ECOLE POLYTECHNIQUE DE BRUXELLES

INFO H419: Data Warehouses

TPC-DS report: Amazon Redshift

Authored by:

Rana İşlek

Simon Coessens

Berat Furkan Koçak

David García Morillo

Professor:

Esteban Zimányi

2023

Contents

1 Introduction

1.1 Overview of the project

This project mainly focuses on using popular *TPC-DS benchmark* on a chosen data warehouse in order to evaluate the performance on the different capabilities of the warehouse while having hands on experience with the process.
The findings in this report mainly focus on comparative presentation of three main performance indicators: Data loading, query execution (with both single and multiple/concurrent users) and data maintenance. The testing is performed on different scale factors on a test purpose generated dataset.
As a result of the project, we will be presenting why *Redshift* is the best warehouse choice for your team, supported with the findings gathered from the benchmarks.

1.2 Choice of Amazon Redshift as the platform

Our research on currently used Data Warehouse solutions made us clear what are the main features that we seek from them. We want a powerful, fully managed data warehousing solution that is well-suited for handling large-scale data analytics.

Other than the consistent performance with different scales of data, we are also looking for a solution that provides ease of use and well-formed documentation with an efficient cost.

Keeping them all in mind, we finally came to the conclusion that Redshift gives you the best environment for applications that require large scale data warehousing solutions.

We are describing all the key features and benefits of *Redshift* that led to this decision on chapter 2.2

1.3 Objectives and expectations

Our main objective is using the popular *TPC-DS benchmark* to evaluate whether our choice of Redshift among all the other warehouse solutions will indeed give us the key benefits we want.

What are we seeking?

We want performance on high data load: We would like to assess the platform's capability to handle data loading tasks efficiently. It's expected that data loading is done in a timely manner, even with larger datasets.

We want efficiency on data maintenance: We would like to evaluate the effectiveness of data maintenance tasks, including updates, inserts, and deletes. We expect the platform to exhibit high performance with maintenance of the data warehouse

We want fast response to OLAP queries: We would like to measure the performance of OLAP query execution, both in single and multiuser scenarios. We expect The system to demonstrate rapid and reliable responses to complex queries, as required by *TPC-DS*.

We want to provide a comparative analysis: We would like to provide a comparative analysis for each scale factor to understand how the system's performance evolves as data volume increases. Additionally, compare the results with other data warehousing solutions to gain insights into *Amazon Redshift's* strengths and weaknesses. Overall, we expect from *Redshift* to perform better than average in each key fields and even best in some of them.

We want device independence: We would like to test the utilization of a serverless cloud based solution that is portable and device-independent, allowing for replication on various computing infrastructures and devices without major compatibility issues.

2 Amazon Redshift: An Overview

2.1 Introduction to Amazon Redshift: Features and Benefits

Amazon Redshift is a *Data Warehousing* solution within the *AWS* cloud. It was launched in 2013, and it was one of the first *Cloud-based Data Warehouse* solutions to be launched in the industry.

As any other *Data Warehouse* is expected to be, *Redshift* is highly **scalable**, and **performant** [GWA22], some of the reasons behind this are:

- **Columnar Storage**: *Redshift* uses a *Column Oriented Storage*, since it is an *OLAP* (Online Analytical Processing) optimised technology, more information about this in [Serd]
- **Distributed Architecture**: *Redshift* distributes the workload over several machines (nodes). You can have a glance at the high level architecture from Figure 2.1. You can read more about *Redshift* internal architecture in [Arm+22]
- *Redshift* is **backed by the *AWS* infrastructure**, which means a native and easy **integration with other *AWS* services**. This is of paramount importance, as *AWS* is as of 2023 the biggest cloud in the world [Law].

Other secondary but still important features *Redshift* support are:

- Support for querying **spatial** data. *Redshift* has partial support for the *Simple Feature Access* standard [Serh]. A useful comparison between the support for querying spatial data between *PostGIS* & *Redshift* can be found in [Cho]. Please note the comparison dates from 2020 so some additional functions could have been added, to check the full catalog of spatial functions, access the documentation. For a more complete coverage of Redshift support for spatial data, please refer to [Bor+20]
- Support for **Materialized Views** [Sere]

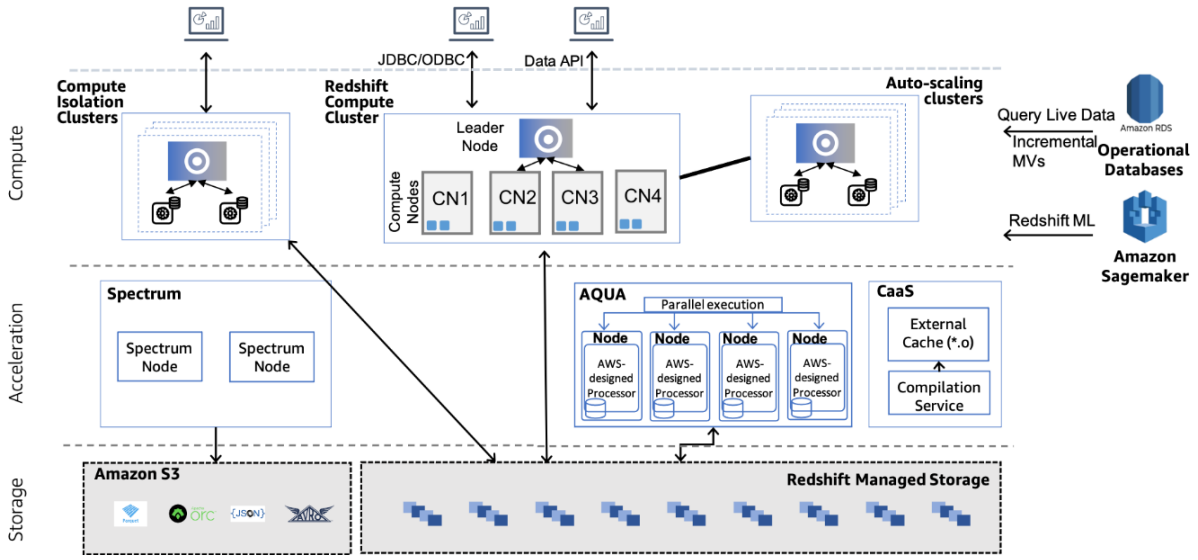

Figure 2.1: Redshift Architecture

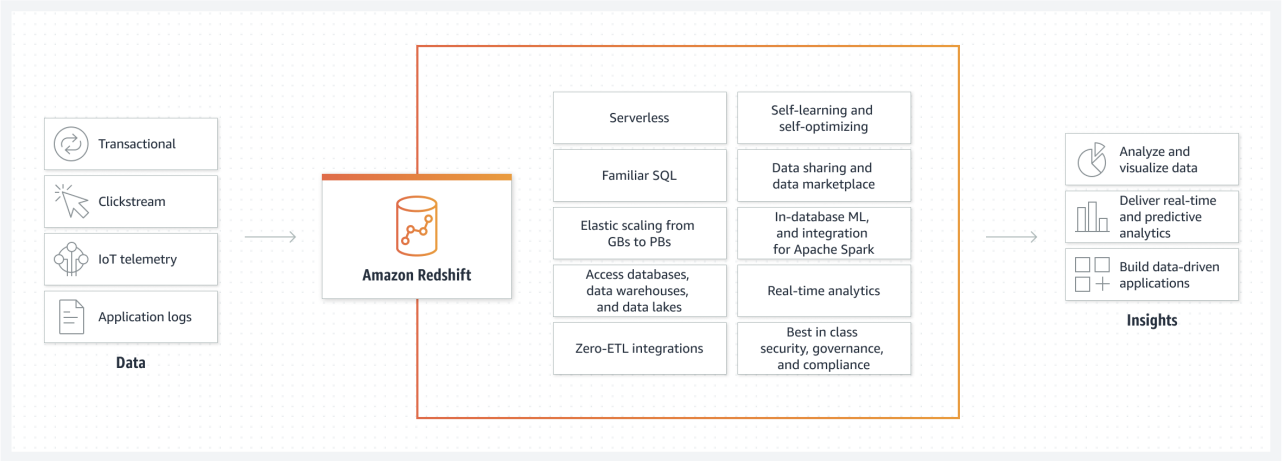

Figure 2.2: Retrieved from `https://aws.amazon.com/redshift/#`

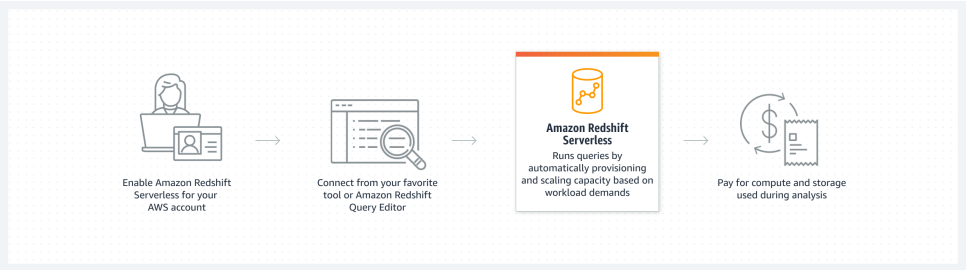

Figure 2.3: Retrieved from `https://aws.amazon.com/redshift/redshift-serverless/`

- **Schemaless data** support: *Redshift* supports the `SUPER` type, which supports `JSON` which is extremely useful to also store data without a defined schema. And it allows to query it using the `PartiQL` language. [Serf]

- **Easy integration** with Data Lakes and other data sources: As already mentioned, since *Redshift* is part of AWS, it can easily connect with other AWS services such as S3 (Storage service used as *Data Lake*), RDS (*OLTP* database service) [Serg].
  A useful feature in this context is **Redshift Spectrum**, a feature that allows you to leverage the computing infrastructure from *Redshift* and the storage from *S3*. In other words, you are able to perform queries on exabytes of data stored in *S3* using *Redshift* computing power.
  This is extremely useful to bridge the gap between your **Data Lake** stored in *S3*, and your Data Warehouse.

- Using **Machine Learning** within *Redshift*. [Seri]

- Support for a *pay as you use* pricing model with **Amazon Redshift Serverless**, that allows you to not have to configure the size of the cluster but rather let *Redshift* scale automatically based on the demand [Serb]. More on this later.

2.2 Pricing

—-TODO—- Pricing for *Redshift Spectrum* is based on the amount of data scanned by the service, you can learn more in [Sera].

2.3 Setting Up Amazon Redshift

We now proceed to cover the two main ways you could use set up Redshift.

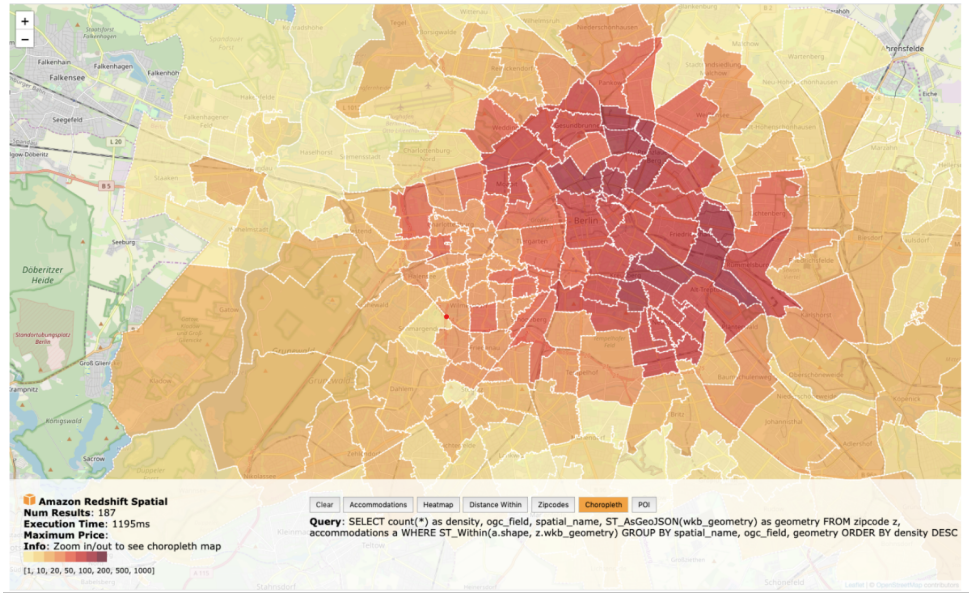

Figure 2.4: Retrieved from the paper "Unified Spatial Analytics from Heterogeneous Sources with Amazon Redshift"

2.3.1 Manual Redshift Cluster

This is the typical and oldest way to set up Redshift, and it consists of manually selecting the infrastructure that will back up Redshift.

On this option, you will need to configure the following:

- **Number of nodes** (2-32)
- **Type & Size of the nodes**
- **Permissions** of the cluster to access other AWS services
- **Networking** configuration
- Other more advanced topics such as backup policies, alarms, between others.

Pricing

On this modality, you are paying for both the data stored, and for the provisioned compute nodes allocated in the cluster. [Sera]

2.3.2 Redshift Serverless

Recently launched in 2022 [Serc], *Redshift* now allows you to adapt the **serverless paradigm**, and it continues the current trend on the cloud services on offering *pay-as-you-use* pricing models, where you are billed uniquely by the resources you actually consume, without having to provision

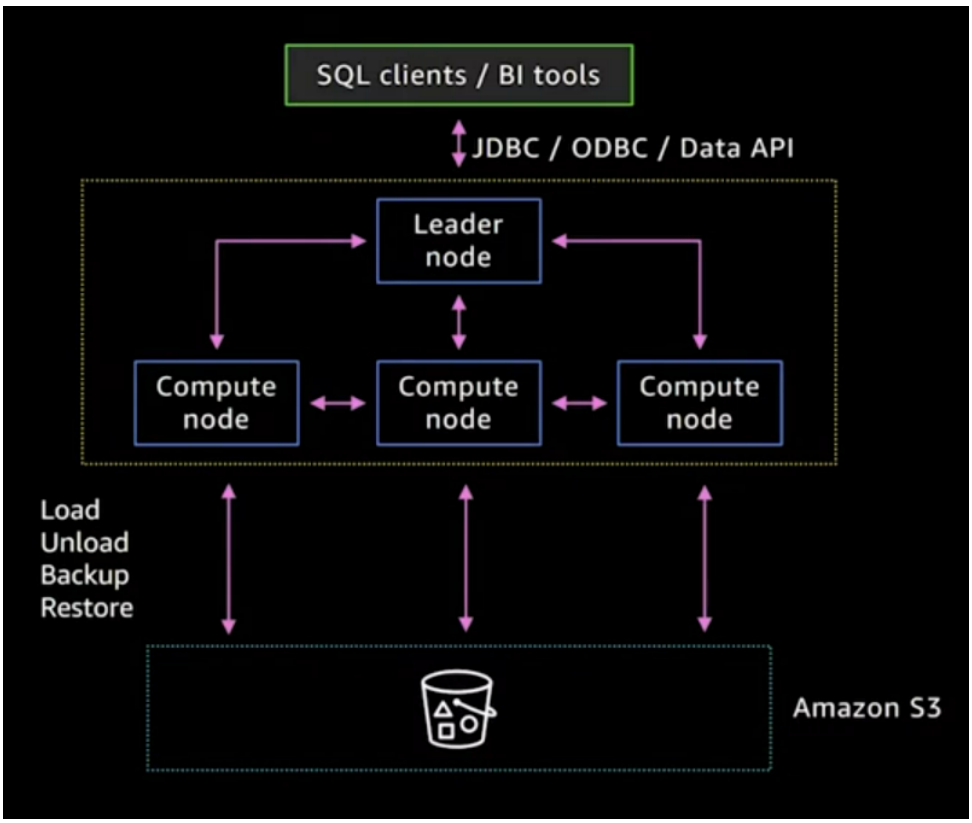

Figure 2.5: Architecture of a Redshift Cluster. Image taken from `https://youtu.be/13iIj34nkQE?t=175`

beforehand any infrastructure. This set-up is ideal for **irregular workloads** where the demand cannot be known beforehand.

This is achieved by *Redshift* having the **storage** and **compute** layers decoupled from each other, as you can observe in 2.10, which allows *Redshift* to spin up only the necessary nodes at each moment based on the demand.

To use *Redshift Serverless*, you don't need to specify the amount of nodes or the size, but rather just the base *RPU* capacity, which can be seen as the indicator to tell *Redshift* how fast do we wanna scale up when meeting higher demand.

Pricing

In this model, you only pay hourly for the read capacity used based on the time taken and data-accessed in the queries, measured in a custom metric denominated *RPU (Read Processing Unit)*. [Sera]

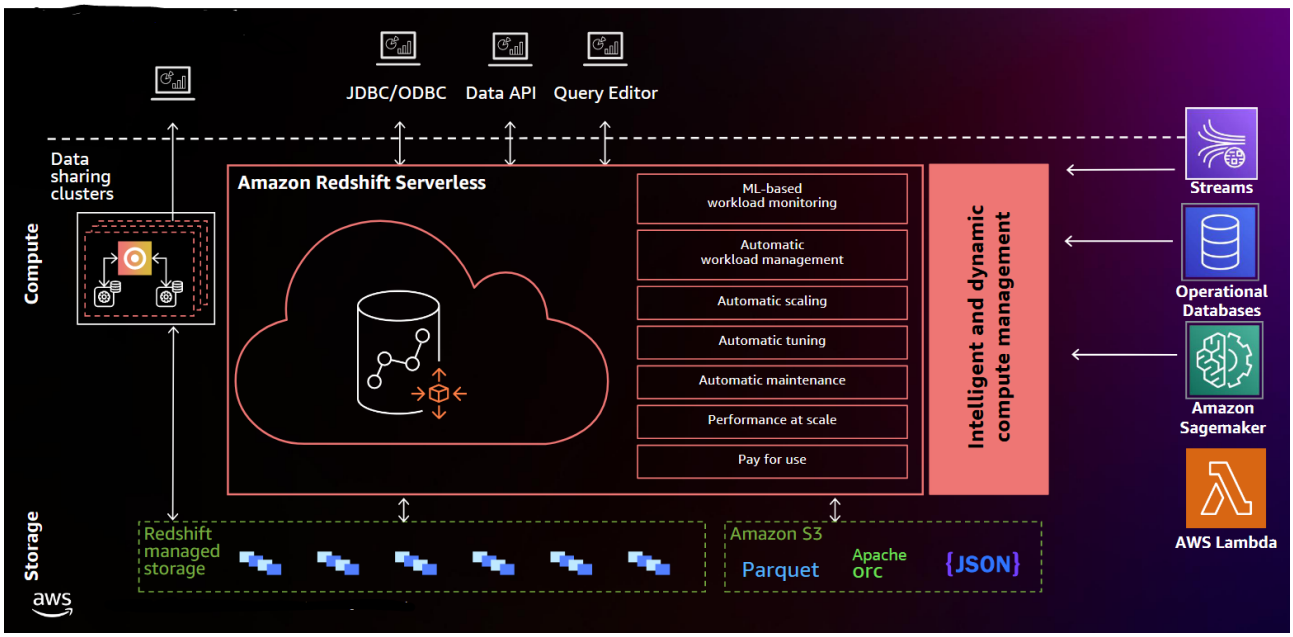

Figure 2.6: Redshift Serverless architecture. Image taken from `https://youtu.be/eIZcoXg39tg?t=1721`

2.4 Comparison with other data warehousing solutions

We find it useful to name the current main competitors of data warehousing solutions: *Amazon Redshift*, *Google BigQuery*, Snowflake and Azure SQL DW.

It is not easy to evaluate their advantages and powerful sides over each other since there are several key factors come into play. We will be presenting our findings from several papers and publications on different key factors. We highly recommend the reader of this report to be always skeptical about these results and mind the context of their own business application's requirements instead of very top view comparisons. As we show below, dependent on the nature of some queries that contain different levels of aggregations, sorts, joins and views, performance of each warehouse solution will change from one query to another.

Pricing, Query execution performance, scalability and integration are the main categories that we are providing our comparison on.

2.4.1 Pricing Comparison

On a TPC-DS benchmark applied on 30TB of dataset with a data model that consists of 24 tables — 7 fact tables and 17 dimensions performed by GigaOm Analytic Field Test on 2019, Figure **??** shows the results obtained for cost:

In terms of price per performance, Azure SQL Data Warehouse ran the GigaOm Analytic Field Test queries 30% cheaper than Snowflake in terms of cost per query per hour, and was 1.9 times

Table 2.1: Your caption here

	SQL Data Warehouse	Redshift	Snowflake Standard	Snowflake Enterprise	BigQuery Flat Rate	BigQuery Per Byte
Instance Type	DW 15000 C	dc2.8xlarge	3X-Large	3X-Large	Flat Rate	Per Byte
Nodes	30	30	64	64	N/A	N/A
Compute $/node/hour	$6.04	$4.80	$2.00	$3.00		
Compute $/hour	$181.20	$144.00	$128.00	$192.00	$55.00	$5.00 / TB
$/hour storage	$0.19					
Total Storage TB	14 TB					
Total Storage $/hour	$2.66					
Price Basis:	2,996	7,143	5,793	5,793	37,283	113 TB
Price-Performance	$153.01	$285.73	$205.97	$308.96	$569.60	$1,310.00

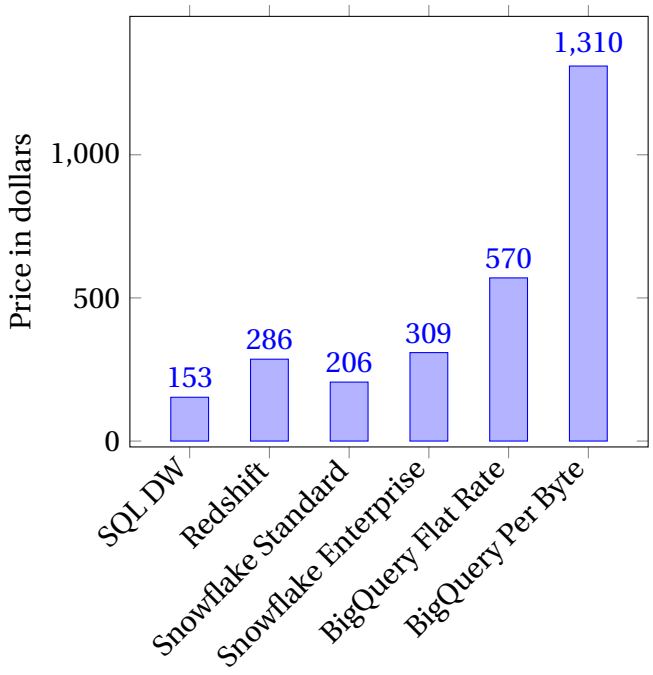

Figure 2.7: Price Performance Comparison (Lower is better)

more cost-effective in terms of cost per query, per hour than Redshift.[McK]

Even though Redshift is not the best when it comes to pricing, we can safely say that it is much better compared to Google BigQuery.

2.4.2 Query Execution Performance Comparison

According to the same publishing mentioned on pricing, one can easily see from figure 2.8 below the query execution time performances of these four different warehouse solutions

According to the publish on AWS by Hunt R. on 2016, TPC-DS benchmarking performed on BigQuery and Redshift with standard 99 queries. The result is that Amazon Redshift outperformed BigQuery on 94 of the 99 queries. [Hun]. Refer figure **??** for the results.

In the report, it is also stated that nine of the 99 queries did not even run successfully on BigQuery because it doesn't fully support standard SQL and has other limitations. We must put emphasize that this report is from 2016 and things might have changed now. This only shows us that

Table 3: GigaOm Field Test Results for All Queries

Query	SQL DW	Redshift	Snowflake	BigQuery	DW	RS	SF	BQ
1	6.580	6.189	5.253	34.285				
2	42.587	10.818	38.983	161.081				
3	3.360	5.118	12.559	35.864				
4	129.220	151.706	138.531	1,346.933				
5	32.990	110.544	59.957	340.166				
6	13.237	6.617	30.881	480.618				
7	14.056	13.934	59.343	66.102				
8	16.120	7.058	18.829	41.122				
9	21.163	61.230	127.016	88.784				
10	15.423	64.724	26.863	59.197				
11	72.667	71.718	83.539	806.247				
12	5.220	2.318	6.101	19.462				
13	13.567	21.465	84.656	164.975				
14	33.963	328.102	183.009	407.651				
14b	45.330	334.808	198.516	2,502.835				
15	13.837	3.991	5.611	121.217				
16	31.563	784.880	7.572	128.854				
17	25.990	16.013	40.055	804.656				
18	21.243	46.531	13.960	803.040				
19	11.163	10.401	34.836	31.275				
20	6.067	3.589	2.883	37.562				
21	3.890	0.485	3.496	7.310				
22	4.820	3.050	7.750	24.510				
23	78.133	238.914	244.657	2,429.327				
23b	74.137	238.130	247.550	2,086.158				
24	19.190	46.293	85.579	542.550				
24b	27.020	43.236	89.010	565.740				
25	22.493	21.702	56.871	733.442				
26	9.690	6.987	7.670	29.952				
27	9.800	44.325	57.998	129.586				
28	23.860	42.985	105.227	51.956				
29	32.887	21.237	59.693	764.899				
30	15.750	3.504	7.261	37.552				
31	32.827	38.903	40.461	98.771				
32	6.413	6.625	3.249	38.277				
33	14.300	16.534	46.862	135.034				
34	15.923	14.113	62.151	33.873				
35	26.960	180.582	20.444	125.728				
36	6.327	37.580	36.046	117.064				
37	3.453	1.763	7.597	20.802				
38	51.353	32.868	46.552	78.514				
39	7.837	1.480	3.779	64.299				
39b	9.774	1.586	5.822	66.600				
40	13.930	44.008	4.204	160.672				
41	2.110	0.386	0.414	1.218				
42	2.663	6.401	20.497	27.365				
43	11.994	7.696	14.561	25.394				
44	4.913	16.603	42.177	31.217				
45	14.867	3.669	10.644	93.772				
46	14.327	15.071	58.279	49.127				
47	16.357	48.060	49.092	265.334				
48	8.134	13.048	41.527	117.884				

Query	SQL DW	Redshift	Snowflake	BigQuery	DW	RS	SF	BQ
49	30.980	33.545	79.611	127.813				
50	26.424	67.223	43.481	254.204				
51	19.227	32.081	31.726	415.552				
52	3.010	5.968	13.612	23.971				
53	3.987	7.468	15.031	30.939				
54	15.453	14.750	38.669	403.032				
55	2.920	6.040	13.373	24.797				
56	12.080	14.789	31.689	41.448				
57	20.243	24.508	20.175	96.284				
58	13.920	11.455	25.054	110.161				
59	84.123	47.015	64.891	200.279				
60	13.314	17.884	32.738	58.777				
61	14.640	21.242	81.391	76.748				
62	6.360	3.840	10.684	17.466				
63	4.117	7.668	23.836	28.965				
64	85.070	248.720	176.065	1,917.566				
65	22.410	22.615	54.556	137.121				
66	9.600	14.098	21.730	39.510				
67	651.083	464.580	641.240	2,291.993				
68	20.517	16.818	71.217	72.956				
69	16.940	43.005	22.695	51.175				
70	10.690	32.247	29.331	115.037				
71	15.770	21.409	73.426	48.157				
72	39.396	1,436.155	22.067	3,600.122				
73	12.313	6.647	36.721	45.325				
74	86.660	60.951	63.694	844.625				
75	100.947	79.675	104.901	530.170				
76	16.187	12.556	66.669	81.148				
77	16.440	21.846	57.154	227.059				
78	96.730	442.275	503.549	1,871.047				
79	16.473	20.711	74.259	55.035				
80	46.560	232.172	99.704	1,969.868				
81	20.970	5.074	8.323	40.865				
82	4.580	2.951	12.227	22.837				
83	11.747	2.228	4.381	14.643				
84	4.760	1.215	4.139	6.042				
85	18.257	13.314	14.097	173.184				
86	4.220	6.991	8.454	33.445				
87	55.556	34.499	40.164	1.266				
88	22.510	35.104	80.431	118.398				
89	6.520	8.791	27.493	33.241				
90	6.493	3.042	11.587	16.891				
91	7.980	1.258	2.796	11.414				
92	4.950	4.432	11.603	29.035				
93	28.863	34.647	45.841	255.616				
94	18.466	117.805	20.459	106.034				
95	28.330	72.557	88.040	2,944.355				
96	3.214	5.156	26.437	24.398				
97	36.087	27.399	39.216	119.807				
98	6.167	8.770	21.643	59.361				
99	10.240	6.475	14.694	27.683				
TOTAL	2,995.972	7,143.242	5,793.037	37,282.718				

Figure 2.8: Retrieved from https://gigaom.com/report/cloud-data-warehouse-performance-testing/#post-id-960556

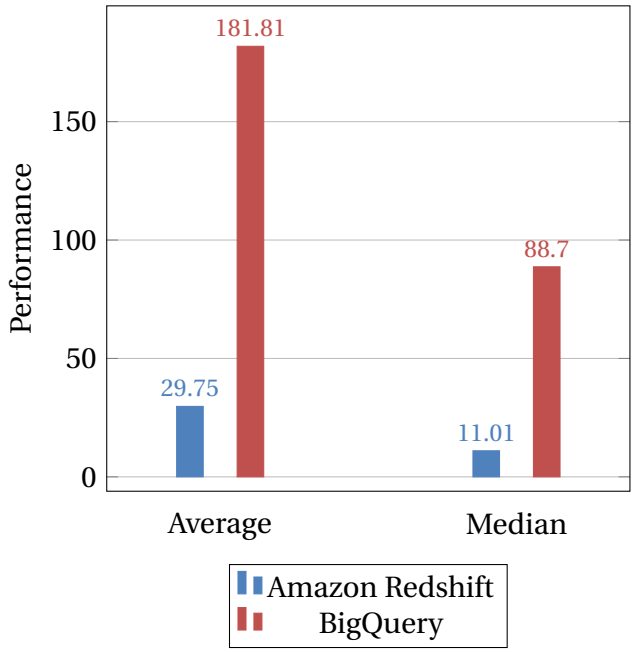

Figure 2.9: Bar Plot of Performance (Redshift vs. BigQuery)

Redshift accomplished this much faster than BigQuery, making their hand stronger in the industry.

2.4.3 Scalability Comparison

This is another crucial factor. Amazon Redshift allows for vertical scaling by adding more nodes or horizontal scaling using *the Auto WLM* feature, making it suitable for various workloads. Google BigQuery is highly scalable and automatically adjusts to workload demands. Snowflake, on the other hand, offers near-infinite scalability by dynamically adding compute resources.

On Figure **??** below, one can find the different scalability capabilities of each warehouse. This figure is taken from Firebolt's whitepaper named A COMPARISON OF DATA WAREHOUSES AND QUERY ENGINES.

2.4.4 Integration Comparison

Integration of several services on cloud can be an important decision depending each application's requirements.

Overall Amazon Redshift provides us easy integration with the AWS ecosystem, making it preferable for us since we are somehow familiar using the other AWS services as well as there are a lot of documentation for each of them.

Google BigQuery also integrates well with other Google Cloud services.

Snowflake stands out between them by supporting both AWS, Azure, and GCP, and offering

	Redshift	Athena	BigQuery	Snowflake	
Scalability					
Elasticity- Scaling for larger data volumes and faster queries	Available via "Elastic Resize" - slow and limited, downtime required	Fully abstracted - Athena decides how many resources to use behind the scenes. No way to "force" more compute for faster queries.	Fully abstracted - BigQuery decides how many "slots" to allocate for each query. No way to "force" more compute for faster queries.	Cluster resize; No choice of node size	Granular cluster resize with node types, number of nodes
Elasticity - Scaling for higher concurrency	15 concurrent queries per cluster, Autoscaling up to 10 clusters	Limited to 20 concurrent queries by default	Limited to 100 concurrent users by default	8 concurrent queries per warehouse by default, Autoscaling up to 10 warehouses.	A single engine can handle hundreds of concurrent queries. Adding more engines is manual.

Table 2.2: Comparison of Scalability Options

connectivity options to popular BI tools.

According to the classification in Figure 2.10, we can see that currently AWS platform is seen as the top leader in service solutions.

2.4.5 Comparison Results

As a result, the choice between these data warehousing solutions highly depends on one's specific project requirements, existing cloud infrastructure, and budget constraints. Each platform has its unique strengths that can be utilized in different application scenarios.

According to all the research and findings above, we can state the following:

1. Amazon Redshift is cost-wise more effective than BigQuery and charges a bit more compared to SQL DW and Snowflake.

2. Amazon Redshift outperforms BigQuery on query execution performance and shares the top place with SQL DW. We also should state that Snowflake performs an overall stable query performance even though hardly manages to outperform Redshift and SQL DW on individual TPC-DS queries.

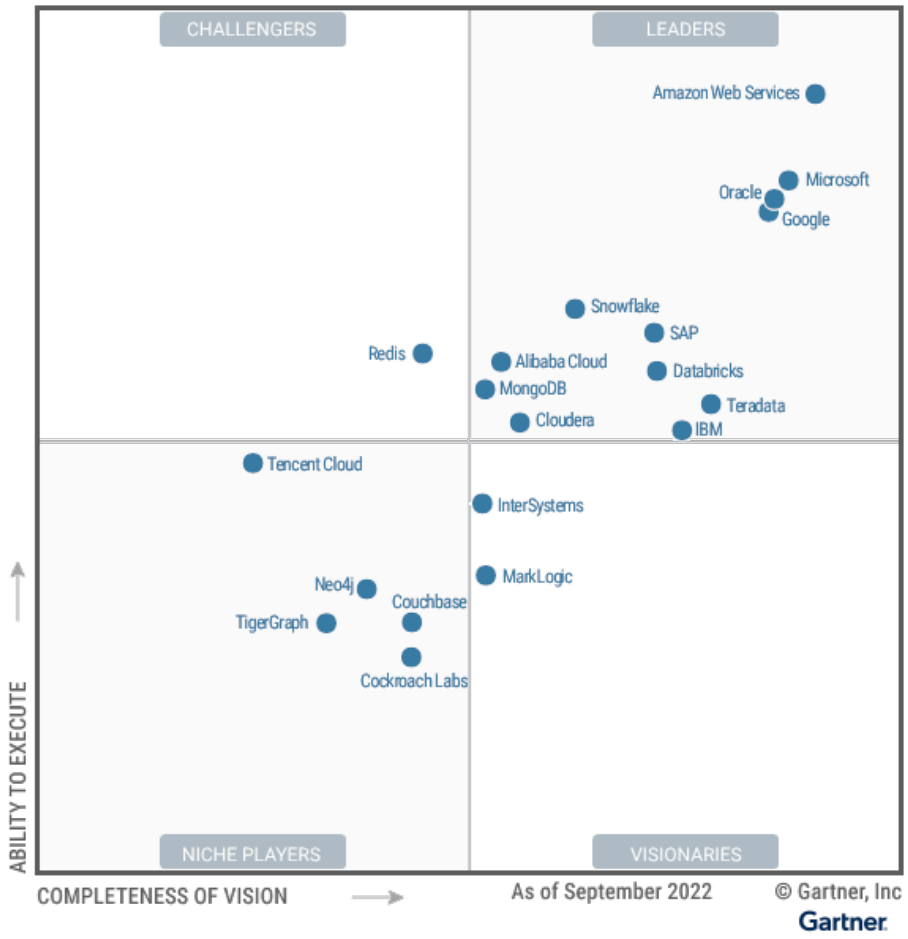

Figure 2.10: Gartner (December 2022)

3. Scalability decision may differ according to application context. On concurrent access, Redshift autoscales up to 10 clusters with 15 concurrent queries per cluster, which is more than Athena's limit of 20 concurrent queries, BigQuery's limit of 100 concurrent queries and Snowflake's 8 concurrent queries per warehouse where it autoscales up to 10 warehouses.

4. Integration availabilities may also differ according to application and business context. We can state that if the customer is looking for support of multiple sources, Snowflake can be an interesting option. However, we think that usage of all the available and integrated Amazon Services are more useful for our project.

3 Understanding TPC-DS Benchmark

3.1 Introduction to DSS and DSS benchmarking

With the evolution of Database technology grew a rising interest in analysing the stored Data. As a result of this fields as Business intelligence and Data Analytics have experienced significant growth and have become essential components of management tasks. The Database systems that support these analytical demands, utilised by Decision Support Systems, have been termed as Data Warehouses.

Comparing different system providers is crucial in any technological context. Standard industry benchmarks provide a fair way to make a comparison. The Transaction Processing Council (TPC), founded in 1988, is a non profit organisation that builds benchmarks for a wide range of Database systems. Many of the TPC benchmarks are regarded as industry-standard. As mentioned in [NP06] the TPC recognised the need for a standardized benchmark to to evaluate the performance of Decision Support Systems. The first benchmark standard, TPC-D, was released in 1994. Due to innovation in Database technology the decision benchmark has been updated several times over the last decades resulting into the TPC-DS benchmark. TPC-DS is now regarded as the industry's standard benchmark for Decision support systems. In the following section we will look into TPC-DS in greater detail.

3.2 TPC-DS specifications

3.2.1 TPC-DS Setup

The TPC-DS standard models a Data Warehouse for a multi-channel retailer where data is imported from different operational data sources. In Figure 3.1 an overview of the TPC-DS bench-

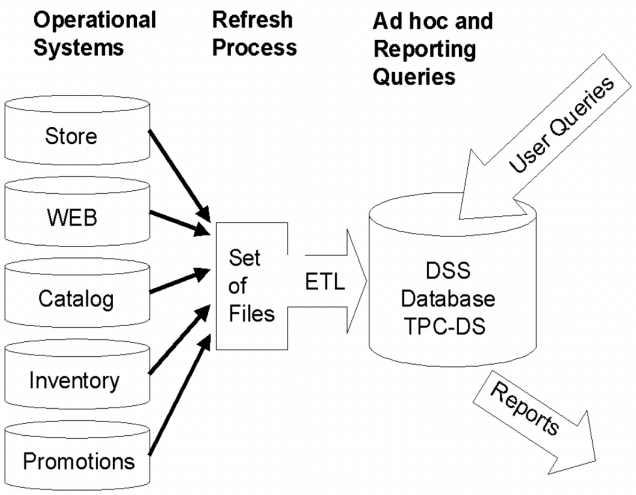

Figure 3.1: TPC-DS benchmark components

mark components can be seen. As can be seen in the Figure, Data is inmported into the DSS Database from different operational systems. It is crucial to remark here that only the DSS Database will be benchmarked in the TPC-DS benchmark. The choice for this Use-case has been made in order to have a generic model that can represent any industry involved in the management, sale, and distribution of products [TPC21].

The schema employed in TPC-DS is a snowflake schema. An example of this can be seen in Figure 3.2. This choice has been made in order to follow the industry's shift into variations of the star schema [NP06]. Once the Database schema for the TPC-DS benchmark has been created, the **dsdgen** tool can be used to populate the tables. The **dsdgen** tool is designed to generate synthetic real-world data at a specified scale. The data generated is derived from statistical distributions such as the Normal or Poisson distribution. This ensures that benchmarks performed on a same scale but using different instance data sets are indeed comparable. The use of a data generator tool offers the advantage that data needs not to be hosted by the supplier nor downloaded by the user.

3.2.2 TPC-DS Workload

The TPC-DS workload is split into multiple components. The two core components are the Data Maintenance workload and the Query workload. Together they constitute the core competencies of any Decision Support System. The query operations convert operational facts into business intelligence while the Data Maintenance operations synchronize the operational side of a business with the data warehouse. [NP06]

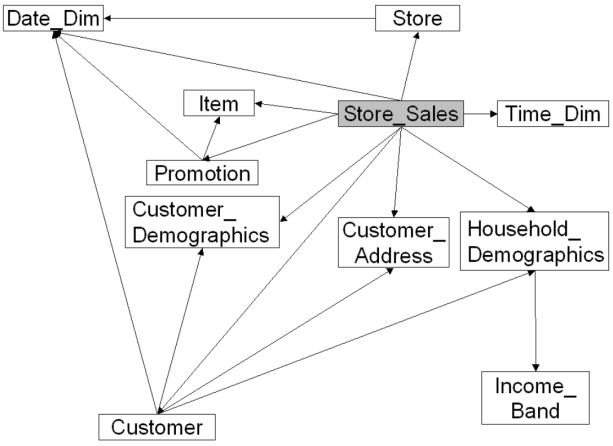

Figure 3.2: Store Sales Snowflake schema

Data Maintenance

Updating and refreshing data are essential components of modern Decision Support System (DSS) operations. For this reason part of the TPC-DS workload, in contrary to the previous TPC DSS benchmarks, includes a realistic data refresh process [TPC21]. The operations performed in the Data Maintenance workload constitute of the Data Transformation and Data Load aspects of the ETL acronym. Further specifications on the Data Maintenance operations can be found in the TPC-DS paper.

Query Workload

The TPC-DS query workload consists of 99 distinct SQL queries. The queries modeled by the benchmark cover the following:

- Ad-hoc, reporting, iterative OLAP and data mining type workloads,
- DSS relevant SQL99 functionality,
- A variety of access patterns, query phrasings, operators, and answer set constraints,
- Possibility of a wide variety to query optimizations,
- The entire data set of all TPC-DS tables and
- Complex DSS business problems.

[TPC21]

The queries can be categorized into the following types: Pure Reporting queries, Ad-hoc queries,

Iterative OLAP queries, Extraction or Data Mining queries. The critical queries here are the pure reporting queries. They respond to pre-defined questions about the business. Ad-hoc queries on the other hand answer specific and immediate business questions. An example of a Reporting and an Ad-hoc query can be seen respectively in Figure 3.3 and Figure 3.4.

Figure 3.3: Query 13 Reporting
Calculate the average sales quantity, average sales price, average wholesale cost, total wholesale cost for store sales of different customer types (e.g., based on marital status, education status) including their household demographics, sales price and different combinations of state and sales profit for a given year.

Figure 3.4: Query 17 Ad-hoc
Analyze, for each state, all items that were sold in stores in a particular quarter and returned in the next three quarters and then re-purchased by the customer through the catalog channel in the three following quarters.

An important aspect here is the assumed prior knowledge for the different query types. For the Reporting queries there is prior knowledge available, on the contrary for Ad-hoc queries there is no foreknowledge that can be assumed by the Database Administrator. For the Optimisation part this is an important aspect to take into account.

3.2.3 Benchmark execution order

In Figure 3.5 the benchmark execution order is displayed. This image is taken from the Making of TPC-DS paper, since then the order has been updated. A Power test has been added right after the Load test and a second Data Maintenance test is performed at the end. We examine each of the parts.

First a **Load test** is performed as part of the Data Maintenance workload. This includes the population of the tables and preparing necessary data structures for the performance test. The Database Load Time (T_{Load}) is the time needed to prepare for the execution of the performance test.

Immediately after the **Power Test** is executed. The Power Test will execute the sequence of 99 queries to the system in a single stream. The Time measured for this test is noted as T_{Power}.

Throughput Test 1 & 2 is performed with a Data Maintenance workload in between. The Throughput Test executes the 99 queries in a multi-user setting. A number of users is chosen and

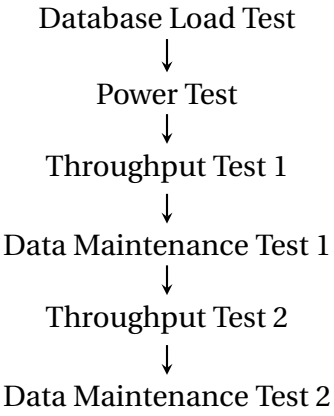

Figure 3.5: Benchmark execution order

denoted as S_q, this value is required to be even and greater or equal to four. This value remains the same for both Throughput Tests. The Time measurement for this test will be noted as T_{TT1}, T_{TT2}. The total time is denoted as T_{TT}.

The **Data Maintenance 1 & 2** tests will make changes to the TPC-DS dataset and will measure the performance of these operations. Each For each Data Maintenance Test $S_q/2$ number of refresh runs will be executed. For each refresh run the data will be seperately generated by **dsdgen**. The Time measurement for this test will be noted as T_{DM1}, T_{DM2}. The total time is denoted as T_{DM}.

3.2.4 TPC-DS Measures

The main metric used in the TPC-DS benchmark is the performance metric QphDS@SF defined in equation 3.1. We explain in detail the parameters used here.

$$QphDS@SF = \left\lfloor \frac{SF * Q}{\sqrt[4]{T_{PT} * T_{TT} * T_{DM} * T_{LD}}} \right\rfloor \tag{3.1}$$

The parameters T_{DM} and T_{TT} in the denominator have already been explained in the previous section. The other parameters are defined as follows:

- $T_{PT} = T_{Power} * S_q$
- $T_{LD} = 0.01 * S_q * T_{Load}$
- SF is the scale factor
- Q is the total number of weighted queries: $Q = S_q * 99$

4 Setting Up Amazon Redshift for TPC-DS

We will now direct our focus into the process we followed to provision the needed infrastructure to implement *TPC-DS* in **Redshift**.

For our use-case, we have used a tutorial from AWS to set-up the necessary resources for the test, you can find it in [Rob23].

4.1 Using a CloudFormation template

CloudFormation is an AWS service that allows you to concretely specify in a file what cloud services you want to create, in other words, it's the main ***IaC*** *(Infrastracture as Code)* solution for AWS. This allows for repeatable and less-error prone infrastructure setups.

In our case, the CloudFormation template we made use of spins-up the following resources:

- **Redshift Serverless**
- **EC2 instance**: A virtual machine to run the commands of the database from, as a result of using this, we are removing the dependency and implementation challenges that arises with different operating systems and configurations that we could face if we were using our own computers.

The template has some parameters we need to specify, such as the **base *RPU***, the **key pair** to use to access the EC2 instance, or the **IP-range** we want to allow to access the instrance from.

In our case, the **base *RPU*** will be set to 32 as opposed to the 128 that come as a default to try to try to lower the costs as much as possible, and we will not restrict the IP-range to access the EC2 instance (`0.0.0.0/0`).

```
[language=YAML, style=yaml, caption=Sample CloudFormation Template, label=sample-c
AWSTemplateFormatVersion: "2010-09-09"
Description: A sample template
Resources:
  MyEC2Instance: # An in-line comment
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: "ami-0ff8a91507f77f867"
      # Another comment -- This is a Linux AMI
      InstanceType: t2.micro
      KeyName: testkey
      BlockDeviceMappings:
        -
          DeviceName: /dev/sdm
          Ebs:
            VolumeType: io1
            Iops: 200
            DeleteOnTermination: false
            VolumeSize: 20
```

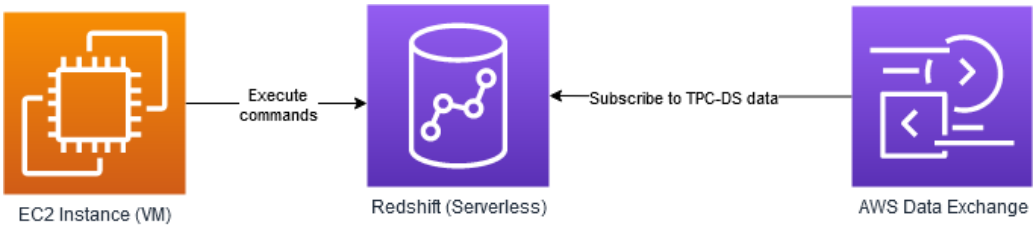

Figure 4.1: Architecture for running the queries

4.2 Creating data schema, and loading the data

As Amazon is aware of *TPC-DS* being a widely used benchmark for *Data Warehouses*, it provides an easy, free, and fast way to load different scale factors of the benchmark data into a *Redshift cluster* using a service called *AWS Data Exchange*.

AWS Data Exchange allows you to subscribe your data source to 3rd party data. In our case, we subscribe to TPC-DS data source[1] without having to worry about the *DDL* (Data Definition Language) and greatly easing the data loading process. Figure 4.2 shows the screen after subscribing to it.

This allowed us to quickly check the performance of *Redshift* by performing the **power** and **throughput** tests in scale factors that could not be achieved if we had chosen to run the tests only with generated data, as the *EC2* instance was not powerful enough to generate more than 10GB of data. More on this later.

[1] `https://aws.amazon.com/marketplace/pp/prodview-iopazp7irqk6s`

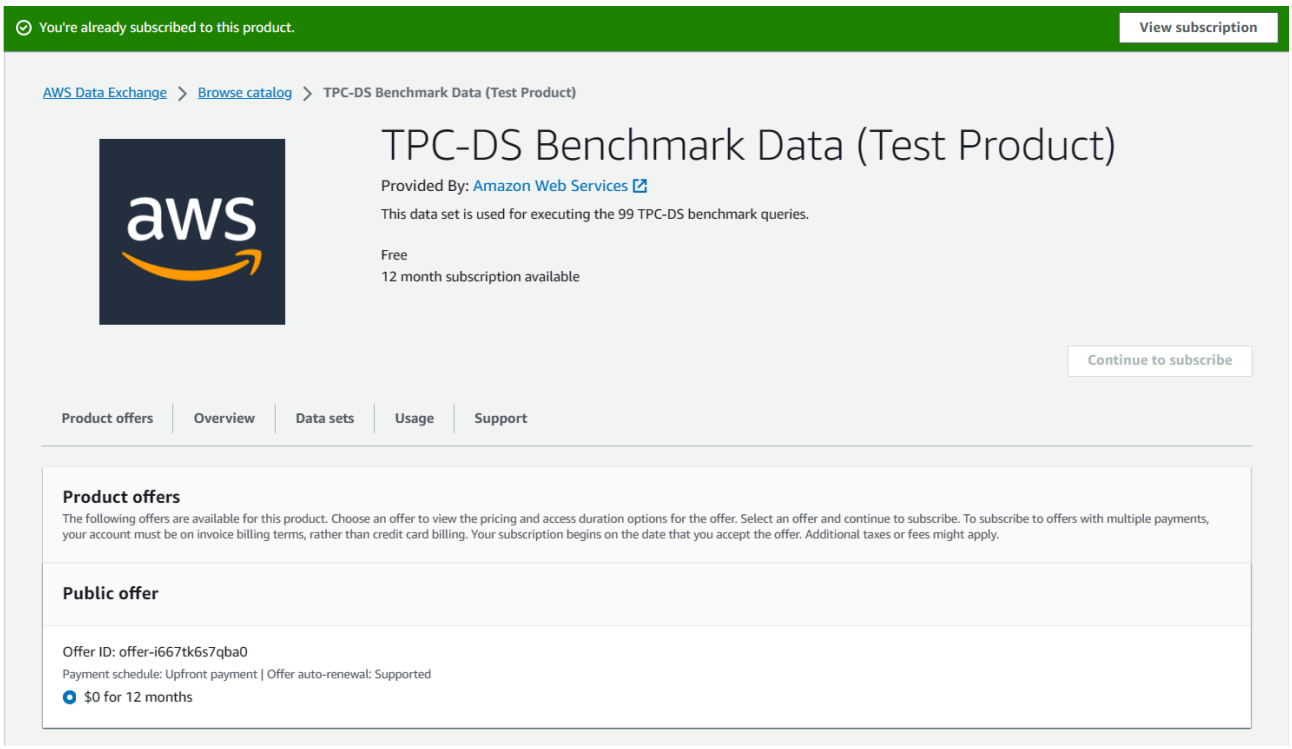

Figure 4.2: TPC-DS Benchmark Data Subscription

For the detailed instructions on setting up the Redshift cluster, please refer to [Rob23].

We will be using *TPC-DS Benchmark Datasets* of size 1GB, 10GB, 100GB and 1TB. For the **load test**, 1GB, 5GB & 10GB have been chosen due to hardware limitation.

To complete the setup, you must then follow these instructions, to find the concrete commands to be run, please again refer to the tutorial [Rob23].

1. You login into the *EC2* instance
2. Open an SQL interactive client with the *Redshift* Database as destination.
3. Create a database associating it with the already subscribed data source from *AWS Data Exchange*
4. Create the schemas with the scale factors of choice. Data will be synced.

Data will automatically be transferred into *Redshift*.

4.3 Generation of the data using *dsdgen*

As illustrated in the previous chapter, *TPC-DS* also consists of a **load** and **maintenance** tests, which require us to have control over the data ingestion process.

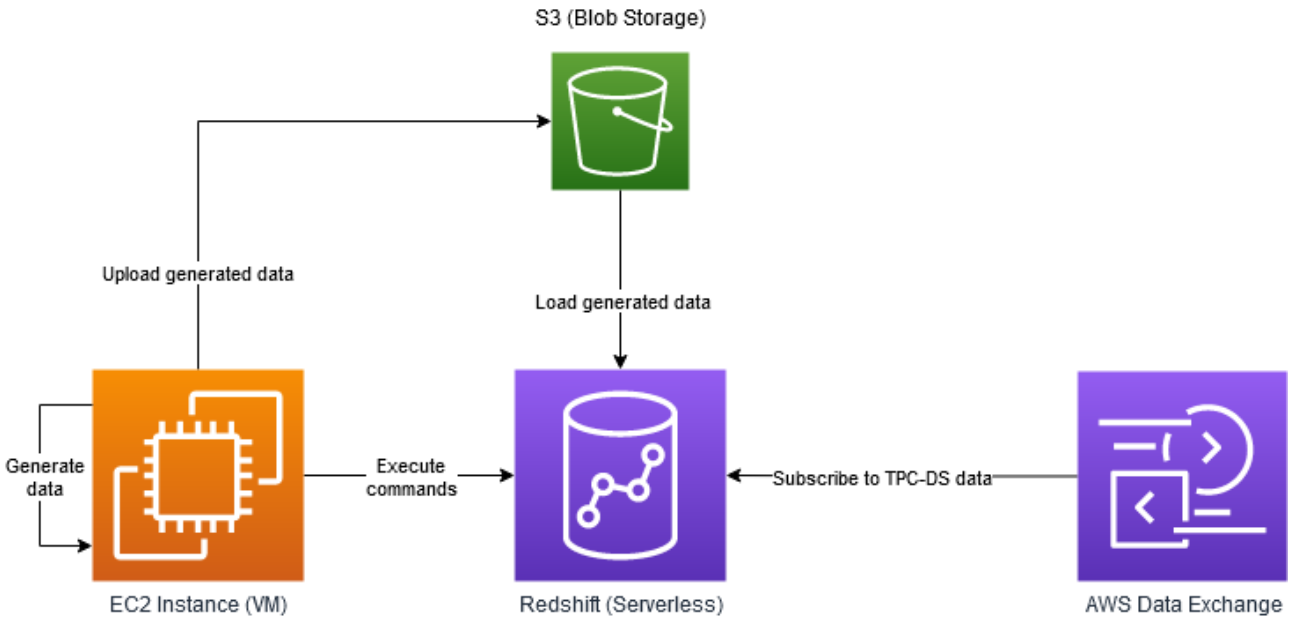

Figure 4.3: Architecture for executing the *TPC-DS* benchmark

To do that, we will make use of the provided CLI tool by *TPC* to generate the data to be ingested in the **load test** and **data maintenance test**, *dsdgen*. This data is then **uploaded to S3** to be easily ingested later by *Redshift*, to do so, we will need to configure the *Redshift* namespace to be able to read from *S3* and the *EC2* instance the ability to upload it, which means that an *IAM*[2] *Role*[3] should be assigned to those resources.

The final architecture for this test can be seen in 4.3

[2] *IAM* stands for *Identity and Access Management*, and it's the AWS service that takes care of Authorization within the cloud.

[3] A Role is a set of permissions assigned to a resource, for example, to allow a EC2 instance to upload data to S3

5 Implementing TPC-DS on Amazon Redshift

We will now cover the actual implementation of *TPC-DS* in *Redshift*.

5.1 Load Test

As we have seen in previous chapters, the **load test** consists in setting up the *Data Warehouse* for the execution of the following tests. This spans creating the necessary tables and it's relational integrity constraints as part of the *DDL*, along with ingesting into the system the appropriate data.

As mentioned above, to implement it we need to leverage the `dsdgen` CLI tool, provided by *TPC*. This will allow us to generate the data for the desired scale factors, in our case, 1GB & 10GBs. Once generated, the data will be uploaded to *S3*, to be then ingested by *Redshift* as Figure 4.3 shows.

To achieve this, the following high level steps needs to be executed in the EC2 instance:

1. Generate the data with the desired scale factor using the `dsdgen` tool.
2. Upload the data to S3, for example using the `aws` CLI.
3. Execute the *DDL* statements.
4. Ingest the data stored in *S3* into *Redshift* using the `COPY` *Redshift* command.
5. Obtain the time that executing 3. & 4. took and persist it.

T_{LD} is defined as the output of 5..

5.2 Power & Throughput Tests

For the **Power** and **Throughput** tests, as already mentioned, we didn't used the manually generated data, but rather used the *AWS Data Exchange* subscription to perform the test for bigger scale

factors.

To execute the tests, we refer again to [Rob23], since the concrete steps can be found in the tutorial, here we limit ourselves to a high level overview of the steps[1].

The only step left is running the queries as such, to do so, [Rob23] provides us with a set of shell files will execute the tests, being the entrypoint to run the tests the `rollout.sh` file located at the base directory. The following steps will be executed by it:

1. Check the connection to the *Redshift* database works as expected.
2. Select the adequate schema to run the queries against, in other words, select the scale factor chosen by us (`01_init` folder).
3. Run the **Power test**, executing the 99 queries in a sequential order. (`02_sql` folder)
4. Run the **Throughput test**, executing the queries by N different users at the same time, having the queries permutated as stated in Appendix D of the TPC-DS specification. (`03_multi_user` folder).
5. Persist logs of both **Power** and **Throughput** tests into the `tpcds_reports` schema, into the `sql` and `multi_user` tables respectively (`04_totals` folder).

To calculate the final numbers for the *TPC-DS* benchmark, we must sum the total time taken by each test, being D_{PT} the time it took to run the **Power Test**, and D_{TT} the time of the **Throughput** test.

5.3 Data Maintenance Test

For the **Data Maintenance Test**, we must generate the **refresh** data using `dsdgen` tool, and we must do it $S_q/2$ times, being S_q the number of users we decided to run in the **Throughput test**. This **refresh** data will contain several files that will be used to create `VIEW`s to insert or delete data as part of the test.

The following steps should be followed:

1. Run `dsdgen` tool $S_q/2$ times
2. Upload the data generated to S3

[1] We assume at this point that the infrastructure from Figure 4.1 has already been deployed and that you are already subscribed to the *TPC-DS AWS Data Exchange* data, as mentioned in Section 4.2

3. Create the intermediate tables that will be used in the `VIEW`s.
4. Insert into the intermediate tables the generated data.
5. Create the *VIEW*s
6. Execute the **Data Maintenance** functions as defined above $S_q/2$ times, one with each of the generated data sets, making sure deletions are executed before inserts.
7. Persist the time taken by the last step.

The output of step 7. will be D_{DM}

5.4 Main differences with the TPC-DS specification

As it can be seen from this chapter, we have not strictly performed the *TPC-DS* benchmark, since the **Power** and **Throughput** tests were performed separately from the **Load** and **Data Maintenance** tests, on another source of data.

6 Benchmark Execution

6.1 Data Load Test

6.2 Power & Throughput Tests

6.3 Data Maintenance Test

6.4 Running TPC-DS benchmark queries

When it comes to benchmark queries, we need to consider 3 different areas of tpc-ds benchmarks.

Firstly, the query execution performance can be tested with 99 standard SQL TPC-DS queries while recording the time it takes for each of them.

Second, the time it takes for data loading with different scale factors needs to be recorded.

Third, data maintenance queries and the time it takes for each of them for different scale factors needs to be recorded.

Running 99 Standard SQL TPC-DS queries

In order to run the queries, these steps should be followed:

First we need to modify tpcds _variables.sh . This is required to run rollout.sh which runs the required 99 queries on the schema specified on tpcds _variables.sh

Here are the parameters of tpcds _variables.sh:

EXT_SCHEMA="ext_tpcds3000": This is the name of the external schema created that has the TPC-DS dataset. The “3000” value means the scale factor is 3000 or 3 TB (uncompressed).

EXPLAIN="false": If set to false, queries will run. If set to true, queries will generate explain plans rather than actually running. Each query will be logged in the log directory. Default is false.

MULTI_USER _COUNT="5": 0 to 20 concurrent users will run the queries. The order of the queries was set with dsqgen. Setting to 0 will skip the multi-user test. Default is 5.

In order to run queries, cd into folder adx-tpc-ds and use the following command:
./rollout.sh > rollout.log 2>&1 &

We run our queries with 4 different scale factors namely 1,10,100,1000.

6.5 Benchmark results and metrics

- We performed a test with the 1 GB ADX TPC-DS dataset on an Amazon Redshift Serverless workgroup with 24 RPUs and with multiuser count of 3.

Figure 6.1 represents the resulting time versus query table.

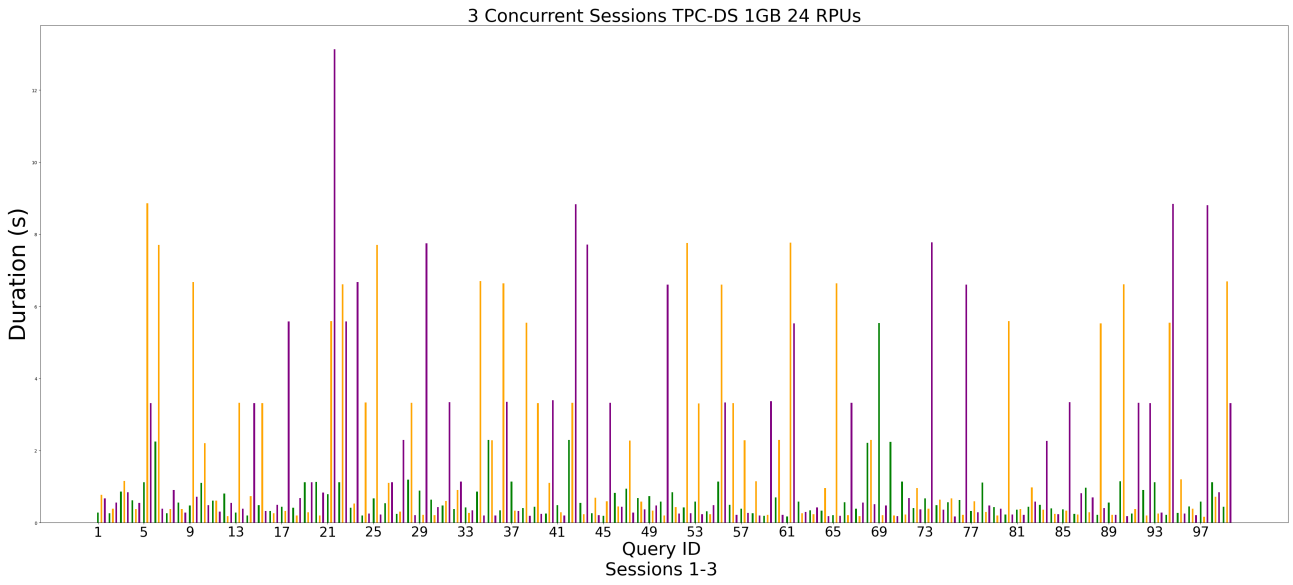

Figure 6.1: 1GB Benchmark Result

- We performed a test with the 10 GB ADX TPC-DS dataset on an Amazon Redshift Serverless workgroup with 24 RPUs and with multiuser count of 3.

Figure 6.2 represents the resulting time versus query table.

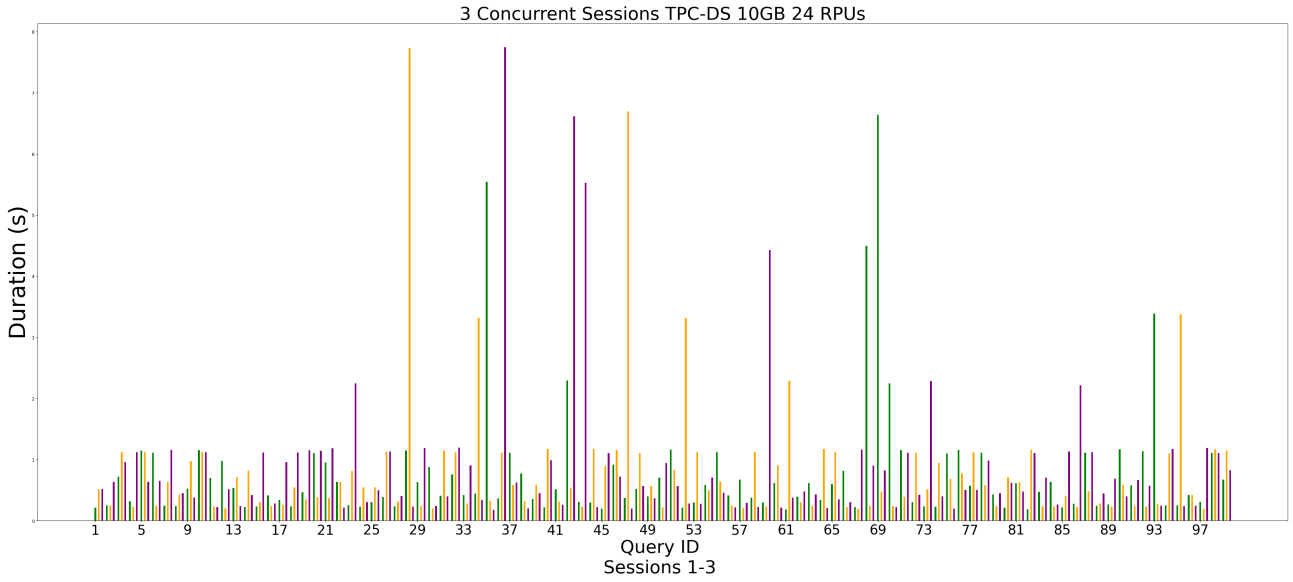

Figure 6.2: 10GB Benchmark Result

- We performed a test with the 100 GB ADX TPC-DS dataset on an Amazon Redshift Serverless workgroup with 24 RPUs and with multiuser count of 3.

Figure 6.3 represents the resulting time versus query table.

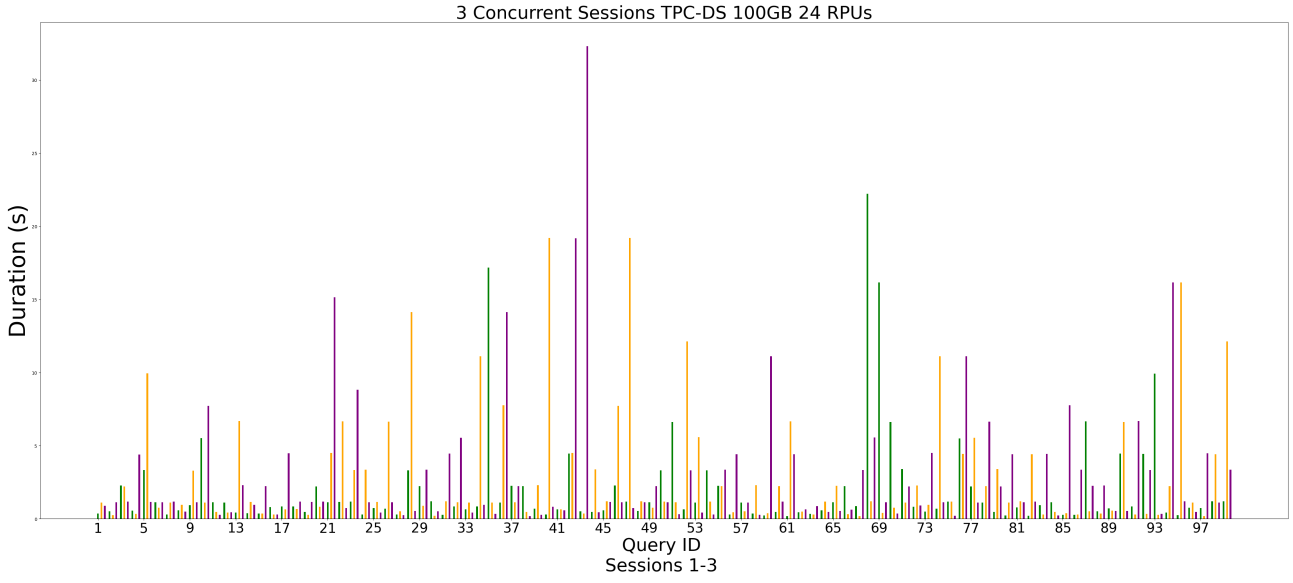

Figure 6.3: 100GB Benchmark Result

- We performed a test with the 1TB ADX TPC-DS dataset on an Amazon Redshift Serverless workgroup with 24 RPUs and with multiuser count of 3.

Figure 6.4 represents the resulting time versus query table.

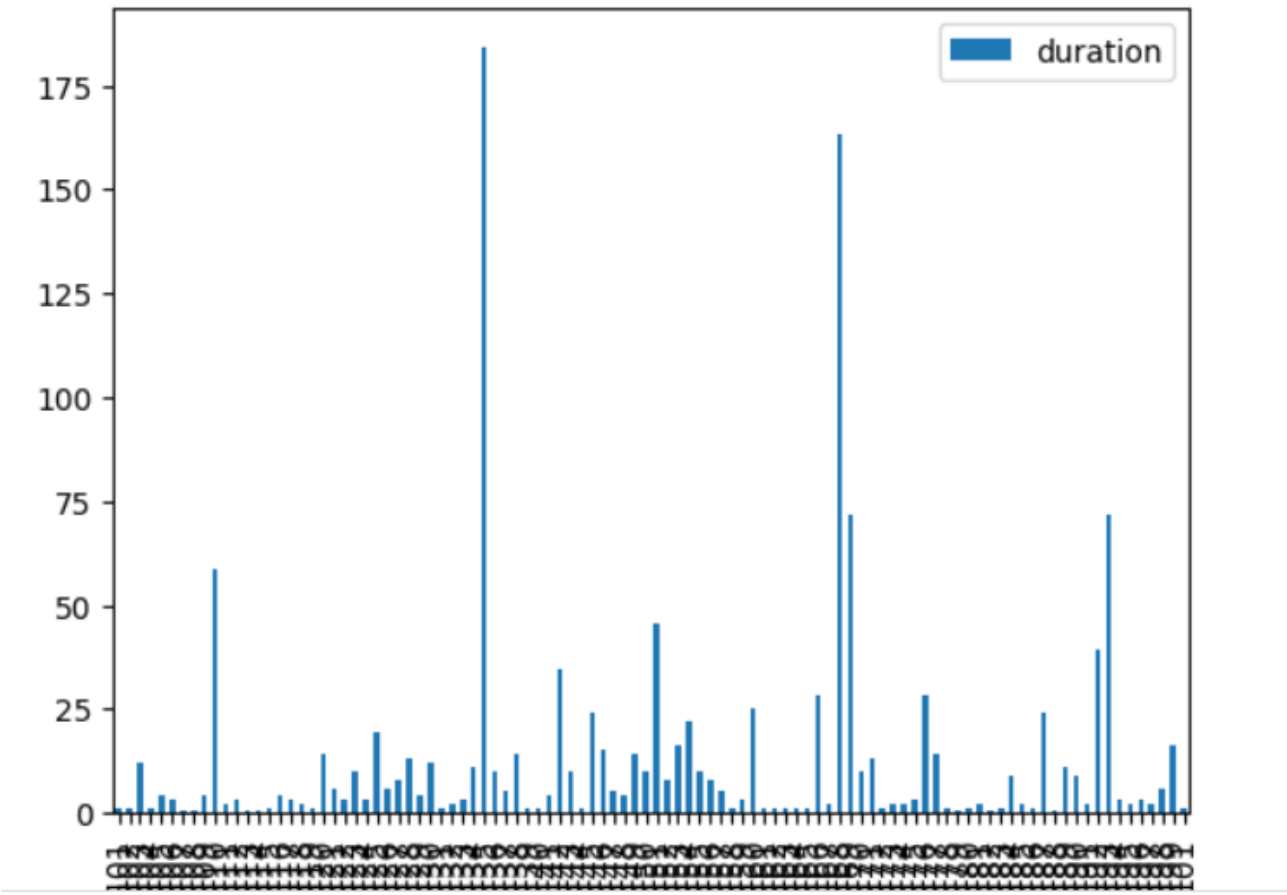

Figure 6.4: 1TB Benchmark Result

6.6 Evaluation at various scale factors

7 Performance Optimization

7.1 Query optimization techniques

7.2 Indexing strategies

7.3 Distribution keys and sort keys

7.4 Monitoring and fine-tuning

8 Replicability and Future Work

We will now focus on how can you replicate our results. We provide a `.zip` file that contains the code to execute all of the test.

8.1 Packaging code and instructions for replication

In the `.zip` attached, you will find *Shell* Files (`.sh`) to execute the different test, refer to the `README.md` to know exactly what files executes what test.

Beware you are supposed to have a AWS account, with the infrastructure mentioned in 4.3, and the generated data uploaded to S3

8.2 Future enhancements or optimizations

Ideally, we could have a single entry-point for performing the *TPC-DS* test, that runs the tests in an automatic manner, as stated by the specification. Also a longer `README.md` mentioning a more step-by-step tutorial would be useful to have in the future.

Some of the code is hardcoded to the scale factors we decided, another great improvement would be to generalize all of the code to run it in any scale factor efortlessly.

9 Conclusion

9.1 Recap of achievements and key takeaways

In this first part of our project, our primary objective was to evaluate *Amazon Redshift's* performance using the well-known *TPC-DS benchmark*. We mostly focused on **three critical areas** which are *data loading efficiency*, *data maintenance* and *the responsiveness* of the system to OLAP queries. Moreover, we tested the device independence of Redshift and provided a comparative analysis.

While working on the project, we always followed our pre-determined objectives and expectations. After tests and comparisons we have achieved and gathered valuable insights about the capabilities of *Amazon Redshift*.

First of all, in terms of data loading, *Amazon Redshift* showed remarkable efficiency in handling data loading tasks. Even after the tests with larger datasets, it had completed the data loading process in a timely manner. Similarly, the platform has succeeded in data maintenance. We saw that all data maintenance tasks like updates, inserts and deletes has been performed in an effective and smooth way. **Our test results** showed that it took *6 minutes 53.348 seconds* to executed **the load test for scale factor 1 (1GB)**, although it took *3 minutes 39.691 seconds* for **the scale factor 2 (10GB).** As we observed, the reason behind is that we're using *Redshift Serverless*, so the very first time we executed it had **to spin up the necessary machines** to make the system ready.

Secondly, the system responded to OLAP queries with high standard which means it executed queries quick and realiable responses both in single and multiuser scenarios.

Additionally, since *Amazon Redshift* is a **cloud-based solution** which offers portability and device independence it allows customers to have replication on different computing infrastructures and various devices without having any major compatibility issues.

Moreover, we provided a comparative analysis within *Amazon Redshift* to show how its performance changes as the volume of data increases. In the mean time, we compared *Amazon Redshift*

with other data warehousing solutions such as BigQuery as you can clearly see *in Chapter 2.3*.

In short, the key takeaway from this part of the project is that *Amazon Redshift* has proven to be a robust and capable solution for our decision support needs. It has all the skills and features any customer can need while handling data operations, responding complex queries and with its servesless, cloud-based infrastructure.

Main obstacles we faced:

- As we decided to use a **Cloud based** solution, we had to beware of the **costs** we may incur, as opposed to using a solution that could run on a local computer. This meant having a lesser flexibility when it came to run tests as every time we run an iteration, a cost was associated with it. We were able to leverage the **free tier** from *AWS*, along side with 300$ free credits for *Redshift Serverless* (of which we consumed 50$), which allowed us to vastly reduce our costs. Our estimated final cost for performing this project is *5$*.

- **Vagueness of the TPC-DS report**. Some parts of the report are not crystal-clear and several reads of the document were needed to understand the requirements of the benchmark

- Unfamiliarity with *AWS* ecosystem: Only one of our members had previous exposure to the *AWS* cloud, which made the initial stage of the project more cumbersome.

- Having to **debug errors** from the provided *TPC-DS* code, as for example the `tpcds_ri.sql` file with the *Referential Integrities* contained typos, or some of the syntax of the `VIEW`s was not supported by *Redshift* and we had to update it, such as changing `SUBSTR` by `SUBSTRING`.

9.2 Value added in this project

The main points where we believe have added value with this project are:

- We provided an overview of *Amazon Redshift* in the context of the lectures of *Data Warehouses* and partially *Advanced Databases*

- Even when we used a tutorial which eased the execution of the **Power** and **Throughput** tests, it helped us to showcase how easy is to get started with *Redshift* and how it does effortlessly supports Terabytes of data. Additionally, we implemented from scratch both the **Load** and **Data Maintenance** tests, which allowed us to demonstrate how straightforward is it to work with this cloud service, as well as the great integration that *Redshift* has with other *AWS* services,

9.3 Importance of benchmarking and Amazon Redshift in decision support

Benchmarking is a critical step when selecting the right data warehousing solution depending our decision support need. Through benchmarking, we could understand how *Amazon Redshift* fits our requirements for the need and how effective it would be if we choose to work with this specific data warehousing solution rather than many others.

AWS's innovative solution, as we demonstrated in our project, has a crucial role in the decision support space. It is an ideal choice for the customers and organizations seeking to make data-driven decisions since it has high performance and demanding features in terms of **scalability**, **flexibility** and **cost-effectiveness**. Furthermore, its **serverless environment** and **cloud-based nature** is a big advantage to reduce the complexities within infrastructure management for customers.

9.4 Looking ahead to Part II of the project on TPC-DI benchmark

In this first part of the project, we have successfully tested and demonstrated *Amazon Redshift* using *the TPC-DS benchmark*. Looking forward, in the next part of the project, we will focus on our next benchmark, *TPC-DI (Decision Support for Integrated Systems)*. This time, we will evaluate the system in the context of integrated systems with the objectives specified by this new famous benchmark.

The TPC-DI benchmark will help us understanding *Amazon Redshift* functions by focusing on relevant key performance metrics and indicators which would expand our understanding of the system's capabilities in decision support for integrated systems. As we move into the second part of our project, we will engage in new insights and further discoveries that will contribute to our strategic decision-making process.

Bibliography

[TPC21] (TPC), Transaction Processing Performance Council (June 2021). "TPC Benchmark™ DS - Standard Specification, Version 3.2.0". In.

[Arm+22] Armenatzoglou, Nikos et al. (2022). "Amazon Redshift re-invented". In: *SIGMOD/PODS 2022*. URL: https://www.amazon.science/publications/amazon-redshift-re-invented.

[Bor+20] Borić, Nemanja et al. (2020). "Unified spatial analytics from heterogeneous sources with Amazon Redshift". In: *SIGMOD/PODS 2020*. URL: https://www.amazon.science/publications/unified-spatial-analytics-from-heterogeneous-sources-with-amazon-redshift.

[Cho] Chow, Yoong Shin (n.d.). *A Comparison of Spatial Functions: PostGIS, Athena, PrestoDB, BigQuery vs RedShift*. https://ual.sg/post/2020/07/03/a-comparison-of-spatial-functions-postgis-athena-prestodb-bigquery-vs-redshift/.

[GWA22] Gromoll, Stefan, Florian Wende, and Ravi Animi (Apr. 2022). *Amazon Redshift continues its price-performance leadership*. https://aws.amazon.com/blogs/big-data/amazon-redshift-continues-its-price-performance-leadership/. Accessed: October 10, 2023.

[Hun] Hunt, Randal (n.d.). *Fact or Fiction: Google BigQuery Outperforms Amazon Redshift as an Enterprise Data Warehouse?* https://aws.amazon.com/tr/blogs/big-data/fact-or-fiction-google-big-query-outperforms-amazon-redshift-as-an-enterprise-data-warehouse/. October 26, 2016.

[Law] Law, Marcus (n.d.). *Biggest cloud providers*. https://technologymagazine.com/top10/top-10-biggest-cloud-providers-in-the-world-in-2023.

[McK] McKnight, William (n.d.). *Cloud Data Warehouse Performance Testing*. https://gigaom.com/report/cloud-data-warehouse-performance-testing/. April 9, 2019.

[NP06] Nambiar, Raghu and Meikel Poess (Jan. 2006). "The Making of TPC-DS". In: pp. 1049–1058.

[Rob23] Roberts, Jon (Jan. 2023). *Run a popular benchmark on Amazon Redshift Serverless easily with AWS Data Exchange*. https://aws.amazon.com/blogs/big-data/run-a-popular-benchmark-on-amazon-redshift-serverless-easily-with-aws-data-exchange/. Accessed: October 10, 2023.

[Sera] Services, Amazon Web (n.d.[a]). *Amazon Redshift Pricing page*. https://aws.amazon.com/redshift/pricing/. Accessed: October 30, 2023.

[Serb] — (n.d.[b]). *Amazon Redshift Serverless*. https://aws.amazon.com/redshift/redshift-serverless/. Accessed on October 29, 2023.

[Serc] — (n.d.[c]). *Amazon Redshift Serverless published*. https://aws.amazon.com/about-aws/whats-new/2022/07/amazon-redshift-serverless-generally-available/.

[Serd] — (n.d.[d]). *Columnar storage*. https://docs.aws.amazon.com/redshift/latest/dg/c_columnar_storage_disk_mem_mgmnt.html.

[Sere] — (n.d.[e]). *Creating materialized views in Amazon Redshift*. https://docs.aws.amazon.com/redshift/latest/dg/materialized-view-overview.html.

[Serf] — (n.d.[f]). *Ingesting and querying semistructured data in Amazon Redshift*. https://docs.aws.amazon.com/redshift/latest/dg/super-overview.html.

[Serg] — (n.d.[g]). *Querying data with federated queries in Amazon Redshift*. https://docs.aws.amazon.com/redshift/latest/dg/federated-overview.html.

[Serh] — (n.d.[h]). *Querying spatial data in Amazon Redshift*. https://docs.aws.amazon.com/redshift/latest/dg/geospatial-overview.html.

[Seri] — (n.d.[i]). *Using machine learning in Amazon Redshift*. https://docs.aws.amazon.com/redshift/latest/dg/machine_learning.html.