Implementation of TPC-DS Benchmark on Microsoft Azure SQL Database

Project Report

Rodaina Mohamed, Karim Maatouk, Yi Chiau Li, Haftamu Hailu Tefera

INFO-H-419 Data Warehouses

Professor Esteban Zimanyi

Fall 2019

Table of Contents

Abstract

This report presents the implementation of TPC-DS benchmark on Microsoft Azure SQL database. The implemented benchmark tests the performance of the respective database management system by running selected queries. Tests are conducted on four different scales factors: 1GB, 2GB, 5GB, and 100 GB and the respective queries runtime is recorded.

Key Words: TPC-DS, Azure SQL Database, Benchmarks, Data Warehouses, Decision support, DBMS

Introduction

One of the crucial issues in designing data warehouses is performance evaluations. Decisions such as architectural choices and optimizations techniques depend on evaluating the performance of the designated data. Nowadays, there are dozens of options of available technologies claiming to have efficient solutions for data analytics. Companies require ways to find simple, efficient, and cost-effective ways to handle their vastly growing data. Data warehouses (DW) benchmarks are technical tests run on data to access performance, scalability, and functionality. These assessments; however, are heavily dependent on the domain of the applications and the analysis objectives chosen for decision support.

One of the mostly used, easy, fast, and cost-effective database management systems is Microsoft Azure SQL database. It is a general multi-purpose relational database which is based on Microsoft SQL database engine. Our project focuses on accessing the speed and the efficiency of running SQL queries on Azure. This assessment is performed using TPC-DS which is a powerful measure that provides insight on the performance of database management systems with different data loads (scale factors).

The implementation of a TPC-DS benchmark is done using a sample generated scaled data on Microsoft Azure SQL database. A benchmark consists of 99 queries that are run to evaluate the performance of the technology. The data was loaded onto the databases management system (DBMS) over which TPC-DS queries were run on different scale factors to measure the running time of the queries.

The first section introduces the main tools and technologies used in conducting this research such as Microsoft Azure SQL Databases and TPC-DS. Secondly, the implementation section focuses on the implementation procedures and details to give the reader an understanding of how this project was implemented. Finally, the third section discusses and analyzes the results of running the TPC-DS queries on different scale factors of data .

1 Technologies

This chapter gives an understanding of the tools and technologies utilized in this project.

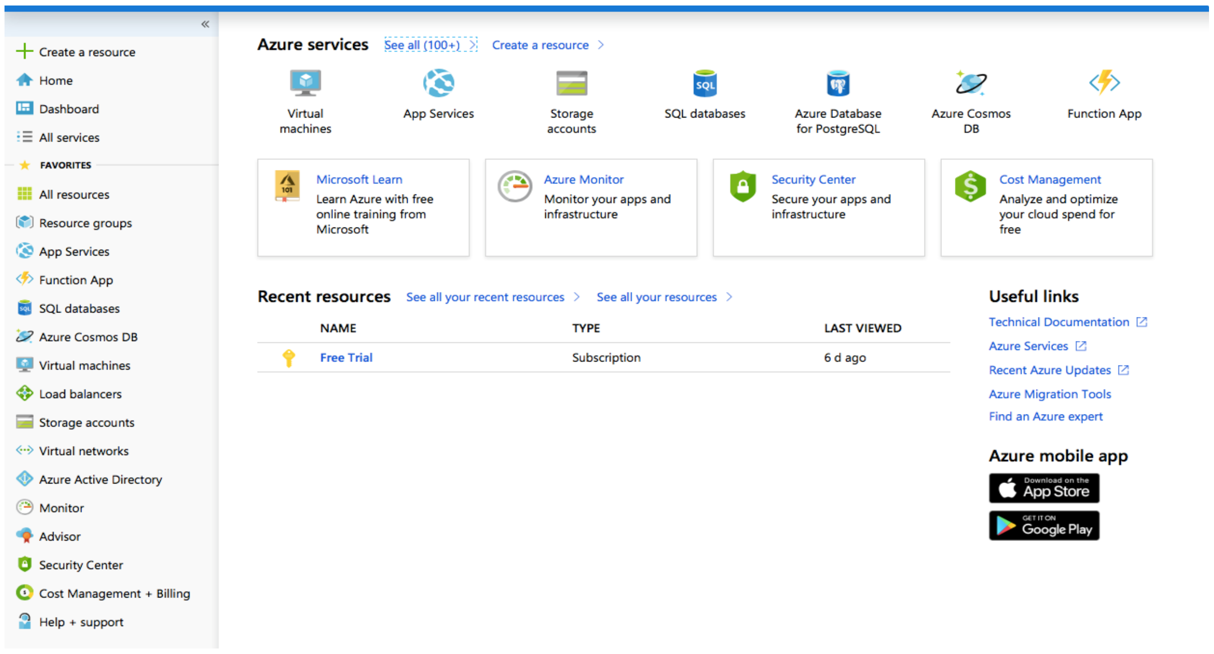
1.1 Microsoft Azure

Azure is a cloud computing multi-services created by Microsoft in 2010. It widely used for building, testing, deploying, and managing a variety of applications and services through Microsoft managed data center. Additionally, Azure supports many programming languages such as Python, Java, JavaScript,.. etc.

Among this collection of services, this project was practically focused on the data management services. One of those is Azure SQL database which is a general-purpose relational database, provided as a managed service. It provides high availability and high -performance data storage layer. Additionally, it can process unstructured non-relational databases such as graph, XML, spatial, and JSON.

Azure is based on the latest version of Microsoft SQL server database engine. It provides three different options of deployment for the database, single database, Managed instance, and Elastic pool. Single database is a fully managed, isolated database that can be used for modern cloud applications that needs single reliable source. Managed instance is a fully managed instance that contains a set of databases that are used together. Whereas Elastic pool is a collection of single databased with shared collection of resources.

On the other hand, Azure SQL database provides extensive monitoring and alerting capabilities that can help in troubleshooting and getting insights into the workload characteristics. Moreover, it provides high level of security as well as compliance features. These features includes advance threat protection features, Auditing for compliance and security, Data encryption, ..etc. Managing and developing in Azure SQL database is done using a web-based application for managing all Azure services called The Azure portal (figure 1).



Microsoft Learn: Learn Azure with free online training from Microsoft

Azure Monitor: Monitor your apps and infrastructure

Security Center: Secure your apps and infrastructure

Cost Management: Analyze and optimize your cloud spend for free

NAME	TYPE	LAST VIEWED
Free Trial	Subscription	6 d ago

Figure 1: Azure Portal

1.2 TPC-DS Benchmark

Benchmarking stands for the process of evaluating and measuring the performance of a tool from all aspects. By comparing those with standard or similar measurements of its peers, it can help organizations to comprehend as well as assess their current position in relation to best practice. In this case, the report benchmarks Microsoft Azure SQL Database using TPC-DS.

Decision support systems are planning and scheduling tools that utilize an organization's data warehouses and other data flows to generate inputs to management decision-making. The main purpose of decision support systems is assisting organization's decision-making. In this way, organizations can not only lower the cost but augment their performance as well. As a result, measuring decision support systems in organization is a crucial issue, and it is essential to promise that the decision support systems in organizations are efficient and suitable for them. With regard to verifying them, using other technology to implement decision support benchmark will be the best choice.

2 Implementation

This section discusses in details of the implementation of the TPC-DS benchmark. Additionally, it gives an overview about the how the previously mentioned technologies were utilized.

2.1 Creating Azure SQL Database and Tables

The first step in the implementation is creating an Azure SQL Database. Azure gives the flexibility to choose amongst a wide variety of specifications of the database hardware depending on how intensive the I/O operations you are to perform on the database. For the purpose of this project we are choosing 50 DTUs service tier which represents the database throughput based on a blended measure of CPU, memory, reads, writes. The larger the DTUs are, the better is the performance power of the database. 50 DTUs represents a standard performance.

After creating the Database, the next step is to create the tables (25 tables) with the query provided in the TPC-DS toolkit, namely tpcds.sql. For automating running the queries, we are using SQLCMD tool which is a tool for SQL server. It is worthy to note that since Azure SQL Database uses SQL server, therefore, it is compatible with all the tools used for SQL server.

To run the query for creating database table, the following SQLCMD command is run from the terminal [Windows]:

```
sqlcmd.exe -S [server name].database.windows.net -d [Database name] -U [Server Username] -P [password] -i  [Query Directory\tpcds.sql]
```

This will successfully create the database tables marking the first step in the implementation of the TPC-DS benchmark.

2.2 Generating the data from the TPC-DS toolkit

The second step in the benchmark implementation is generating the test data from the toolkit. We will provide the details for generating the data on a windows machine since the IOS data generation is more straightforward. Before being able to generate the data, we will need to compile and run the data generator using Microsoft Visual Studio, namely we need to run the **dbgen** solution that can be found inside the tools folder of the toolkit. After compiling the solution in Visual Studio, it will produce a tool for data generation which is **dsdgen.** The following tool can be run from Windows terminal (CMD) to generate test data. The tool allows for specification of multiple options like:

- Scale factor
- Delimiter (column separator, default “|”)
- Row separator (enabled by default)

A sample command to generate data for 1GB with a "|" delimiter and no row separators is:

```
dsdgen /scale 1 /dir [output directory] /terminate n
```

When generating larger data sizes, it is recommended to use commands that run in parallel to make the data generation faster. A sample command to generate 100 GBs of data in 4 parallel processes:

```
start /b dsdgen /scale 1 /parallel 5 /child 1
start /b dsdgen /scale 1 /parallel 5 /child 2
start /b dsdgen /scale 1 /parallel 5 /child 3
start /b dsdgen /scale 1 /parallel 5 /child 4
```

For the purposes of this project, we are generating data on the following scales:

- 1 Gigabytes
- 2 Gigabytes
- 5 Gigabytes
- 100 Gigabytes

The result of the data generation is 25 data files (.dat) files representing data for 25 tables in the database.

2.3 Loading the data on Microsoft Azure

After generating the test data, we need to load it into the Azure SQL Database on the cloud. This involves three main challenges:

1. Choosing a compatible tool

   For this project, we used SQL server BCP tool which is a Bulk Copy Program Utility used to copy data between SQL server and a data file.

The BCP tool can be run as well from Windows terminal (CMD) using commands and can be configured to be compatible as it offers several options to specify:

- Batch size
- Column separators

Therefore, it helps in making the database defaults compatible with the data files defaults.

-A sample .bat script (dbloadscript.bat) can be found in the project to upload data of 25 tables and can be run from CMD or simply clicking the .bat file.

-A sample BCP command to read from file call_center.dat and write the data to "call_center" table on Azure SQL DB using "|" as a delimiter

```
bcp call_center in [File Directory\call_center.dat] -S [Server Name].database.windows.net -d [Database Name] -U [Server Username] -P [Password] -c -t"|"
```

2. Uploading huge data over internet
3. Limited Computation capacity

It is not a difficult task to load 1 Gigabytes , 2, or even 5 GBs of data into Azure SQL DB or any other cloud solution; however, when talking about larger scales , 100 GBs for instance, one can notice the need to address the two latter challenges. Two sample scenarios:

- Sudden Internet connection cut off or other interruption
- Computer freeze

The two above scenarios can happen on any standard computer with limited computational power; therefore, this motivates the need to a solution that is available and powerful enough to handle loading big data scales.

Therefore, for loading data into the cloud, we used Azure Virtual Machines which made the process faster, and uninterrupted, and allowed for a better team accessibility to the working desktop.

2.4 Validating Data Correctness

Validating data correctness after loading the test data into the cloud is essential. This step is to make sure that the data in Azure SQL Database matches the data we have in the generated data files.

Therefore, we select a measure that signifies the data correctness. For this we chose **Data rows count**. This measure gives an idea if the number of rows in our database matches the **expected row count** in the data files for the selected scale factor. This is a simple measure, however effective and not a costly operation even for large scale factors.

To accomplish this, we run similar query for every table in our database:

```
SELECT COUNT(*) FROM [TABLE_NAME];
```

Running on the 25 tables will generate 25 counts which we compare to the row counts in the generated files. This is done for every scale factor post loading the data and prior to running the test queries.

2.5 Azure Virtual Machines

Given the challenges introduced in the previous section, Azure Virtual machine is chosen to make the benchmarking more efficient through the following:

- Good Computational Power
- Availability
- Better team cooperation
- Fast Internet connection

Microsoft Azure allows choosing among a spectrum of powerful machines accommodated to your project’s needs. For this Project, the most valuable feature was the availability and fast internet connection of Azure Machines.

Creating a Virtual machine is a very straight forward process that would take few minutes after choosing the specifications, and the machine is deployed instantly. Connecting to the machine is straightforward as well through remote desktop.

For the purposes of this project we used two virtual machines.

- TPCDS-SMALL: A less powerful machine for smaller scales
- TPC-DS: A more powerful machine for larger data scales (100 GBS)

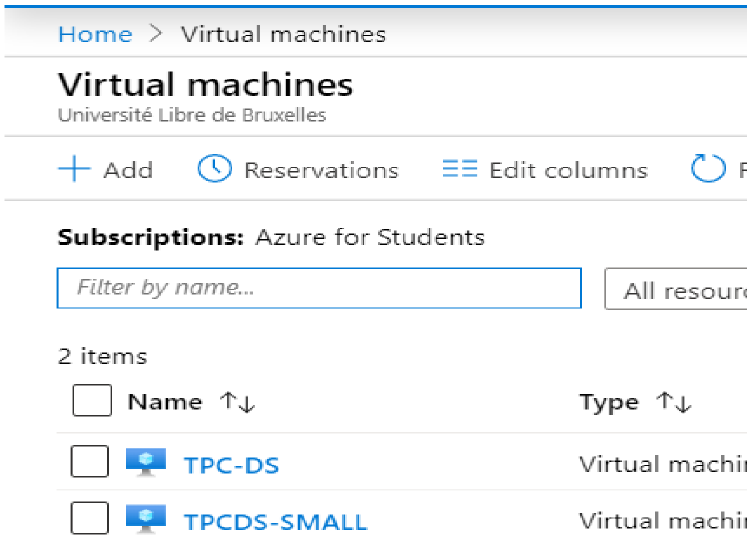


Figure 2: Virtual Machine on Azure Portal

2.6 Translating and running TPC-DS queries

The TPC-DS toolkit contains query templates which should be translated into actual sql queries to be able to be run on the database. The query templates .tpl files are generic and should be translated into the specific dialect that the DBMS is compatible with. Since Azure SQL Database works with Microsoft SQL server , the queries should be translated to SQL server dialect. The tool **dsqgen.exe** which is generated when compiling the dbgen2 solution on visual studio has the option of choosing SQL server through the following command:

```
dsqgen /VERBOSE Y /DIALECT sqlserver /DIRECTORY [Templates Directory] /SCALE [Scale] /OUTPUT_DIR ../[Output Directory] /INPUT ../[Templates Directory]/templates.lst /QUALIFY Y
```

The latter command translates queries into SQL server dialect and generates an SQL file with the 99 queries required to benchmark our DBMS.

However, the translation into specific dialect is not enough as most queries still need to undergo syntax changes to be compatible with SQL server, and thus Azure SQL DB.

* Check the Appendix for sample changes required. The changes are done to the templates (.tpl files) to avoid fixing queries for every data scale factor.

After adjusting queries syntax, the last step in implementing TPC-DS benchmark is to run the queries on the Database.

To do that, we use SQLCMD tool of SQL server to automate running the queries.

Sample SQLCMD command:

```
sqlcmd.exe -S [Server Name].database.windows.net -d [Database Name]  -U [Server Name] -P [Password] -i [Queries File Directory]-o [Output File Directory]
```

For recording queries time, we add to the queries file the following line:

```
SET STATISTICS TIME ON;
```

The result of the SQLCMD command above will be a result file with the run times of the 99 queries in a single file. After that, we apply a regular expression to extract the run time from the results file.

3 Results and Discussion

This chapter discusses the results of running the TPC-DS queries on 4 different data scales used in this experiment. Moreover, the effect of some added features to the generated data such as indexing or additional services on azure portal on the execution time is discussed. Section 3.1 and 3.2 suggest some potential changes in parameters that can lead to performance improvements.

During the execution, some inconsistences in the run time were found when the same query runs for several times due to caching. In order to overcome this issue, the average of the execution time was calculated for each query after running each one for several times. Expectedly, the execution time of most of queries linearly increased as the data scale increased. However, this observation doesn’t hold for query 22 which takes more time to run on 1GB data scale than 2GB and 5GB data scales. Graphs were created for all data scales with different parameters to visualize the performance of the queries execution time on Azure. Note that the 99 queries execution times were distributed on 4 graphs for better readability. Additionally, the data was plotted using logarithmic scale of base 2 in order to respond to skewness towards large values, i.e., cases in which few points have much larger values than most of the data(outliers). On the other hand, the execution time of queries from 1 to 58 were ran on 100GB data scale on 200 DTUs database throughput. The average and the median execution times for these queries are ~2339s and ~355s respectively. Figure 4 shows the execution time of these queries.

3.1 Execution Time with different DTUs

Azure offers calibration of the DTU (Database Throughput Unit) measure in order to get better performance. DTUs are some type of measure for the performance of a certain service tier. Consequently, it helps the user to choose the appropriate service tier. The percentage of DTU indicates the percentage of throughput units that the database is using. This experiment was conducted with different DTU measures in order to monitor the performance in each case. At the beginning, execution times were recorded using 20DTUs, so the system was upgraded to 50 DTUs which made a huge difference for some of the queries. On 20 DTUs, query 78 ran in ~235s while query 88 ran in ~25.5s. After upgrading both queries ran in almost half the old execution time. Charts 2.1 – 3.4 show the execution time results of TPC-DS queries on the 1GB, 2GB and 5GB data scales. Using 50 DTUs, the recorded average execution time of all the queries on the 1GB, 2GB, and 5GB data scale is 11s, 80.78s, and 217.79s respectively.

3.2 Execution Time after Indexing

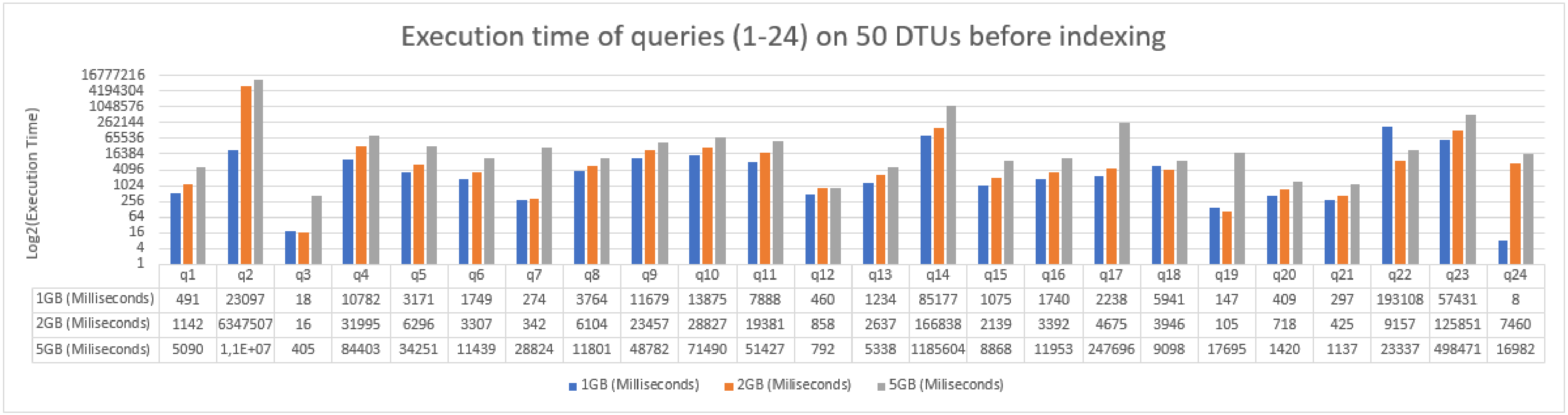
On azure portal, one of the features that are offered is performance recommendations. This feature suggests potential indexes that might improve the performance by analyzing the workload across the server. This index recommendation helps improve the performance of most of the time-consuming queries. In this part, the recommended indexes were added then the execution time was recorded for various data scales in order to monitor the performance after the indexing. According to the recommendation, indexes were applied to two data tables to investigate if there will be any performance improvement.

```
-- The script to apply the recommendation is the following:
CREATE  NONCLUSTERED  INDEX
[nci_wi_store_sales_52F4F1D7ADA97FB5C50DC8841961C99A]  ON  [dbo].[store_sales]
 ([ss_sold_time_sk]) INCLUDE ([ss_hdemo_sk], [ss_store_sk]) WITH (ONLINE = ON)
```

```
-- The script to apply the recommendation is the following:
CREATE  NON  CLUSTERED  INDEX
```

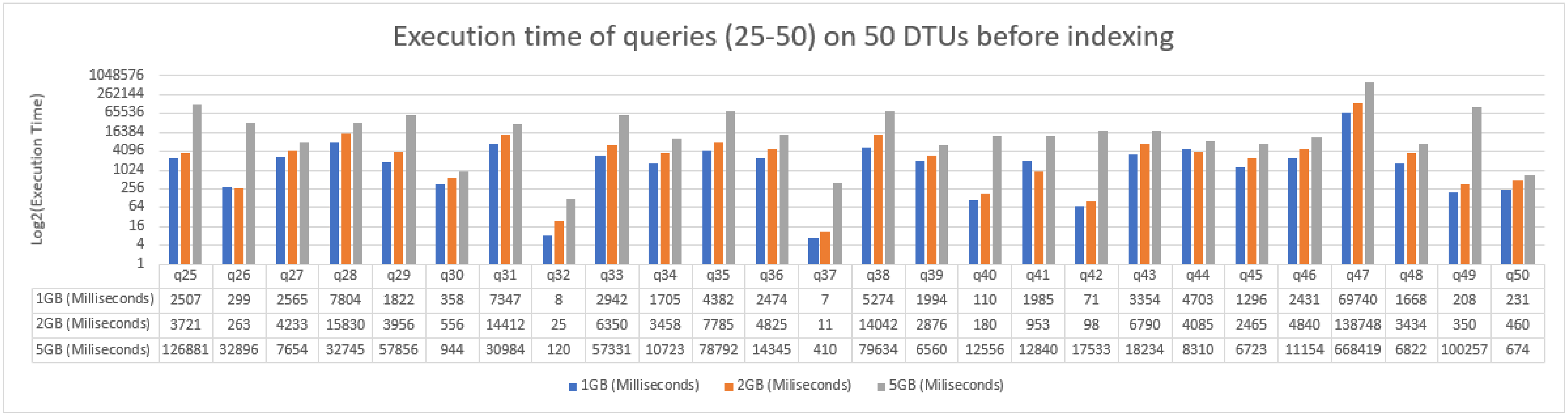
```
[nci_wi_web_sales_3C60B862EBCDAA99B616864C30B1140A]  ON  [dbo].[web_sales]
 ([ws_order_number]) INCLUDE ([ws_warehouse_sk]) WITH (ONLINE = ON)
```

After running the queries again on 2GB data scale with the indexed tables, execution times were recorded. It was found out that the indexing had very little effect on the execution time. The average execution time before indexing was 80.68s whereas the average execution time after indexing is 78.34s.



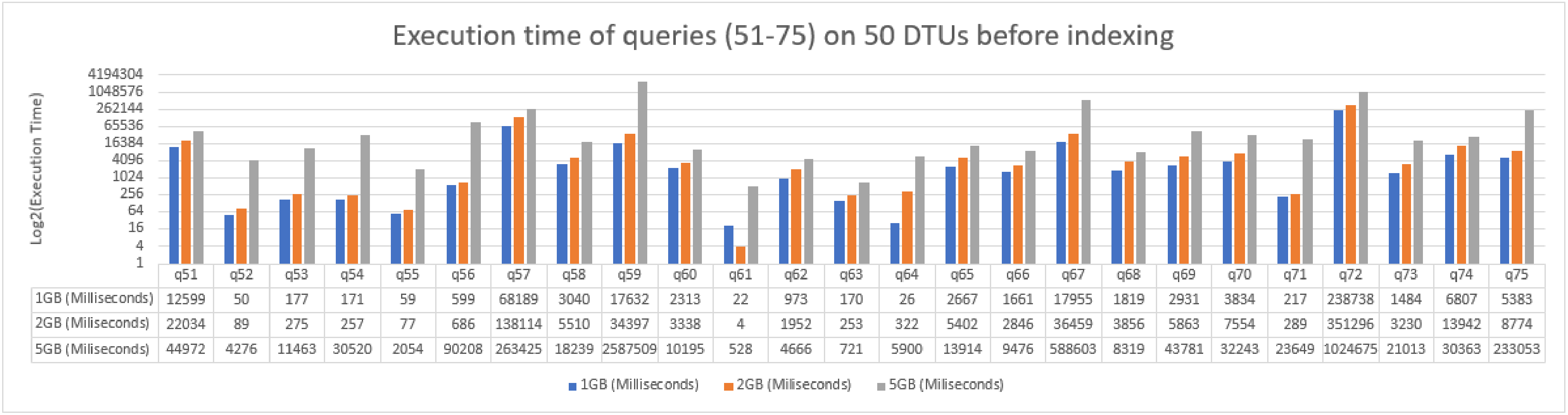
	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18	q19	q20	q21	q22	q23	q24
1GB (Milliseconds)	491	23097	18	10782	3171	1749	274	3764	11679	13875	7888	460	1234	85177	1075	1740	2238	5941	147	409	297	193108	57431	8
2GB (Miliseconds)	1142	6347507	16	31995	6296	3307	342	6104	23457	28827	19381	858	2637	166838	2139	3392	4675	3946	105	718	425	9157	125851	7460
5GB (Miliseconds)	5090	1,1E+07	405	84403	34251	11439	28824	11801	48782	71490	51427	792	5338	1185604	8868	11953	247696	9098	17695	1420	1137	23337	498471	16982

Figure *3.1: Histogram presents the execution time of the TPC-DS queries 1-24 for the 1GB, 2GB, and 5GB data scales*



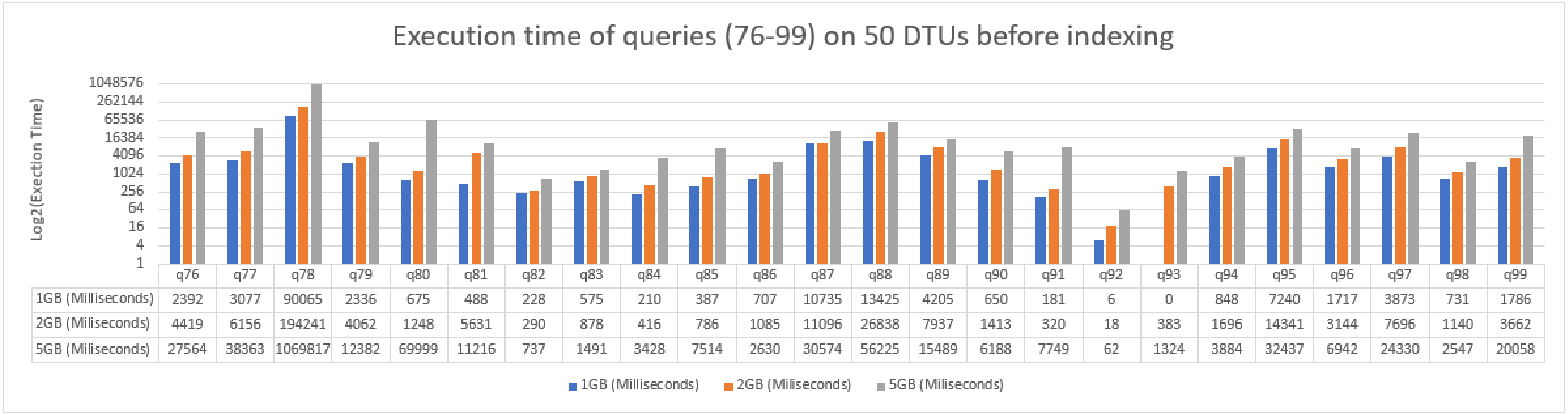
	q25	q26	q27	q28	q29	q30	q31	q32	q33	q34	q35	q36	q37	q38	q39	q40	q41	q42	q43	q44	q45	q46	q47	q48	q49	q50
1GB (Milliseconds)	2507	299	2565	7804	1822	358	7347	8	2942	1705	4382	2474	7	5274	1994	110	1985	71	3354	4703	1296	2431	69740	1668	208	231
2GB (Miliseconds)	3721	263	4233	15830	3956	556	14412	25	6350	3458	7785	4825	11	14042	2876	180	953	98	6790	4085	2465	4840	138748	3434	350	460
5GB (Miliseconds)	126881	32896	7654	32745	57856	944	30984	120	57331	10723	78792	14345	410	79634	6560	12556	12840	17533	18234	8310	6723	11154	668419	6822	100257	674

Figure 3.2: Histogram presents the execution time of the TPC-DS queries 34-66 for the 1GB, 2GB, and 5GB data scales



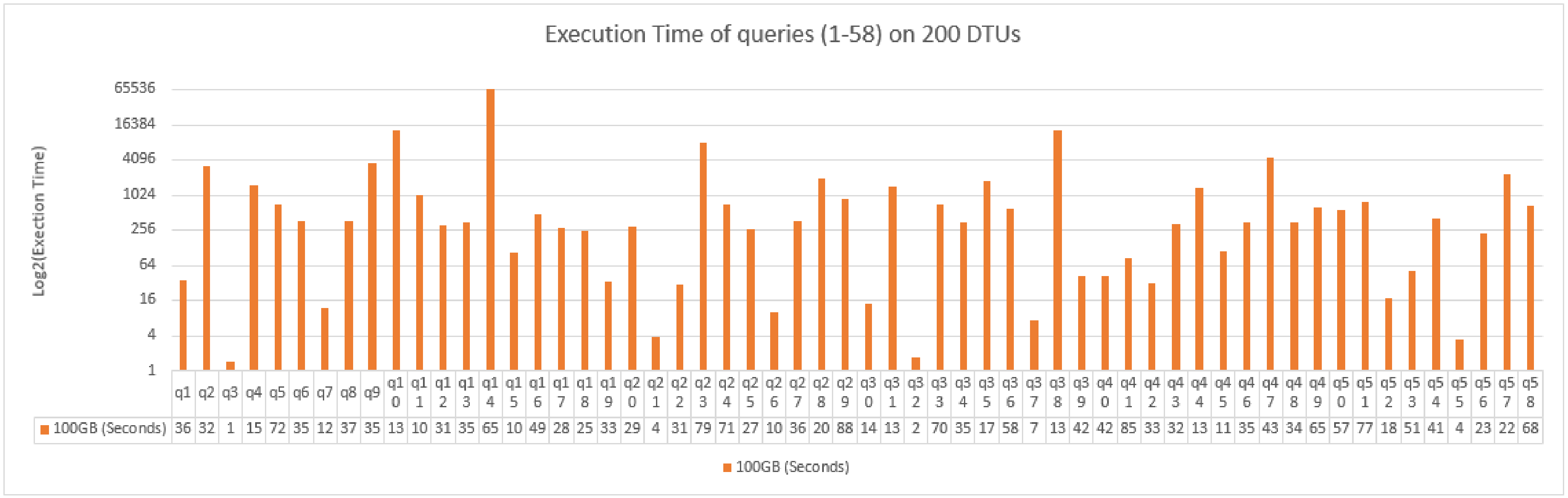
	q51	q52	q53	q54	q55	q56	q57	q58	q59	q60	q61	q62	q63	q64	q65	q66	q67	q68	q69	q70	q71	q72	q73	q74	q75
1GB (Milliseconds)	12599	50	177	171	59	599	68189	3040	17632	2313	22	973	170	26	2667	1661	17955	1819	2931	3834	217	238738	1484	6807	5383
2GB (Miliseconds)	22034	89	275	257	77	686	138114	5510	34397	3338	4	1952	253	322	5402	2846	36459	3856	5863	7554	289	351296	3230	13942	8774
5GB (Miliseconds)	44972	4276	11463	30520	2054	90208	263425	18239	2587509	10195	528	4666	721	5900	13914	9476	588603	8319	43781	32243	23649	1024675	21013	30363	233053

Figure 3.3: Histogram presents the execution time of TPC-DS queries 51-75 for 1GB, 2GB, and 5GB data scales



	q76	q77	q78	q79	q80	q81	q82	q83	q84	q85	q86	q87	q88	q89	q90	q91	q92	q93	q94	q95	q96	q97	q98	q99
1GB (Milliseconds)	2392	3077	90065	2336	675	488	228	575	210	387	707	10735	13425	4205	650	181	6	0	848	7240	1717	3873	731	1786
2GB (Miliseconds)	4419	6156	194241	4062	1248	5631	290	878	416	786	1085	11096	26838	7937	1413	320	18	383	1696	14341	3144	7696	1140	3662
5GB (Miliseconds)	27564	38363	1069817	12382	69999	11216	737	1491	3428	7514	2630	30574	56225	15489	6188	7749	62	1324	3884	32437	6942	24330	2547	20058

Figure 3.4: Histogram presents the execution time of TPC-DS queries 76-99 for 1GB, 2GB, and 5GB data scales



	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18	q19	q20
■ 100GB (Seconds)	36	32	1	15	72	35	12	37	35	13	10	31	35	65	10	49	28	25	33	29

	q21	q22	q23	q24	q25	q26	q27	q28	q29	q30	q31	q32	q33	q34	q35	q36	q37	q38	q39
■ 100GB (Seconds)	4	31	79	71	27	10	36	20	88	14	13	2	70	35	17	58	7	13	42

	q40	q41	q42	q43	q44	q45	q46	q47	q48	q49	q50	q51	q52	q53	q54	q55	q56	q57	q58
■ 100GB (Seconds)	42	85	33	32	13	11	35	43	34	65	57	77	18	51	41	4	23	22	68

Figure 4: Histogram presents the execution time of TPC-DS queries 1-58 on 100GB data scale

Conclusion

This report presents in detail the implementation of TPC-DS benchmark on Microsoft Azure. The first chapter introduces the used technologies throughout the project. The second chapter explains in detailed steps of the implementations procedures that would allow the user to replicate the project if needed. The last chapter shows and discusses the results of the attempted experiment. Furthermore, the project also explores some features on Azure portal that have some potential performance improvements.

On the other hand, the execution time of queries on several data scales (1GB, 2GB, 5GB, 100GB) were analyzed and compared in order to explore the ability of Microsoft Azure SQL database in benchmarking. Further work could be done by comparing the execution time of queries on Microsoft Azure SQL database to other database Management systems.

References

1. Stevestein. (2018, August 4). What is the Azure SQL Database service?
2. TPC-DS - Homepage. (n.d.). Retrieved from http://www.tpc.org/tpcds/ .

   Retrieved from https://docs.microsoft.com/en-us/azure/sql-database/sql-database-technical-overview
3. Stevestein. (2019, June 9). Azure SQL Database service tiers - DTU-based purchase model. Retrieved from https://docs.microsoft.com/en-us/azure/sql-database/sql-database-service-tiers-dtu .
4. Danimir. (2018, December 19). Performance recommendations - Azure SQL Database. Retrieved from https://docs.microsoft.com/en-us/azure/sql-database/sql-database-advisor.

Appendix

Following is an Example of some changes we applied to the query templates:

Original Syntax	Corrected Syntax	SQL Server
d_date between cast('2000-08-19' as date) and (cast('2000-08-19' as date) + 14 days)	d_date between cast('2000-08-19' as date) and (DATEADD(day,14,'2000-08-19'))	DATEADD
‘catalog_page' || cp_catalog_page_id as id	‘catalog_page' | cp_catalog_page_id as id	|
'web_site' || web_site_id as id	'web_site' | web_site_id as id	|
'store' || s_store_id as id	'store' | s_store_id as id	|

Query 5 template changes