Summary:

This section details our data warehouse including the structure, data within and insights on how the data warehouse consumes the data and the usefulness of query results.

We will create the data warehouse using MySQL, test ingestion techniques and provide observations on optimization and reporting from the data warehouse.

Analysis:

For this part of the project, we build a MySQL data warehouse with a single table for collecting raw tweet data2. This has the following attributes, a unique, autogenerated TweetID, a Created\_At time stamp, the Tweet text itself, retweet count, and finally location. This was the best solution for trying to gather the tweets into one location and seemed the simplest structure for importing them. Normally in an RDBMS we would want to create a more complex and normalized schema with tables for things like location or a created by user id. However, there were issues with this that will be outlined later.

The first step in capturing our stream of Tweets was to build out a Jupyter notebook that would allow us to connect to both the Twitter community API as well as to our on-prem MySQL server. Once this was accomplished, we set the filter up to look at tweets specifically with the hashtag of Covid-19. Due to this particular hashtag’s prevalence, we quickly collected a sample group of about eighty Tweets. This grouping would be used for our testing and querying needs for the rest of the project.

Step two was to quickly analyze the data that came out of the API. The Jupyter notebook was set to parse the data from the tweet using Twitters internal header system. Almost immediately when looking at the data, it is clear that internal structure has issues with porting into the DBMS. Probably the clearest indicator of this is looking at the location column. Items like “Psalm 46:5” (Table 1) should not be parsing into the location attribute, but it is.

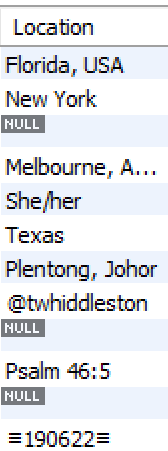


Table 1: Example of Parsed Data from the Location column

This is due to the semi structured nature of the data coming in. While this would not be a problem for a NoSQL database or a data lake, it is a limitation on the traditional RDBMS. This is also why we had to limit our schema structure to a single table. Because the columns other than Created\_At and Tweets cannot be relied on to import predictable and similar data types this makes it impossible to normalize the data. Ideally there would be a schema with user information, a tweet grab that could be joined by a foreign key as well as a location table. However, due to loading issues of the semi-structured data, this is not possible. This then presents the first and greatest problem of using the data warehouse for this type of work, no normalization, which means no efficiency creation and columns can return almost any data type making it hard to query.

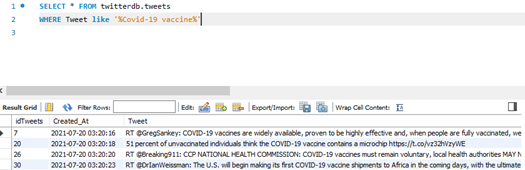
The final issue with moving this data into an RDBMS is in querying the data itself. While it is possible to query results to find certain keywords, each query requires multiple layers to find exactly what is being sought in the data (Figure 1). 

Figure 1:Example query to find Tweets pertaining to Covid-19 Vaccine

Also, the returned results are not necessarily the most helpful without exporting the data to a script capable of running Natural Language Processing. However, some metrics about the data can be gained. For example, a count of all Tweets containing information on the Covid-19 vaccine, is possible but noting whether it is a news piece, positive opinion, or negative opinion is not. This extremely limits the capacity of the RDBMS to provide useful insights from the data stream. The other part is because the location data is unable to be loaded properly, that limits querying and then visualizing the data by a location basis.

Overall, due to the limited retention capabilities of RDBMS to handle the data structure as well as the throughput of the data’s necessary write time, we do not believe that an RDBMS would or should be used for capture of this or similar data types. The infrastructure needs to attempt to parse the data into specific columns as well as the necessary for a 288 character field to hold the main body of the document itself makes the RDBMS unwieldy. This coupled with the parsing issues presented earlier greatly limit the RDBMS’s greatest strength in being able to normalize the data. This makes using SQL for the purposes of querying and analyzing the data problematic and overall unhelpful at best and at worst misleading and chaotic.

# Data Lake observations

Summary:

This section details our data lake including the structure, data within and insights on how the data lake consumes the data and the usefulness of the query results.

We will create the data lake using MongoDB, add two MongoDB collections, test ingestion techniques and provide observations on optimization and reporting from the data warehouse.

Analysis:

Prior to beginning the build for the data lake, we first wanted to establish the data sources that would be leveraged. This would be achieved by creating two MongoDB collections, one for #DataLake tweets and one for #Covid-19 tweets.

The #DataLake MongoDB collection need only have been created. Contrary to the data warehouse, no tables needed pre-defined. This proved beneficial when working together on distributed project team as the data warehouse could be developed in conjunction with the data and then the data lake data load was changed to match the MySQL data warehouse.

The #DataLake collection was populated by building a Jupyter notebook that connects to the Twitter community API and the MongoDB Collection3. The notebook leveraged the tweepy library to extract the tweets from a specific time period with a specifics hashtag, in this case #DataLake. Once the data is captured by the API, we loop through the list to extract the desired information (text, favourite count, retweet count, created at) and store it in a list. This list is then converted to a dataframe and uploaded into MongoDB. Each item in the list is a document within the collection.

The second collection, #Covid-19, was populated by importing a csv from our data warehouse into the collection. This is also referred to as the common data set to provide a like for like comparison during query analysis.

We did not encounter significant issues loading the data into the data lake. In fact, we were pleasantly surprised by the flexibility of the process. This feature would enable the users to quickly ingest and wrangle data.

Now that the two collections were created, the data lake wrapper, MongoDB Atlas Data Lake, was ready for implementation. The MongoDB Atlas Data Lake made it easy to combine the collections into a data lake. It was merely an action of dragging and dropping the collections to be used. (Figure 2)

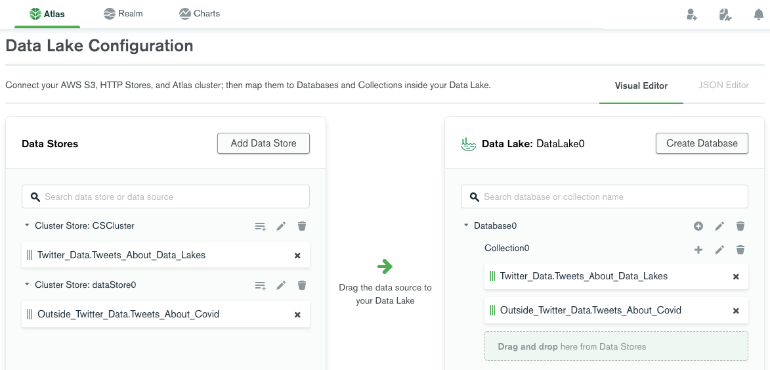


Figure 2: Adding data stores to data lake.

As mentioned previously, these data stores can be collection from multiple MongoDB databases or Amazon S3 buckets. In our case, we added the collections mentioned above. After completing the Data Lake configuration, we discovered that we can use the same methods to connect as a standard MongoDB database. In our case, we chose MongoDB Compass. Upon connecting to the data lake, we observed that two collections were combined into one collection to enable querying the whole data set. To test our ability to query across the entire data set, we performed a query to find documents with the string “data” and a retweet count of 0. One document was returned from each original data set. (Table 2)

|  |  |
| --- | --- |
| Query | db.Collection0.find({"retweet\_count":0,"text":/ data /}).sort({created\_at: -1}).pretty() |
| Result 1 | {\_id: ObjectId("60f9e95fffc0381c73f2c993"),  index: '35',  created\_at: 2021-07-20T10:20:00.000Z,  text: 'RT @SuicidePrevAU: The @aihw released updated **data** on suicide &amp; self-harm monitoring today. Despite increased calls to support lines &amp; ment \_',  retweet\_count: **0**,  Location: 'Canberra, Australia' } |
| Result 2 | { \_id: ObjectId("60e630a8b6b55c49032ae632"),  index: 8,  text: 'To efficiently ingest **data** into a **data** lake, your team must properly evaluate the environment and technology choices. NEOS shares its lessons learned. \nhttps://t.co/V474QDX1LP\n#datalake #database #insurance https://t.co/EGVAadTH1a',  favourite\_count: 1,  retweet\_count: **0**,  created\_at: 2021-07-06T20:04:03.000Z } |

Table 2: Data Lake Query Results

# Data warehouse vs data lake Analysis

With the completion of our observations of both the data warehouse and the data lake, we now set out to answer our first question, “What is the difference between a data warehouse and a data lake?” As discussed in our methodology, we set out to use two concepts to define differences between a data warehouse and a data lake. The first concept involved the structure of the data, while the second concept involved the notion of big data. When talking about big data.

We observed a stark contrast between the data warehouse and data lake in terms of the structure of the data. In the case of the data warehouse, we had to analyze initial data pulls from the Twitter API, discuss available fields, determine what question we wanted to answer with a query and then align on the table structure of the database. This made us realize how accurate the schema on write moniker was for a data warehouse. While for the data lake, we merely had to capture the desired variables from the Twitter API, load them into a data frame and load them into the database. Based on these observations, the unstructured data ability of the data lake enabled speed of discovery, ingestion, and integration with other data sets. The structured data of the data warehouse is powerful, but we realized that without normalization and indexing, the speed of response could not be achieved.

From a big data concept we learned a valuable lesson with the Twitter API feed for the data warehouse. We realized the structure of data from Twitter was not consistent. This caused some anomalies in the location field, seen in Table 1. These types of data issues would cause errors that would need to be handled when loading directly into a data warehouse, potentially preventing the entire entity from being loaded. This could also cause resource constraints on the support team to check and clear said issues. The data lake on the other hand, would load the data in its raw format, ignoring the data issue. This handling of this issue would be handled by a user when the data is picked up for an analysis project. While this may appear to shift the work for fixing the issue, it specifically puts the ability to fix the issue in the hands of the person that has the business knowledge to make an informed decision. The data lake is clearly made to handle large amounts of data when compared to the data warehouse.

# Conclusion

Now that we understand the differences between a data warehouse and a data lake, we now have the information to answer our second question, “When does it make sense to use a data lake instead of a data warehouse?”.

When to use a data warehouse and a data lake can easily be summarized in terms of the problems being solved by the system.

A data warehouse should be used where the solution is known for the problem and accessed routinely for operational decision. An example of this would be start of day reports for investment bankers. These users have a core set of known data that is aggregated from multiple systems and show historical information to prepare them for the start of the trading day.

A data lake should be used when we still do not know the solution for the problem or are improving an existing solution. This would require the discovery, ingestion, and wrangling of data to better understand the current state and present potential solutions for further analysis. This approach would benefit from speed and flexibility provided by the data lake data platform.

We have clearly seen that the data warehouse and data lake have distinct use cases based on the problem being solved. A good analogy to solidify these concepts comes from the MSDS7330 class slides from week 9. Think of a data warehouse vs data lake as “agriculture vs. hunting/gathering”.

REFERENCES

1. Creating a Data Lake in MongoDB: <https://docs.mongodb.com/datalake/>
2. Twitter API to MySQL: <https://github.com/nedeinlein/DBMFinal_Paper/blob/main/Twitter%20API%20to%20DB.ipynb>
3. Pull Twitter Data and push to MongoDB: https://github.com/nedeinlein/DBMFinal\_Paper/blob/main/Gather\_Tweets\_and\_Upload\_to\_MongoDB.ipynb

# Appendix

All code, data extracts and documents can be found on our GitHub site: <https://github.com/nedeinlein/DBMFinal_Paper>

1. [↑](#footnote-ref-1)