Data Warehouse

Design Document

Submitted by Group # 6

Greg Petsul - 36258747
Michael Sheroubi - 21218169
Ahmed Fayed - 87874376
Wang Tianhao - 61903167

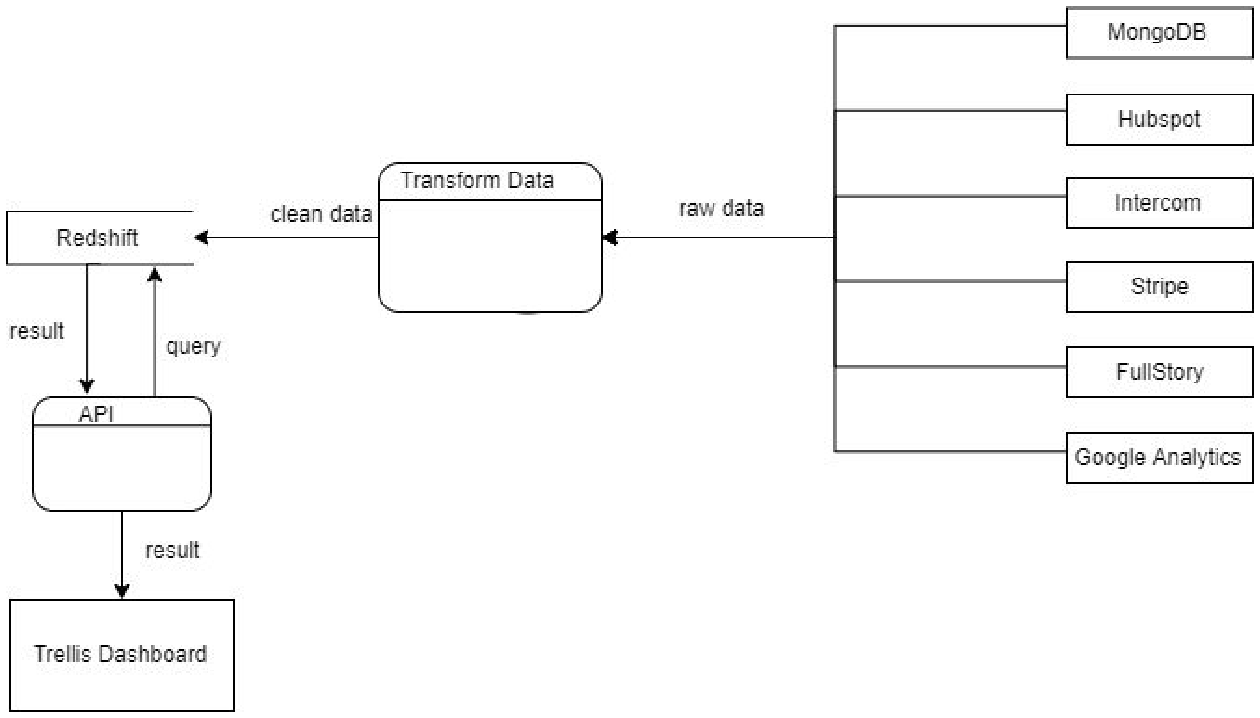
Table Of Contents

<u>Project Description</u>

This project is centred around building a Data Warehouse (DWH) to house checkouts and fundraiser related data for Trellis Social Enterprise Inc. This involves extracting Data from various data sources, storing the extracted data in a Data Warehouse designed by us, and creating an API to display analytics based on business use case questions required by our client, on their internal dashboard. We will be choosing which cloud DWH to implement, designing the DWH schema based on Trellis' various data sources and needs, finding and implementing an ETL tool to transfer the data from their sources into the DWH, and bringing it all together in an API so they can implement it in their dashboard.

<u>System Architecture</u>

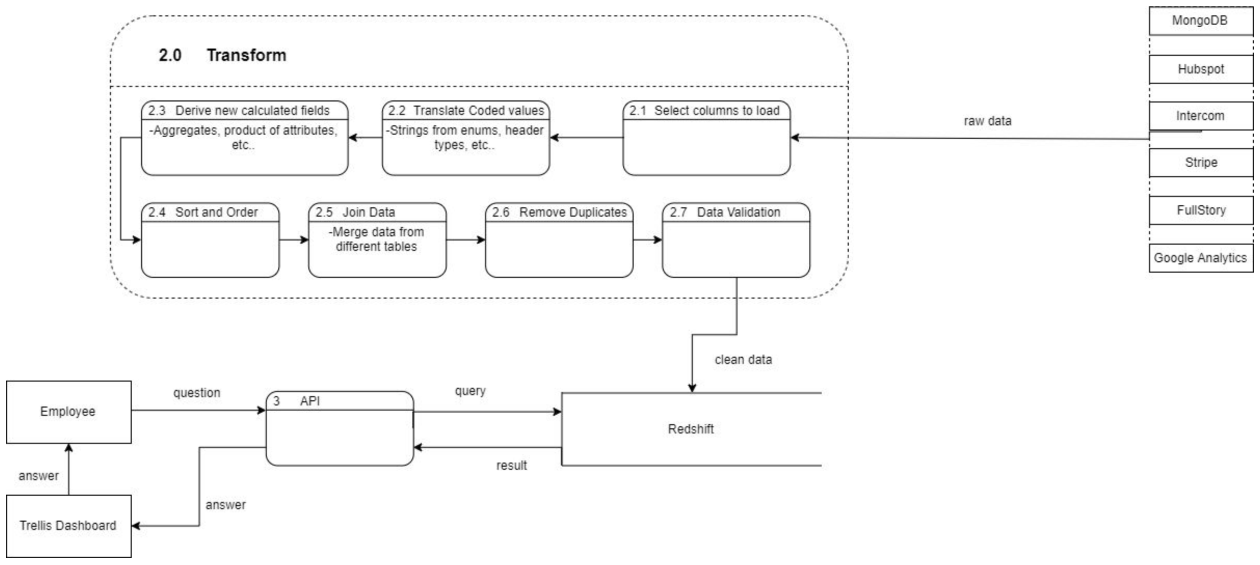
DFD Level **<u>0</u>**



Notes:

- The data-flow focuses on how the third-party data reaches the Trellis dashboard. This system does not interact with systems other than the dashboard and the listed third-party applications.

DFD Level **1**

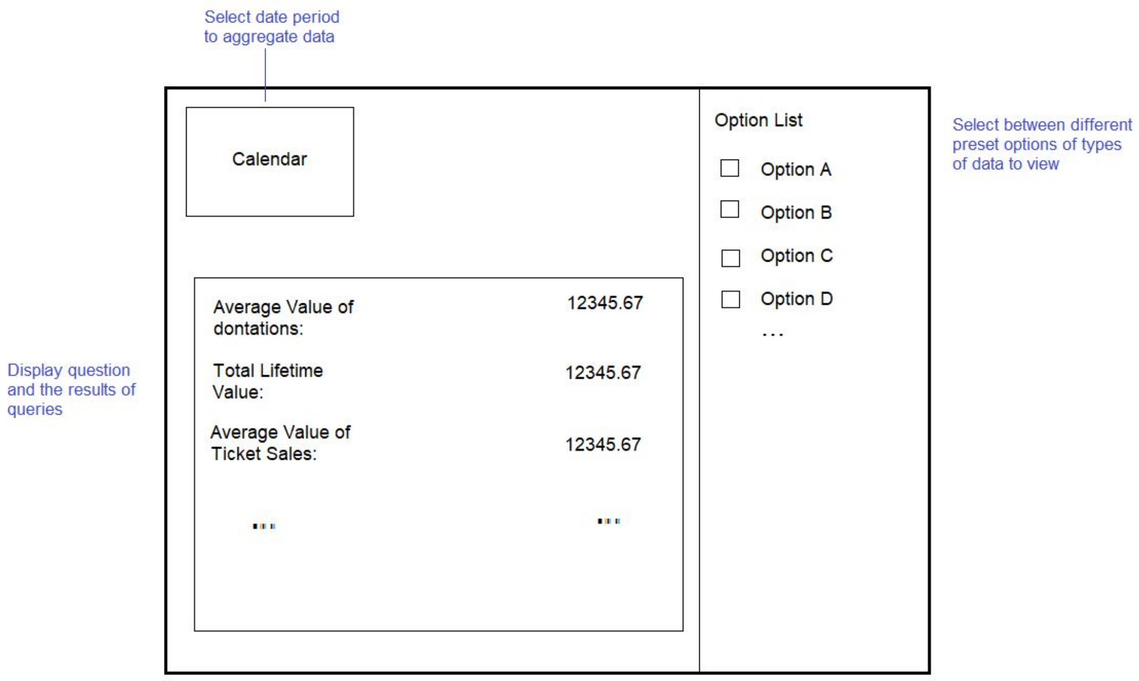


Notes:

- The API will allow the trellis dashboard to connect to the Redshift cluster hosting the data warehouse.

User-Interface

First Mockup


Select date period to aggregate data
Calendar
Option List
Option A
Option B
Option C
Option D
Select between different preset options of types of data to view
Display question and the results of queries
Average Value of dontations: 12345.67
Total Lifetime Value: 12345.67
Average Value of Ticket Sales: 12345.67

Client Feedback

- Select an Org
- Select a specific Fundraisers
- Select all fundraisers/ all of an orgs fundraisers
- Ticket/Donation/Item breakdowns
- Date range

Questions:

1. What are these preset options?
2. Do the options set what content will be display?

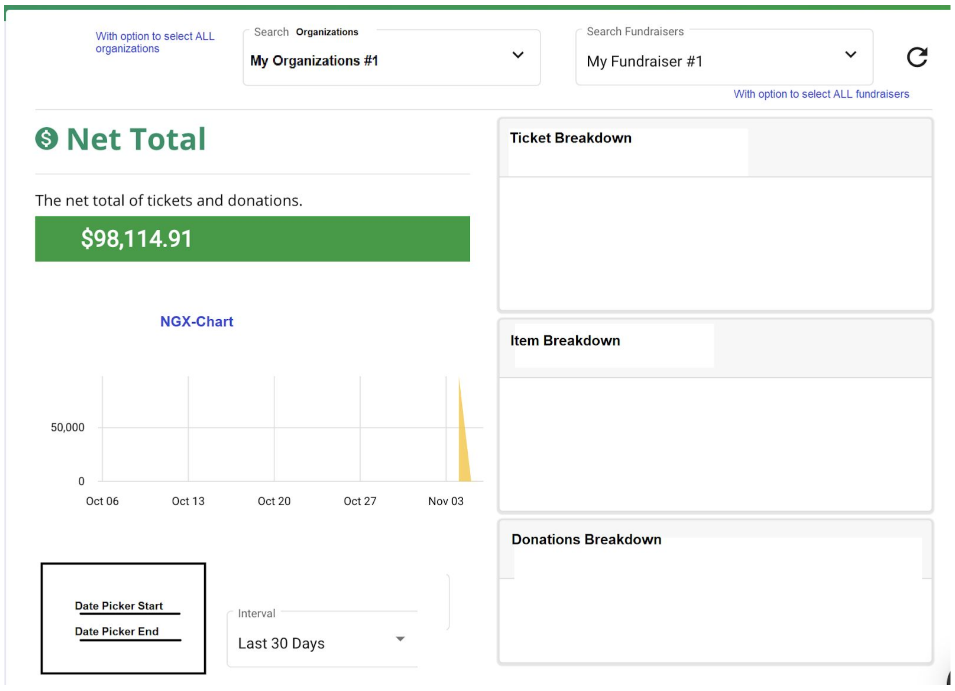
Should only be the answer to the questions that is asked

Use @swimlane/ngx-charts to chart the results.

Angular material or Bootstrap - date picker.

Second Mockup

Fundraiser View


With option to select ALL organizations

Search Organizations: My Organizations #1

Search Fundraisers: My Fundraiser #1

With option to select ALL fundraisers

Net Total

The net total of tickets and donations.

$98,114.91

NGX-Chart

Date Picker Start

Date Picker End

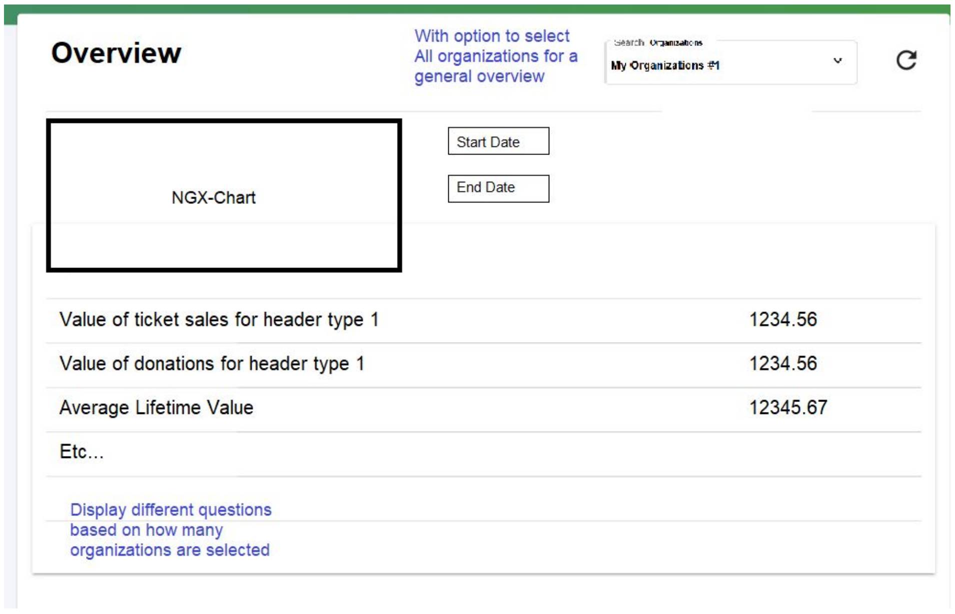
Interval: Last 30 Days

Ticket Breakdown

Item Breakdown

Donations Breakdown

Organization View


Overview

With option to select All organizations for a general overview

Search Organizations: My Organizations #1

NGX-Chart

Start Date

End Date

Value of ticket sales for header type 1	1234.56
Value of donations for header type 1	1234.56
Average Lifetime Value	12345.67
Etc…	

Display different questions based on how many organizations are selected

Technical Specifications

Amazon RedShift

Tellis has experience using AWS and has a credit to go towards amazon services, making RedShift as our Data Warehouse of choice. Keeping Trellis in a familiar ecosystem, having the reliability of Amazon's cloud services, and having a Data Warehouse that fulfills their needs while at a reasonable, pay-for-what-you-use, price point is why we chose RedShift over Data Warehouse service like Google BigQuery and Snowflake . BigQuery is a less cost effective option if the company does not plan to use it frequently and does not require many large queries. RedShift, on the other hand, has predictable hourly rate or an option to pay upfront for 1 or 3 years (at a deep discount) at a much lower and predictable rate than BigQuery can offer. There is a further breakdown of RedShift, Snowflake, BigQuery, as well as Microsoft Azure SQL Data Warehouse that we brought to our client to help make our decision which can be found in the pages following this section.

Stitch Data

To Extract the data from Trellis' various data sources and integrate it into Amazon RedShift, we chose to go with Stitch Data. Stitch Data is an ETL tool that will help us Extract, Transform, and load our client's data sources into our data warehouse. Stitch supports all of the data sources required by Trellis as well as RedShift. Informatica, another ETL tool, has less transparent pricing and requires a locked in annual contract. While another ETL tool, AWS Glue, has a less predictable, hourly, pricing and would require us to orchestrate Glue to work for our needs where Stitch Data supports what we need out of the box.

Typescript

Typescript was outlined by our client to be requirement for our project as it is what they are familiar with. Typescript is a strongly type language that, when compiled, gets translated into javascript, essentially making it a more familiar-feeling, object-oriented style language with some additional error checking over Javascript. Java is a language with similar benefits as Typescript has over Javascript. We are also familiar with Java, however, Typescript can be used for both back-end server side programming as well as having the benefit of Javascript for any front-end client side programming we need. Being able to use one language for our project that the developers at Trellis are used to using outweigh the benefits of Java.

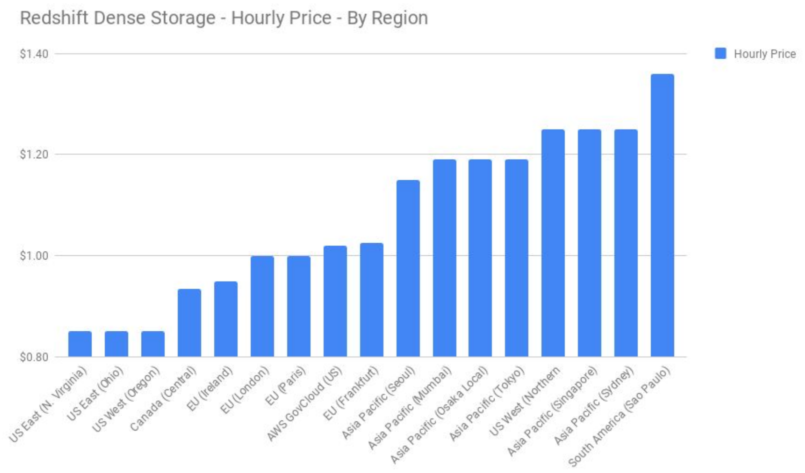
Nest.js

Nest.js was also a requirement from our client for our project. Nest.js is an angular styled wrapper for Node.js and Express.js which enables us to create server-side applications using typescript. Nest.js is an opinionated framework, that is, it will help guide the structure of our project, which should help our project be more maintainable. This is as opposed to an un-opinionated framework, like Express.js. This should hopefully allow us to build a more cohesive project, while still allowing us to use the features of Express.js. Nest.js implements the functionality of a typical Node.js framework, however, Nest.js introduces tools and structure to better organize our project over plain old Node.js. This should allow our code to be a bit more readable and maintainable when we hand our project off to Trellis.

Data Warehouse Solutions

Amazon Redshift

Redshift Pricing: https://aws.amazon.com/redshift/pricing/ (Also has a 2 month Trial)



Pros:

- It's fast; it is able to run multiple SSD nodes at once as well as running complex queries flawlessly.
- It's one of the cheapest compared to its competitors for what you're getting.
- Very Scalable; the more nodes you have the quicker it goes.
- Insane Storage Capacity; A normal setup has about a petabyte of data storage.
- User friendly SQL Interface that also works with JDBC/ODBC. (Basically, if you know SQL you'll have a great time with this)
- It's much more ahead of alternatives like Bigquery and Snowflake.

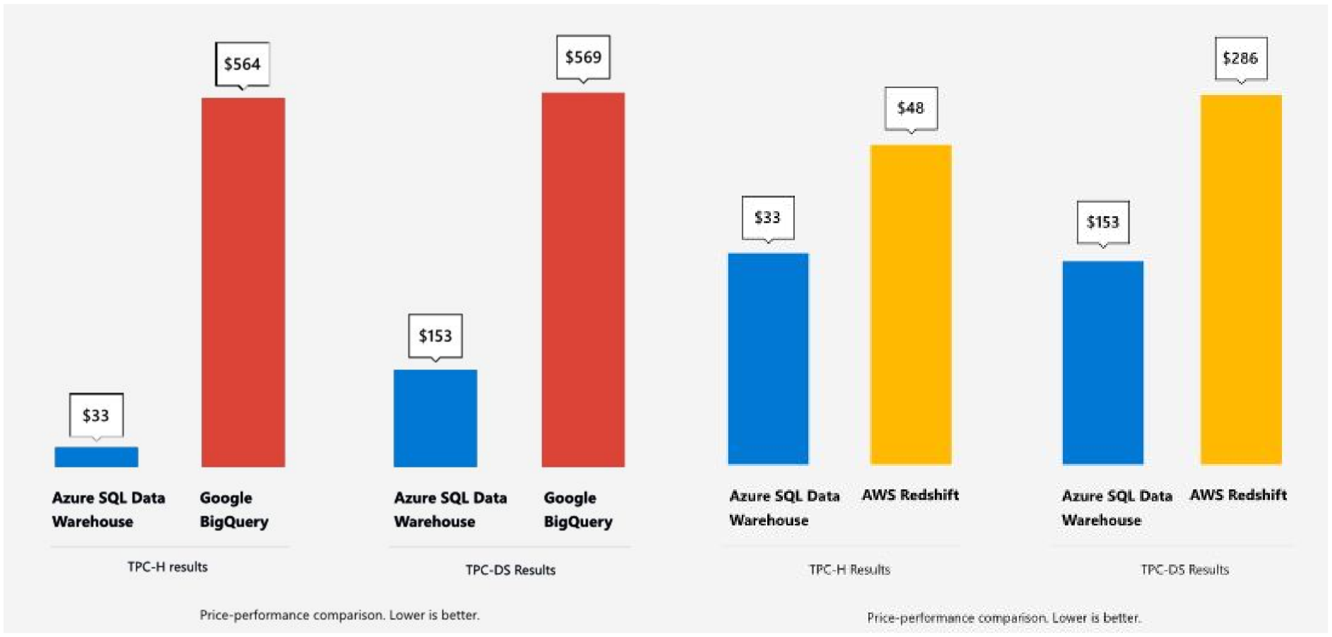
Cons:

- Uniqueness is not enforced; Amazon does not currently have a structure available to maintain data integrity. Meaning there's no way to check expressions.

- A substantial understanding of Sort/Dist keys is recommended otherwise you may run into a lot of issues as only one distribution key for a table can be used and cannot be altered later.
- "If you want to achieve consistent results when using Amazon Redshift, then you must keep your clusters below 75% for best results. If you let the clusters become overloaded with multiple queries, then you'll start to have performance issues as well. Do your best to limit yourself to 10 concurrent queries or less when working with this data warehousing solution."
- Redshift doesn't support semi-structured data types like Array, Object, and Variant.

Microsoft Azure SQL Data Warehouse

Pricing Calculator:
https://azure.microsoft.com/en-ca/pricing/calculator/
Pricing comparison (On Microsoft website)



Pros:

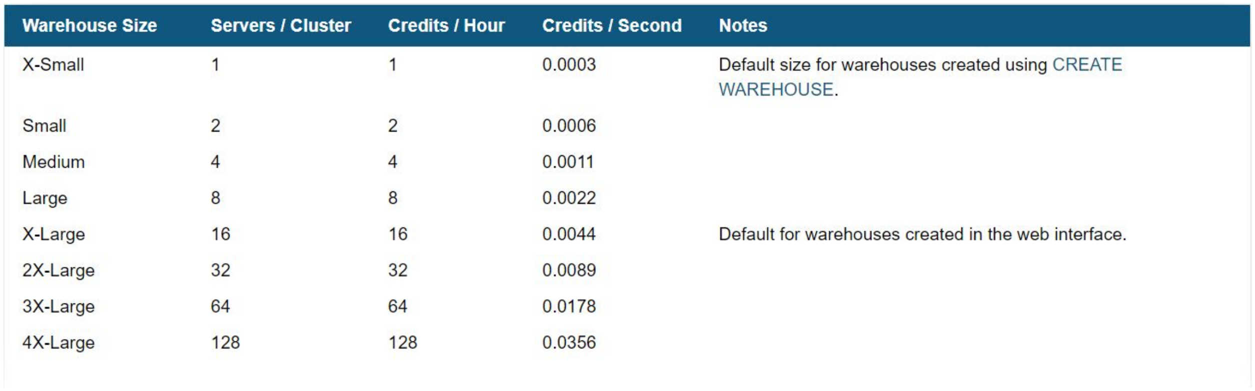
- Can independently scale storage and compute so the customers only have to pay for the query performance they require.
- Advanced security with always-on encryption, and threat detection.
- Only pay for what you use.
- Fast
- Very easy to quickly spool up new websites, databases, or even virtual machines
- Well suited for moving single use servers to the cloud
- Database Admin is made easy
- It's virtual environment is a market stand-out
- Nice interface where you can see everything in different views such as graphs, logs, etc.

Cons:

- Can be hard to gauge how much it will cost (you pay for what you use)
- Interface is complex and uses Curve
- Changing a lot, sometimes adopting the change can be difficult
- Some people are saying it's expensive for them
- Azure renames some of its services frequently, so it can be hard to keep on top of what's what

Snowflake

Pricing: Snowflake utilizes per-second billing (with a 60-second minimum each time the warehouse starts) so warehouses are billed ***only*** for the credits they actually consume.



Warehouse Size	Servers / Cluster	Credits / Hour	Credits / Second	Notes
X-Small	1	1	0.0003	Default size for warehouses created using CREATE WAREHOUSE.
Small	2	2	0.0006	
Medium	4	4	0.0011	
Large	8	8	0.0022	
X-Large	16	16	0.0044	Default for warehouses created in the web interface.
2X-Large	32	32	0.0089	
3X-Large	64	64	0.0178	
4X-Large	128	128	0.0356	

*Credit Consumption (https://www.snowflake.com/wp-content/uploads/2017/07/CreditConsumptionTable.pdf)

Pros:

- Fast
- Can be hosted on Azure or Redshift
- Separates usage from storage in billing, unlike Redshift
- Compatible with structured and unstructured data
- Very easy to use, requires no management
- Features compute, storage and cloud services that scale
- "Data Sharehouse" - The ability to share and receive data from other organizations with ease
- Wide support for ETL tools

Cons:

- When hosted on Azure, there are some limitations
  - Some third-party applications are not compatible
  - Does not have anything like Amazon's PrivateLink
  - Virtual Private Snowflake (VPS) is not offered while hosted on Azure

Summary:

The main benefits of Snowflake are its speed, its ability to optimize performance, and its multi-cluster data architecture. Snowflake outperforms other data warehouses, which is a huge factor in a business environment that relies heavily on data-driven insights.

Google BigQuery

Pricing:https://cloud.google.com/bigquery/pricing

Billed per query, or flat-rate ($10k/mo or annually $8.5k/mo). Storage is &0.020/GB

Pricing Calculator: https://cloud.google.com/products/calculator/

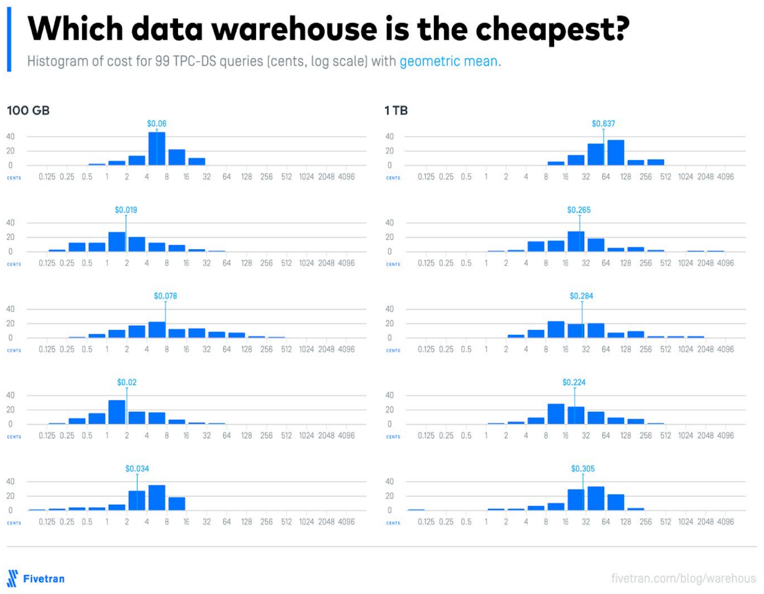
Pros / Features:

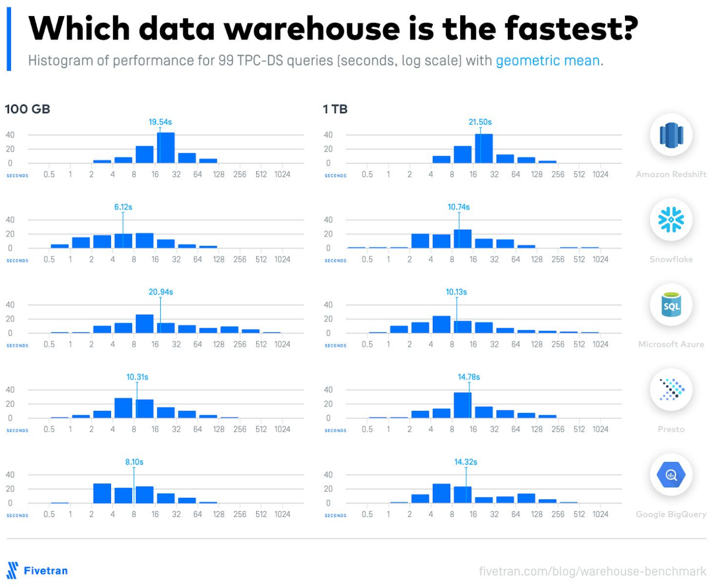
- Storage and compute separation
  - With BigQuery's separated storage and compute, you have the option to choose the storage and processing solutions that make sense for your business and control access and costs for each.
- Automatic backup and easy restore
  - BigQuery automatically replicates data and keeps a seven-day history of changes, allowing you to easily restore and compare data from different times.
- Petabyte scale
  - Get great performance on your data, while knowing you can scale seamlessly to store and analyze petabytes more without having to buy more capacity.
- Flexible pricing models
  - On-demand pricing lets you pay only for the storage and compute that you use. Flat-rate pricing enables high-volume users or enterprises to choose a stable monthly cost. For more information, see BigQuery pricing or cost controls.
- Data governance and security
  - BigQuery makes it easy to maintain strong security with fine-grained identity and access management with Cloud Identity and Access Management, and your data is always encrypted at rest and in transit.
- Foundation for BI
  - BigQuery forms the data warehousing backbone for modern BI solutions and enables seamless data integration, transformation, analysis, visualization, and reporting with tools from Google and our technology partners.
- Flexible data ingestion
  - Load your data from Cloud Storage or stream it into BigQuery at thousands of rows per second to enable real-time analysis of your data. Use familiar data integration tools like Informatica, Talend, and others out of the box.
- Programmatic interaction
  - BigQuery provides a REST API for easy programmatic access and application integration. Client libraries are available in Java, Python, Node.js, C#, Go, Ruby, and PHP. Business users can use Google Apps Script to access BigQuery from Sheets.

Cons:

- Even though the cost is pay-per-use, it's still expensive. This may make the program impractical for companies that won't use it frequently enough or for high-powered processing as it is meant for.

- Sometimes it is difficult to import data from alternate sources and manage it. The integrations between BQ and other online cloud storage aren't always a smooth transfer.
- More documentation is needed. Takes a strong learning curve to get used to.





Source: https://fivetran.com/blog/warehouse-benchmark

Additional Sources:

https://www.youtube.com/watch?v=GgLKodmL5xE&t=268s

https://hevodata.com/blog/amazon-redshift-pros-and-cons/

https://brandongaille.com/23-pros-and-cons-of-amazon-redshift/

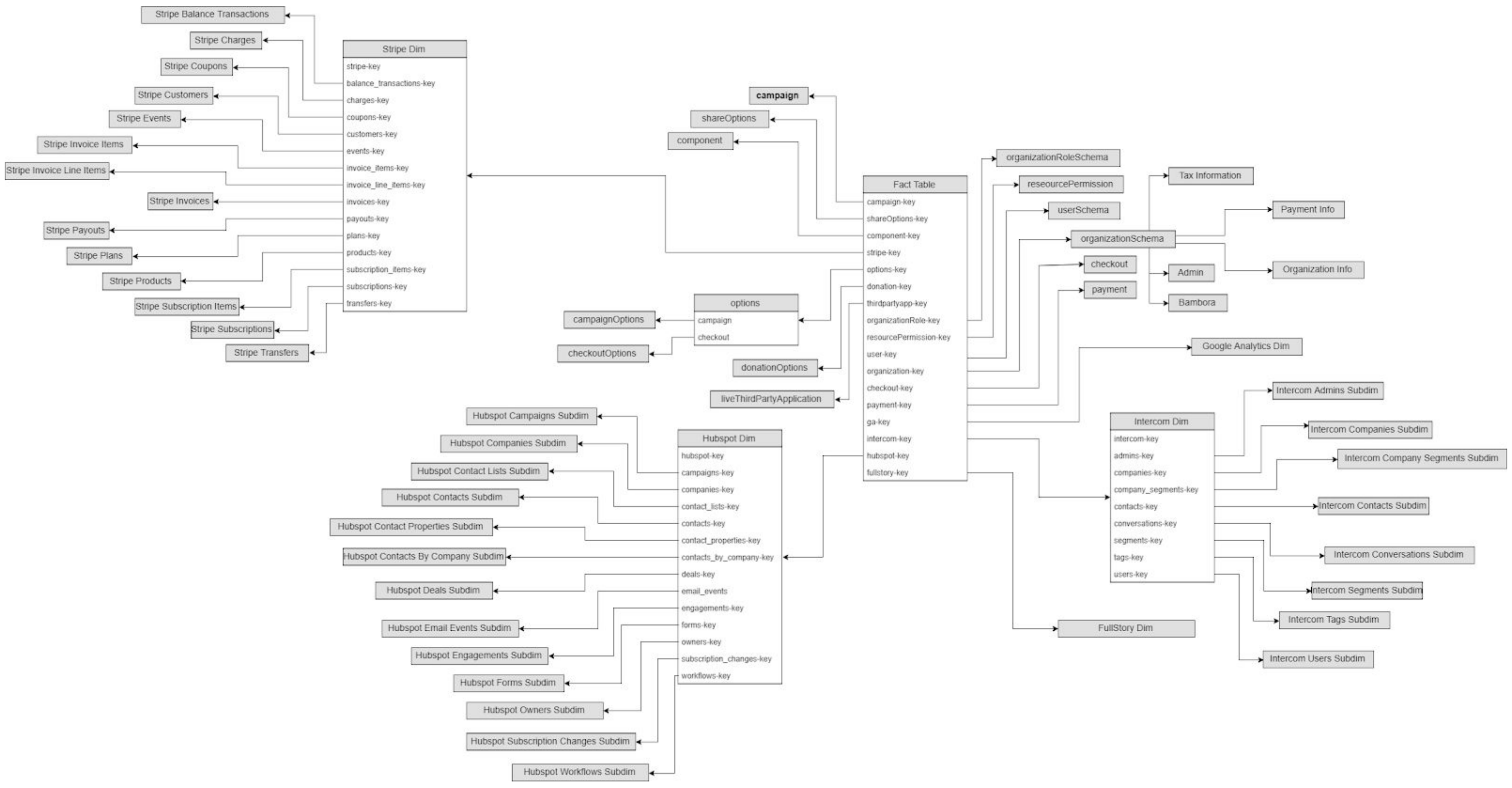
https://www.stitchdata.com/resources/snowflake-vs-redshift/

https://docs.snowflake.net/manuals/user-guide/warehouses-overview.html

http://comparecamp.com/snowflake-review-pricing-pros-cons-features/

Data Warehouse Schema

Version 1



Client Feedback

- Remove Bambora data
- Remove any attributes that can not be analyzed/aggregated (All string inputs that aren't IDs or have set values; enums)

Notes

- This schema is a foundation for what the data warehouse schema will look like. This is **not** the final schema for the data warehouse.

Caching

Caching is a technique to reduce time of data lookups&checkings. Instead of reading the data directly from the database, the data can be read directly from a cache on the computer that needs the data

Considering the nature of our project, caching is not really needed to speed up data lookups & checkings because of the scale of data and budget. In case that our client is requiring caching, we would directly use Amazon ElastiCache for several reasons. First of all, we store our clients' data on Amazon Redshift. It makes us very easy to deploy and operate the cache in the cloud with Amazon ElastiCache . Secondly, Amazon ElastiCache would significantly increase read throughput with low latency. Lastly, Amazon ElastiCache is already a mature solution for many industries.

Software Engineering Best Practices

1. **YAGNI** - ***You ain't gonna need it.*** Basically code that you're "gonna" use in the future but ends up becoming dead code or just a waste of lines that can mess up other code or cause confusion later on, which isn't a good idea.
2. **DRY** - ***Don't repeat yourself***, pretty much aims at reducing repetition in software patterns, which works hand in hand in reducing redundancy in the code.
3. **KISS** - ***Keep it simple, stupid!*** Not over complicating everything and adding unnecessary clutter.
4. **SOLID** - It is a bunch of Object Oriented principles.
   S*ingle responsibility*

   O*pen-closed*

   L*iskov substitution*

   I*nterface segregation*

   D*ependency inversion*

In our Design, we will take KISS as our approach. We want to keep things simple and not overcomplicate tasks. The code,designs and documents should be simple, stupid and easy to understand, because we do not want to spend too much time on generalizing what have done previously.