Database Benchmark Based on User Queries

Lim, Tian Xing Eddy

Award date:
2022

Department of Mathematics and Computer Science
Architecture of Information Systems Research Group

Database Benchmark Based on User Queries

Master Thesis

Tian Xing Eddy Lim

Supervisors:
dr. O. Papapetrou
dr. E. Ioannou

Assessment Committee:
dr. O. Papapetrou
dr. E. Ioannou
dr. R.M. de Carvalho

Eindhoven, Wednesday 10th August, 2022

Abstract

This work introduces a benchmarking tool that creates a new benchmark using a synthetic workload that has similar properties as the original. By preserving essential properties of the original workload, a synthetic workload could be generated having these similar properties. The properties that were identified as essential were of the data, queries and benchmark execution. Using these properties, other synthetic workloads could be created similar to the original. These essential properties were captured from a dataset within a specific use case, based on which synthetic data and queries were generated. This work also discusses how to measure the performance of the generated synthetic data and queries. The process of generating and validating the workloads comprises different scripts that could be applied in other use cases. By reusing this implementation, the synthetic workloads could be published similar to the original but do not contain sensitive information.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Nowadays, much data is being collected for research and by different companies to get an insight into their sales or use the data for their customers. This data is usually stored in database management systems (DBMS), and researchers and companies use many different DBMSs for storing their data. There has always been an interest in which DBMS has better performance, i.e. faster [1]. In order to compare the performance of different DBMSs, a benchmark could be used, and based on the results an informed comparison could be made. A benchmark consists of data and queries, called the workload, and based on the execution time of the queries on the data, DBMSs could be compared.

Different types of domain-specific benchmarks already exist, such that the DBMS could be tested for a particular setting. The first problem with these benchmarks is the execution details that are not given, e.g., the order of the query or how often a query is executed. The frequency of a specific query and the order of the query execution could be a problem because a DBMS could adapt to a query when run multiple times [2].

The second problem is related to the workloads of these domain-specific benchmarks. An example of such a benchmark is the TPC-DS benchmark [3]. However, there are domains for which no workloads exist, meaning there are also no benchmarks for these domains. Nevertheless, there are companies that have data for those domains but companies do not want to share their data and query logs because they could contain confidential information [4].

In order to tackle the two mentioned problems above, a benchmark was created by preserving essential features of the data and queries. The created benchmark contains a synthetic workload based on a real-world workload and could therefore be used for more domains. While creating the benchmark, the essential properties of the data and queries were introduced and discussed how to preserve them. One of our contributions is a benchmarking tool that could be reused to extract essential properties from an existing workload. Companies could use this tool to create a synthetic workload and share this workload with others without privacy issues.

1.1 Research Problem

A new benchmarking tool was created in this work consisting of a synthetic workload based on a real dataset. An important aspect is that the synthetic workload should look like the original workload. An option is to use essential features of the original workload and use these features, which led to the first part of the research question:

What are the essential properties of a workload?

After identifying the essential properties of a workload, the next step is extracting these features from the workload, which is the second part of the research question:

How could the essential properties be extracted from an existing workload?

Once the essential properties are identified and extracted, a synthetic workload could be created. Creating a synthetic workload using the essential properties leads to the third part of the research question:

How could the essential properties be used to create a synthetic workload?

Using these properties, a synthetic workload can have similar features and look like the original workload. However, evaluating the similarity between the workloads leads to the last part of the research question:

How could the similarities between the original and synthetic workload be evaluated?

By evaluating both workloads, the focus is on the performance of the workloads. Therefore, the workloads can be tested holistically by executing a benchmark focusing on the execution times.

1.2 Contributions

The main contributions of this thesis are:

- Identified properties of a dataset and queries that influence the execution performance of a database and should therefore be included in a benchmark.
- Creating a benchmarking tool that preserves essential properties of a workload. Using this tool, a synthetic workload could be created that looks like the original by with similar properties.
- Evaluated the benchmarking tool by creating a benchmark using reverse engineering since the workload is what is tested and not the performance of the database. Using a use case for which a dataset and queries are downloaded, the benchmarking tool created the benchmark.
- The benchmarking tool can be used by others such that companies, for example, could share synthetic workloads [4].

1.3 Outline

The remainder of this thesis is organized as follows. Chapter 3 introduces the workload's essential properties and is divided into three categories: the data, queries, and the execution of the benchmark. These essential properties of a workload could be implemented and are explained by showing a data workflow in Chapter 4. Chapter 5 gives a use case by introducing the data and queries used in this work. Chapter 6 discusses the experiments, which are the benchmark executions, and shows how the essential properties influence the execution times. Finally, Chapter 7 concludes this work by discussing known limitations and suggestions for future work.

Chapter 2

Related Work

This chapter discusses existing publications related to the creation of a new benchmark using a workload based on user data and queries. Two well-known benchmarks will be discussed and how these benchmarks are different. Benchmarks that were created in other publications will be discussed as well and how these are different from the new benchmark presented in this work.

The organization renowned for its different benchmarks is the Transaction Processing Performance Council (TPC). TPC is a non-profit organization whose main objective is to establish criteria for obtaining information about the performance of databases using benchmarks [1]. Two TPC benchmarks are the TPC-H and TPC-DS benchmarks used to compare databases but using different scenarios. TPC-DS is the predecessor of TPC-H. The TPC-H and TPC-DS benchmark use a specific scenario to compare databases. The TPC-H benchmark models a static decision support system database environment [1]. The TPC-DS benchmark models a decision support system in a retail product supplier and was improved by taking the best features of the TPC-H benchmark [1, 5].

An example is the schema used the TPC-H benchmark follows a 3rd Normal Form (3NF) schema, which consists of eight tables, while nowadays, different variants of star schemas are used [3]. The TPC-DS benchmark was improved by using a snow-flake schema, a hybrid schema, because the shift from 3NF to star schemas was taking place while making this benchmark. Even though these benchmarks were frequently used to test the performance of databases, these benchmarks do not have the execution details mentioned in Chapter 1. The number of queries executed is for both benchmarks given but not how often a query is executed or in which order.

There have been other research in which a new benchmark has been developed based on user queries and data; however, these are different in some aspects. Such a benchmark is the `public bi benchmark` what is a follow-up of previous research in which it is also stated that current benchmarks do not represent the real-world [6]. Their benchmark consists of 46 workbooks and has queries consisting of joins, these workbooks are tableau workbooks created using Tableau Public [7]. However, for the benchmark created in this work, the focus lies on queries with aggregations and joins. `Public bi benchmark` also has queries with joins; however, these are only self-joins, and their queries are machine-generated and not created by users [6].

Deep et al. [8] also mentions that benchmark workloads are either standardized or manually curated. However, the latter is difficult to create for every use case. In their work, the emphasis is also partly on synthetic workloads and that these synthetic workloads should represent the original workload. The essential features also address the original workload because when these essential properties are captured, the synthetic workload will have high coverage and high representation. However, creating a synthetic workload requires that the query log be available through the database management system (DBMS). Using a query log directly from a DBMS is a problem when creating a synthetic workload based on a real-world query log, since companies are unwilling to share their query logs.

Different real-world datasets were used to create a benchmark in Battle et al. [9]. The benchmark created in their work also consists of generated queries and mentions that current benchmarks do not correctly represent how queries are generated. However, the focus is heavily on visualization, specifically cross-filters, since these cross-filters are intuitive and the most demanding use case for a DBMS.

Chapter 3

Preserving Properties

In this work a primary goal is holistically testing the benchmark as mentioned in Section 1.1, which means the focus is on the performance. This performance was measured by focusing on the execution times of the benchmark. Therefore, the interest is mainly in the difference in execution times between the original and synthetic workload, where the synthetic workload is based on the original. By executing a benchmark, the execution times of both workloads could be compared, and the goal is to minimize the difference in execution time. Minimizing this difference is possible when the original and synthetic workloads have a similar behavior when executed. In this Chapter, essential properties are discussed that could be used to make a synthetic workload look like the original. The essential properties could be divided into three categories, discussed in the following Sections. The three categories are the essential properties of the data, the essential properties of the queries, and the essential properties for the benchmark execution.

3.1 Data

The synthetic workload should look like the original, meaning the synthetic and original data have similar properties. The synthetic data could look like the original using these properties, meaning similar results are returned. To get similar results, properties such as correlations and cross-relationships are essential. In this work, the original data consists of different tables and attributes that should be present in the synthetic data. The following paragraphs provide the essential properties of the data.

Correlations. The first essential property is data correlations that could make the synthetic look like the original. Correlations in the data are not only correlations of attributes between the tables but also between attributes inside the same table. When these two are correlated, two attributes are related, which means an attribute could be predicted using another attribute. The correlation coefficient value is not as crucial as knowing there is a correlation because the relation between the attributes could be complicated. Therefore knowing two variables are related is of more value than knowing the coefficient.

Cross-relationships. The second essential property partly related to correlations is the cross-relationships in the tables. Using cross-relationships, connections between tables could be identified using a joining key of the table. These joining keys are specified in tables and should be present in the synthetic data. A specific number of rows could be returned from the data based on cross-relationships. The synthetic data should produce a similar number of rows to state that the data is similar to each other. Cross-relationships could also indicate functional dependencies is in the data; using functional dependencies, an attribute from a table could be determined by another high probability attribute [10].

Table 3.1 is a simple example of data that contains correlations between the attributes `Age` and `Salary`. These correlations between the attributes should also be present in the synthetic data. Between Table 3.1 and Table 3.2, there are cross-relationships, and these two tables could be joined with each other using the attribute `Works_At` from Table 3.1 and attribute `Company_Name` of Table 3.2.

Name	Age	Salary	Works_At
Timon	25	2500	University B
Zazu	50	4600	University C
Pumba	25	3000	University A
Rafiki	50	5000	University C

Table 3.1: Example data in which correlations could be seen between the attributes `Age`, `Salary`, and `Hometown`.

Company_Name	Location
University A	Amsterdam
University B	Rotterdam
University C	Eindhoven

Table 3.2: Example data on fake universities and have cross-relationships with Table 3.1.

3.2 Queries

Section 3.1 discussed one part of the workload: the data of the workload. The other part is the queries, and as with the data, the queries should also have the same properties. These properties of the queries are essential such that a similar number of rows are returned once the synthetic and original queries are executed. Two properties of a query could influence the returned results, which are the operators and clauses used in a query. The following paragraphs provide the essential properties of the queries.

Query Operators. An operator could be used to filter the data based on certain conditions which could influence the returned results. Different operators could be used in a query, such as arithmetic, comparison, and logical operators. Using a combination of these query operators could be created to answer specific questions based on the returned result.

Query Clauses. Another essential property is the clauses used in a query, the operators mentioned above could be used in the `WHERE`-clause of a query. Other clauses that are important in the query are the `SELECT`- and `JOIN`-clauses.

Listing 3.1 and Listing 3.2 contain example queries that use Table 3.1 and Table 3.2. In these listings, the query operators could be seen in the `WHERE`-clauses of line 4. Both listings also contain `SELECT` and `JOIN` query clauses, essential for the synthetic queries.

```
1  SELECT t1.Name, t2.Location
2  FROM Table 3.1 as t1
3  JOIN Table 3.2 as t2 on t1.Works_at = t2.Company_Name
4  WHERE t2.Salary < 4000
```

Listing 3.1: Example query that uses Table 3.1 and Table 3.2

```
1  SELECT t2.Company_Name, t1.Age
2  FROM Table 3.1 as t1
3  JOIN Table 3.2 as t2 on t1.Works_at = t2.Company_Name
4  WHERE t1.Age < 30 AND t2.Location NOT LIKE 'Eindhoven'
```

Listing 3.2: Another example query that uses Table 3.1 and Table 3.2

3.3 Benchmark Execution

In the above two sections, essential properties are discussed in order to make synthetic data and queries look like the original. Using these important properties, the difference in execution time of the original and synthetic workload could be minimal. However, the above only discussed properties about the workload but not specifics on the benchmark execution on the original and synthetic data and queries. These specifics are also missing in current benchmarks, as mentioned in Chapter 1. The conditions in which the benchmark is executed could influence the execution time and is therefore essential.

Amount of data. The first essential property in the benchmark execution is the amount of data used. Since the data in this work consists of the original and synthetic tables, both tables should be filled with a similar amount of data. If the original tables contain less data than the synthetic tables or the other way around, the execution time could be influenced. Fewer data would result in an execution where the database could execute a query faster because fewer data would have to be retrieved, making the total execution time faster.

Number of queries. The second essential property in the benchmark execution is similar to the essential property mentioned above, however focuses on the queries instead of the data. When the synthetic workload contains more queries than the original workload, the execution time of the synthetic workload will take longer. Therefore, the same number of queries should be used for the original and synthetic workload to make a proper assumption about the execution times.

Chapter 4

Implementation

This chapter discusses the implementation of the essential properties introduced in the previous chapter. The implementation will be addressed in a general manner since the implementation could be reused on other workloads.

Figure 4.1 shows the steps of preserving the essential properties of a workload, and the different steps will be discussed using the circled numbers. Steps (1) and (2) are use case specific and will be discussed in Chapter 5, which is downloading the seed dataset. The discussion of the implementation will follow the same categories introduced in Chapter 3 and will have two additional sections. The fourth section is related to the implementation, such as practical constraints while downloading the seed dataset. The last section explains the different programming languages used shown in the figure. Using the created benchmarking tool, the essential properties can be preserved and the usage of this tool will be explained in this last section.

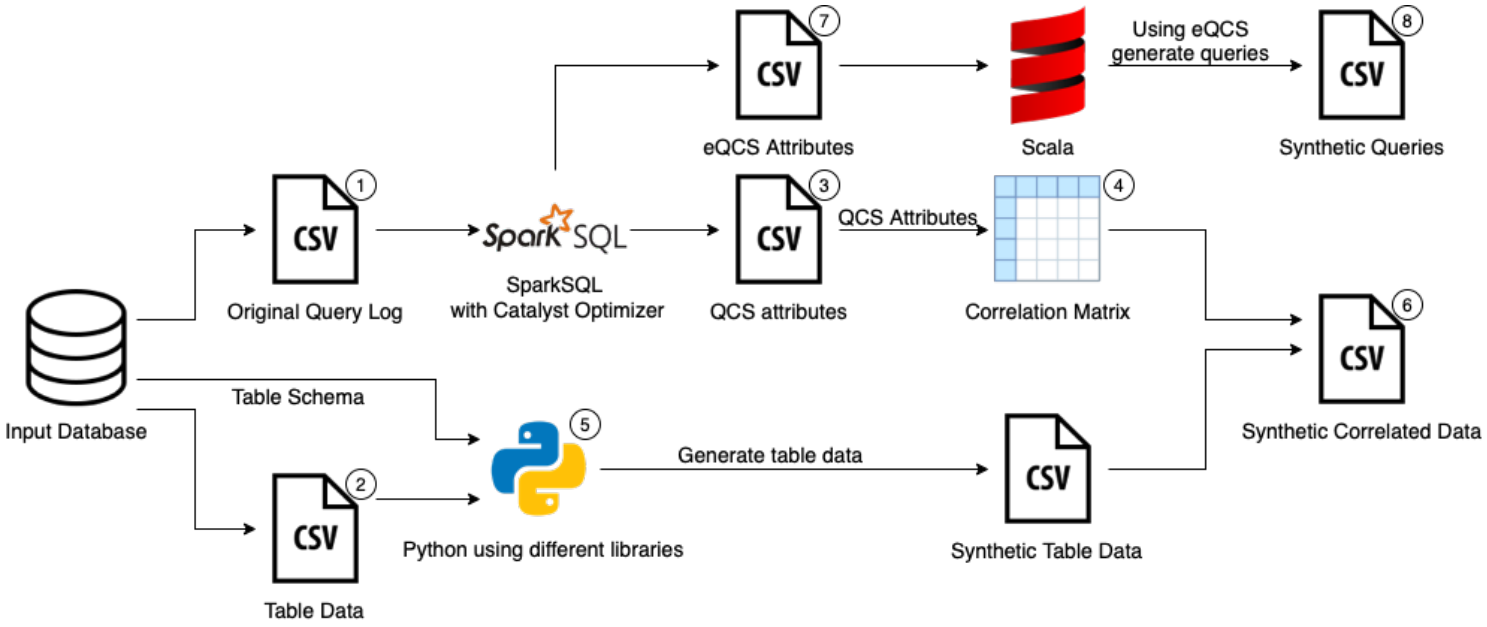

Figure 4.1: Diagram showing the different implementation steps.

4.1 Essential Properties of the Data

The essential properties of the data are the correlations and cross-relationships, as mentioned in Section 3.1. The implementation of preserving these properties could be seen in steps (3), (4), and (5) of Figure 4.1. These implementation steps will be further explained below.

Correlations in a dataset could be extracted using a Query Column Set (QCS) shown in step three of Figure 4.1 and was implemented using SparkSQL. A QCS is a set of attributes names in the `WHERE`-, `GROUP BY`-, and `HAVING`-clauses used by queries [11]. QCSs are used to predict attributes in future queries; however, in this work, an assumption was made based on QCS. The assumption is that attributes grouped in a QCS are correlated since the attributes are often grouped in a query. Since QCS extracts correlations from queries, the implementation was done on a query log as shown in step three of the figure and not on the dataset. Based on a QCS, these correlations could be found between different tables but also within tables. The dataset was used to select the attributes stated in the different QCSs for creating a correlation matrix, as shown in step four of Figure 4.1. The created correlation matrix contains the correlation coefficients between the attributes; synthetic data could be generated following the same correlations based on this correlation matrix.

The second essential property of the data is the cross-relationships in the dataset, and using the joining keys mentioned in Section 3.1; the cross-relationships were extracted. Extracting these cross-relationships was done using the schemas of each table in which these joining keys were defined. Depending on the used database, these table schemas could be found in the database itself when looking at the properties of a table. In this work, the tables schemas were used from a website[1] and is shown as the outgoing middle arrow of "Input Database" in Figure 4.1.

The table schemas were used to generate synthetic data for the attributes not stated in a QCS and are therefore not in the correlation matrix. This synthetic data will have the correct data type based on the table schema, shown in step five of Figure 4.1. The data based on the correlation matrix will be merged with the generated data from the table schemas shown in step six in the figure.

4.2 Essential Properties of the Queries

Making the synthetic queries look like the original could be done using the different operators and clauses of the original query, as mentioned in Section 3.2. The implementation of the latter is shown in step seven of Figure 4.1 using SparkSQL. The essential properties of the queries were extracted using a query log shown in step one of the Figure. The previous Section discussed QCS, which uses the `WHERE`-, `GROUP BY`-, and `HAVING`-clauses of a query. Extending QCS could extract additional clauses such as, the `SELECT`-, `FROM`-, and `JOINING`-clauses that are essential properties of the queries. Using these extra clauses, a synthetic query could be generated however; the usage of these extra clauses does not guarantee that a similar amount of rows is returned. As mentioned in Section 3.2, using the same attributes and similar query operators could return a similar amount of rows. QCS captures the names of the attributes stated in the `WHERE`-, `GROUP BY`-, and `HAVING`-clauses. The extended QCS (eQCS) captures attributes stated in the other clauses mentioned above and also possible operators used in those clauses by adding them to the QCS.

Extracting this additional information from a query creating the eQCS was done using the unresolved query plan in SparkSQL. This plan was used because attributes were shown linked to their corresponding tables, making capturing these attributes possible in the form of *table_name.attribute_name*. Synthetic queries containing the essential properties, could be generated using the implementation of eQCS, as shown in step eight of Figure 4.1.

[1] `http://skyserver.sdss.org/dr16/en/help/browser/browser.aspx`

4.3 Essential Properties of the Benchmark Execution

The synthetic and original workload should be tested by executing a benchmark that contains the essential properties for this execution, as mentioned in Section 3.3. In the above two sections, the essential properties of the workload could be extracted and used to generate synthetic queries and correlated data. The essential properties for the benchmark execution should be regulated during this generation.

The size of the synthetic dataset should be equal to that of the original since the amount of data is the first essential property. This is regulated by using a size parameter when generating the synthetic data, and the usage will be further explained in Section 4.5. The second essential property is the number of queries and, as with the data, should be equal in numbers. Using eQCS, a single query for each original query is generated and ensures the number of queries in the original and synthetic workloads is the same.

4.4 Handling Practical Constraints

The above three implementations can be used on different workloads and databases; however, some practical constraints might be applied depending on the use case. In some instances, downloading a whole dataset or a whole query log might not be possible due to external influences. In this use case, which will be discussed in Chapter 5, downloading a whole dataset was impossible due to the limitations of the website called SkyServer[2]. This limitation means some tables can be left empty while executing queries, which resulted in empty results that could negatively influence the execution times. These empty tables should be removed from both the database and the queries to retrieve a result from the dataset being used.

Another constraint is related to the execution of queries in a database, and each database executes a query differently. Since the focus is on the queries written by users, some of them could be written that are inefficient for a database; in this use case, those were the usage of subqueries. Databases do not handle subqueries well and execute these slower. Therefore next to removing empty tables from query statements subqueries had to be flattened.

4.5 Using the tool

The implementation mentioned in the Sections above could be reproduced using different scripts written for the implementation and found on a repository on GitHub[3]. As shown in Figure 4.1, Python and Scala were the main programming languages used for the implementation in Scala. SparkSQL in combination with Catalyst Optimizer was used to retrieve an analyzed query plan that shows the unresolved column names. Python was used with some additional libraries that will be further explained in Section 4.5.

The repository could be cloned to a local computer using Git or the GitHub Desktop App and could be opened in a code editor such as VSCode. VSCode[4] was also used to run the Python scripts, and the integrated development environment (IDE) IntelliJ IDEA[5] was used to run the Scala files. The Python scripts and Scala files could be found in their respective folders inside the `code` folder. Python version `3.7.0` and Spark version `3.1.1` using Scala version `2.12.10` were used to write the implementation.

[2] `http://skyserver.sdss.org/dr16/en/help/docs/limits.aspx`
[3] `https://github.com/TxEddy/2IMC00_thesis_benchmark`
[4] `https://code.visualstudio.com`
[5] `https://www.jetbrains.com/idea/`

Prerequisites

The scripts and files are dependent on four folders already present in the repository; however, when reusing the code, these folders should be present These folders are used for the output of the code and are automatically created, if not present, using a function called `create_dirs()`. This function should be run first and is stated in the `main` function located in the `synthetic_tables.py` script. All other declared functions should be commented out except the `create_dirs()` to create these folders.

Scala Scripts

In Scala QCS and eQCS were implemented in the `QCS.scala` file and to use this file some changes have to be made in the methods `qcsToCsv()`, `generateQueries()`, and `getQueriesCsv()`. In these methods, the path to the repository has to be changed. Currently, as of writing that path until the repository is `/Users/eddy/Documents/study_github/` and should be changed to the path where the repository was cloned. In `getQueriesCSV()`, the variable `qcsTableColumn` should be changed to a query that contains the attributes names stated in all the QCSs. Next to changing these paths, the other variables `DirLog` and `DirTable` should be changed to the folders containing the query log and tables. After changing the paths and variables, the `QCS.scala` could be run in an IDE such as IntelliJ IDEA.

Python Scripts

Synthetic data, and a correlation matrix were created using Python. Using this correlation matrix, synthetic correlated data was generated. As mentioned above, three libraries were used to generate the synthetic data. The Python scripts are located in the folder called `python`, and the important scripts are named `db_execute.py` and `synthetic_tables.py`.

The script `synthetic_tables.py` generates synthetic tables using a correlation matrix based on the large table containing the correlated attributes. The table schema will be used to generate synthetic data for attributes not present in this correlated table. Mentioned above, an essential property for the benchmark execution was the *Amount of data* that should be the same for the original and synthetic data. This could be regulated by using the parameters `generate_number` in the functions `correlation_matrix(...)` and `generate_table_data(...)`. The function `update_synthetic_table` merges the synthetic data and correlated synthetic data into one table. Lastly, in the script `database_tasks.py` different functions are stated that could be used for the three databases that will be discussed in the next Chapter. These functions are used for the execution of the benchmark and could be reused to execute another benchmark.

Chapter 5

Use Case

This chapter discusses a use case in which the essential properties of the workload were identified as introduced in Chapter 3. These properties were captured and implemented following the implementation discussed in Chapter 4 to create a synthetic workload that looks like the original. Figure 5.1 shows that implementation in this use case and is an extended version of the figure introduced in the previous chapter. This implementation is shown in steps three, four, five, six, seven, and eight of Figure 5.1, which have already been discussed. The first two steps are use case specific, as mentioned in Chapter 4, and will be discussed below. Outputs of this implementation and steps nine, ten, and eleven of Figure 5.1 are part of the execution of the benchmark and will be further discussed in Chapter 6. Chapter 4 briefly discussed data was downloaded from a website called SkyServer; this data is collected by Sloan Digital Sky Survey (SDSS). The practical constraints mentioned in Section 4.4 will be discussed when downloading the data from SDSS.

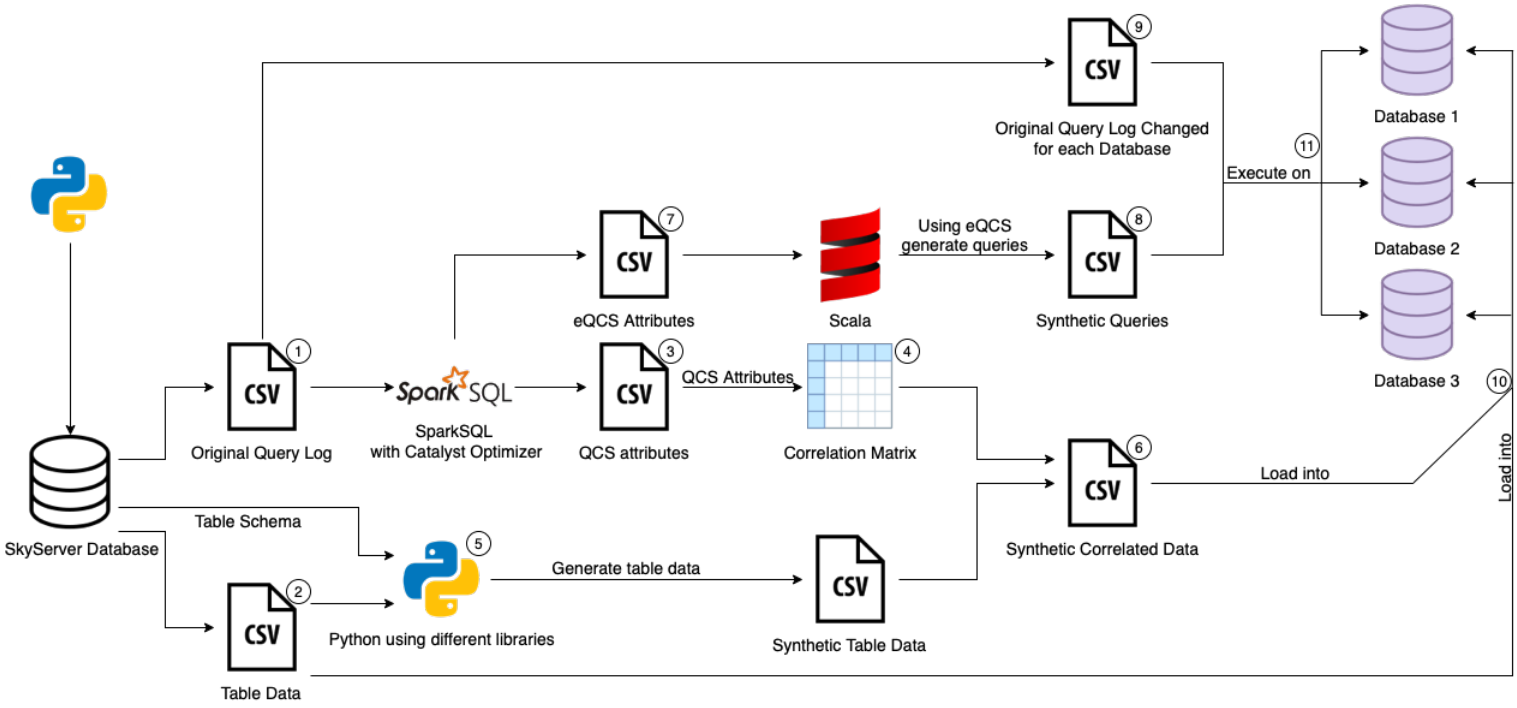

Figure 5.1: Data Flow Diagram showing the different implementation steps for the use case.

5.1 Context

One of the problems of using a real-world workload is the data and queries since companies do not want to share them, as mentioned in Section 1. The dataset and queries used in this work come from SDSS. SDSS captures different images from the sky using other 2.5mm telescopes [12], these images are processed, and numerical estimates for the physical attributes of objects in these images are saved. Since 2001 SDSS has had different data releases from which the latest is DR16, which was also used for this work [13]. The captured data is stored and organized in different tables that have relationships, as shown in Figure 5.2. Figure 5.2 shows the different tables and their cross-relationships, and based on these tables a selection was made. The data is publicly available via SkyServer and mainly used by professors and students who can query this table data. These queries written by the professors and students are logged and also publicly available through a specific page[1].

[1] http://skyserver.sdss.org/log/en/traffic/sql.asp

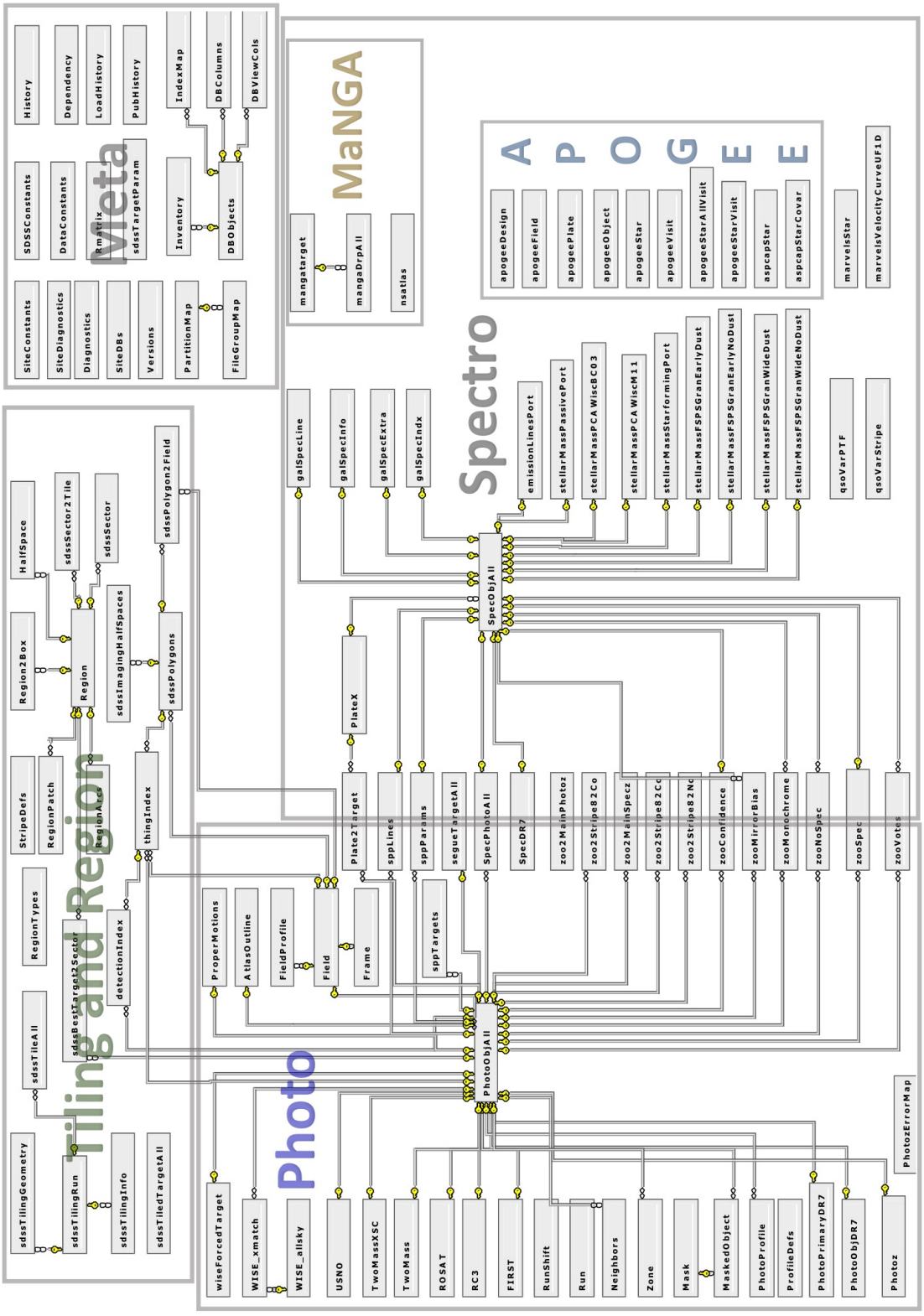

Figure 5.2: Data Flow Diagram showing the different implementation steps for the use case.

5.2 Acquiring The Seed Dataset

In this use case, a dataset of 20 tables and a query log consisting of 1700 queries were downloaded using SkyServer, as mentioned above. Two specific webpages of SkyServer were used to download the workload, one for the table data[2] and another one for the query log[3]. As shown in Figure 5.1, Python was used to download this workload using these webpages. This Python script is in the same repository as mentioned in Section 4.5. Subsection 5.2.3 will discuss how the selection of the query log was made, and subsection 5.2.2 the selection of the tables.

5.2.1 Base Query Log

Before downloading the actual query log and table data used in this use case, a base query log was downloaded. Figure 5.2 shows all the tables available in SkySever, and downloading all these tables would be impossible as mentioned in Section 4.4. Using a base query log a selection of the tables could be made and using filters tables can be selected that are used in queries of our scope. Year is another filter because at the time of writing there was a minimal amount of queries available of the year 2021. Filtering on the year 2020, a larger selection of the tables and queries was possible. When downloading this base query log, the types of queries were already determined to use. Using a query, the query log was downloaded from the specific SkyServer webpage; in this query the latter mentioned filters were used and could be seen in Listing 5.1. In this listing the filter for the year (line 3) could be seen and in lines six until ten the filters for the type of queries. Lines four and five show other filters related to queries raising no syntax errors and filtering on user-created queries. The last filter was used since SkyServer also makes queries available that were used to maintain the database.

```
1   SELECT *
2   FROM SqlLog
3   WHERE yy = 2020
4       and access like 'Skyserver.Search%'
5       and (rows > 0 and error != 1)
6       and (lower(statement) like '%count(%'
7       or lower(statement) like '%avg(%'
8       or lower(statement) like '%sum(%'
9       or lower(statement) like '%group by%'
10      or lower(statement) like 'select%'
11      )
12      and lower(statement) like '%join%'
```

Listing 5.1: Query used to download the base query log

5.2.2 Table Data

Using the base query log, a frequency table was created that contains the tables names and the frequency a table was used in the `FROM` clause. The top 20 tables were selected using this frequency table, shown in Table 5.1. The next step was downloading table data for these 20 tables in which different decisions were made based on constraints mentioned in Section 4.4. One of these decisions selecting the tables since downloading all tables would not be possible. Related to this decision was choosing to select one of the main two tables, which is `PhotoObjAll` this decision was made using the frequency table. The second main table is `SpecObjAll`; however, this table was not as often stated in the `FROM` clause as `PhotoObjAll`.

[2] http://skyserver.sdss.org/dr16/en/tools/search/sql.aspx
[3] http://skyserver.sdss.org/log/en/traffic/sql.asp

Table name	Frequency
PhotoObj	1030
Galaxy	921
SpecPhoto	129
stellarMassFSPSGranEarlyDust	54
PhotoObjAll	46
mangaPipe3D	45
mangaDRPall	43
SpecPhotoAll	41
sppLines	29
GalaxyTag	28
Photoz	24
sppParams	19
galSpecLine	19
galSpecIndx	15
zooSpec	10
apogeeStar	9
PhotoTag	8
galSpecExtra	8
WISE_xmatch	3
mangaGalaxyZoo	3

Table 5.1: Frequency table with the top 20 selection of the tables.

The selected tables are shown in Table 5.1 and were downloaded for this use case. The downloaded data should be joinable since the `crossrelationships` are one of the data properties, as discussed in Section 3.1. Therefore based on the 20 tables, the relationships between the tables had to be identified such that data could be downloaded, which is joinable. An overview of the relationships between the tables could be seen in Figure 5.3.
The table data could be downloaded using queries as with the query logs. The identified relationships between the tables were used in these queries. As mentioned above, SkyServer has some constraints when downloading the table data, and because the table data should be joinable, the used queries were complex. This complexity of the queries caused time-out errors making downloading the table data impossible, and a solution was to use ranges to download table data. Using a range based on a primary key, downloading table data was possible; however, still limited to downloading 25.000 rows per range. The aim was to download 200.000 rows for each table when possible. Therefore eight ranges were used since 25.000 rows could be downloaded per range. The choice of downloading 200.000 rows for each table and 250.00 per range was made based on the constraints of SkyServer. The tables related to the main table `PhotoObjAll` used the following ranges:

- 1237671260124676376 - 1237680262909460580 (1 - 25.000)
- 1237668333104136344 - 1237671260124545288 (25.001 - 50.000)
- 1237667910053920988 - 1237668333104136325 (50.001 - 75.000)
- 1237667537476714695 - 1237667910053920971 (75.001 - 100.000)
- 1237667252936507474 - 1237667537476649142 (100.001 - 125.000)
- 1237665535465685136 - 1237667252936507458 (125.0001 - 150.000)
- 1237665329321672933 - 1237665535465619929 (150.001 - 175.000)
- 1237664877803143327 - 1237665329321672838 (175.001 - 200.000)

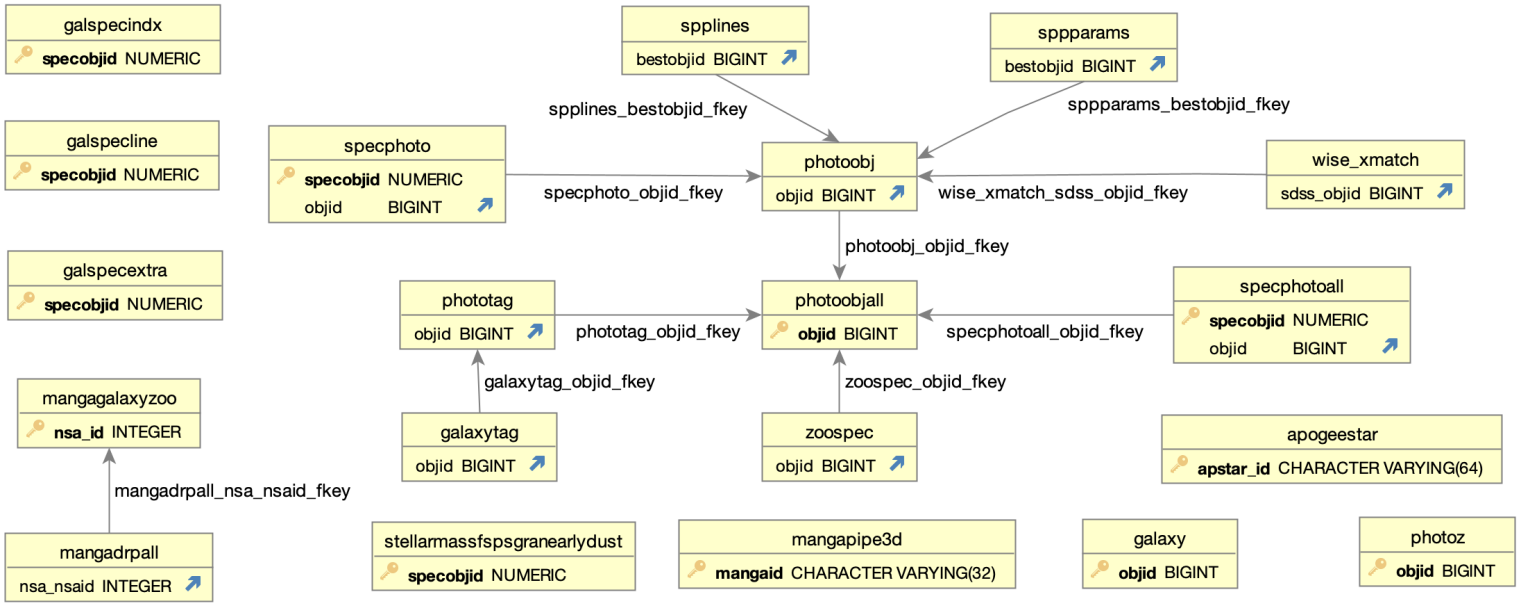

Figure 5.3: The selected tables and their relationships.

However, for the tables `galaxyTag`, and `sppParams`, 100.00 rows could be downloaded due to constraints of SkyServer. Instead of using the first four ranges stated above, eight other ranges were used as a result of these constraints. By downloading 12.500 rows per range, no time-out errors were given. The used ranges are:

- 1237673703969259741 - 1237680262909460580 (1 - 12.500)
- 1237671260124676376 - 1237673703968801146 (12.501 - 25.000)
- 1237668623553200221 - 1237671260124545288 (25.001 - 37.500)
- 1237668333104136392 - 1237668623553134728 (37.501 - 50.000)
- 1237668272983834891 - 1237668333104136344 (50.001 - 62.500)
- 1237667910053920996 - 1237668272983769437 (62.501 - 75.000)
- 1237667736111611963 - 1237667910053920988 (75.001 - 87.500)
- 1237667537476714786 - 1237667736111546579 (87.501 - 100.000)

The tables `MangaPipe3D`, `MangaDRPall`, and `mangaGalaxyZoo` also use another range, but this is due to the availability of joinable data. The range used for these three tables was 24982 - 694828 using the joining key `nsa_id`. Figure 5.3 shows six tables not connected to the main table `PhotoObjAll` but for these tables, data was also downloaded using the primary key of `PhotoObjAll`. This primary key was used for these six tables because these tables were able to join using the joining key of the other main table `SpecObjAll`.

5.2.3 Query Log

The above section discussed how the base query log was used to make a selection for the tables. A final query log containing these tables was downloaded. The final query log was downloaded using the same webpage from which the base query log was obtained. However, an extended query of Listing 5.1 was used to retrieve this final query log, as shown in Listing 5.2. Listing 5.2 contains the 20 tables mentioned in the section above and one other filter stated in line 34 of the listing. This filter avoids downloading queries in which functions are stated that are solely available in the SkyServer database. The final query log consisted of 1700 queries.

```
1  SELECT *
2  FROM SqlLog
3  WHERE yy = 2020
4      and (access like 'Skyserver.Search%')
5      and (rows > 0 and error != 1)
6      and (lower(statement) like '%count(%'
7      or lower(statement) like '%avg(%'
8      or lower(statement) like '%sum(%'
9      or lower(statement) like '%group by%'
10     or lower(statement) like 'select%'
11     )
12     and lower(statement) like '%join%'
13     and (lower(statement) like '%photoobj%'
14     or lower(statement) like '%galaxy%'
15     or lower(statement) like '%specphoto%'
16     or lower(statement) like '%stellarmassfspsgranearlydust%'
17     or lower(statement) like '%photoobjall%'
18     or lower(statement) like '%mangapipe3d%'
19     or lower(statement) like '%mangadrpall%'
20     or lower(statement) like '%specphotoall%'
21     or lower(statement) like '%spplines%'
22     or lower(statement) like '%galaxytag%'
23     or lower(statement) like '%photoz%'
24     or lower(statement) like '%sppparams%'
25     or lower(statement) like '%galspecline%'
26     or lower(statement) like '%galspecindx%'
27     or lower(statement) like '%zoospec%'
28     or lower(statement) like '%apogeestar%'
29     or lower(statement) like '%phototag%'
30     or lower(statement) like '%galspecextra%'
31     or lower(statement) like '%wise_xmatch%'
32     or lower(statement) like '%mangagalaxyzoo%'
33     )
34     and lower(statement) not like '%fget%'
```

Listing 5.2: Query used to download the final query log

Chapter 6

Experiments

As mentioned in Section 1, a new benchmark is developed by reverse engineering since the original, and synthetic data and queries are tested and not the database. The new benchmark was tested holistically, which means the difference in execution times between the original and synthetic data and queries should be minimal. The execution times should be as close as possible to each other however, each database executes queries differently. Therefore, getting a specific difference in seconds was not viable to compare the execution times. A better way to compare the execution times was by looking at the difference in percentages, which should not be more than 20%. Since a 20% difference is a difference of around 10 seconds when the execution times do not take longer than five minutes. As discussed in Section 1 and shown in Figure 6.1, three different database management systems (DBMS), off-the-shelve DBMSs, were used to test the new benchmark. In Section 6.1, the execution of the benchmark is described using steps 9, 10, and 11, shown in Figure 6.1. While testing and executing the benchmark, different combinations of the essential properties introduced in Section 3 were used. Section 6.2 will discuss the influence of these essential properties on execution times.

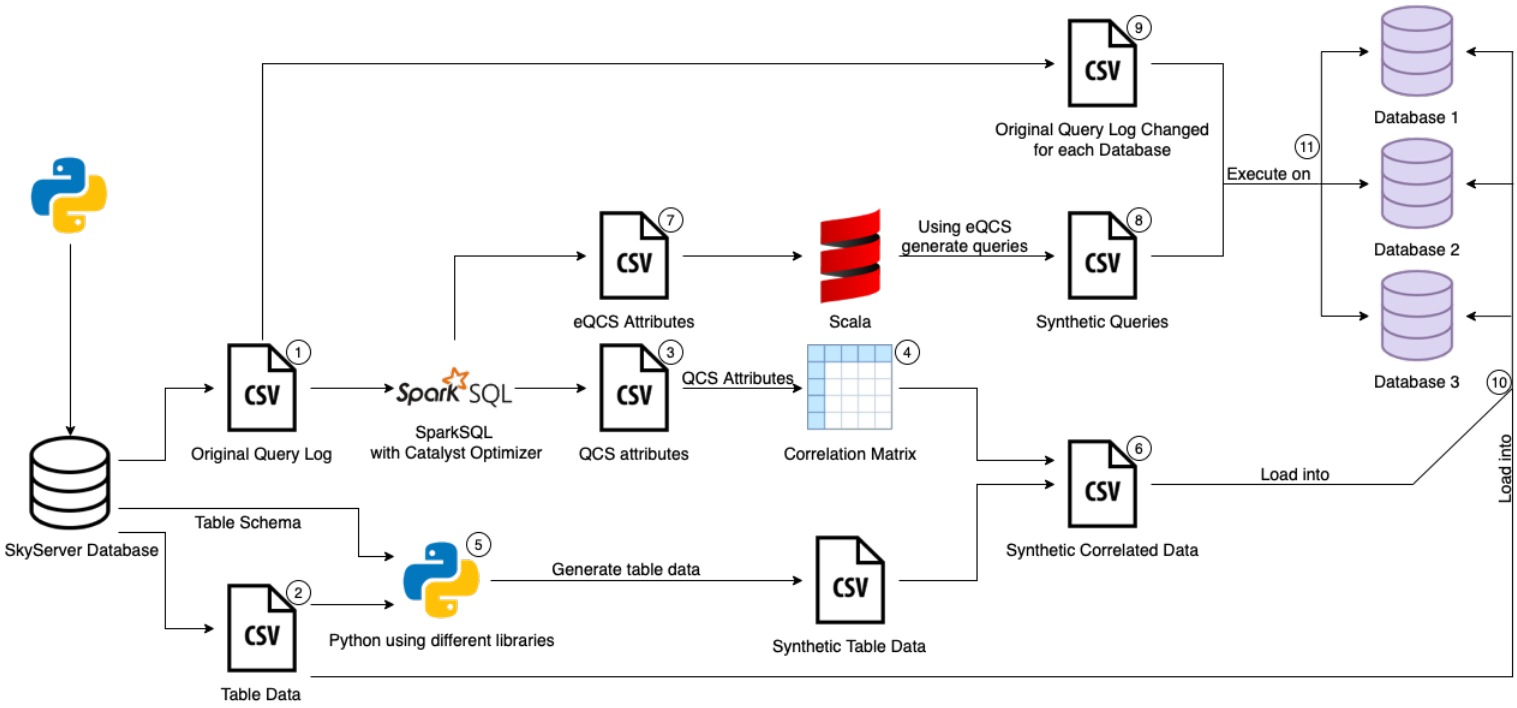

Figure 6.1: Data Flow Diagram showing the different implementation steps for the use case (copy of Figure 5.1).

6.1 Evaluation Setting

Section 5.2 discussed the selection and downloading of the workload, and using the steps discussed in Section 4, synthetic table data and queries are generated. Before the benchmark is executed, the data should be loaded into the databases, shown as step 10 in Figure 6.1. The data were manually loaded into the databases by specifying the paths to the table data in the load commands of each database. In step 9 of Figure 6.1, the query statements in the original query log were changed to the accepted syntax of each database. Using the Python script mentioned in Section 4.5, a connection to the database will be made and the original and synthetic queries will be executed. Making the connection and executing the original and synthetic queries in the three databases is shown in step 11 in Figure 6.1. The script exports the execution times to a CSV format, as seen in Table 6.1; those execution times are in seconds. Since the execution of the original and synthetic workloads was done locally on a laptop, the execution times could fluctuate even though as much applications were closed. Therefore, the original and synthetic workload were executed 15 times in the benchmark using the script, and the average execution times were compared. After one execution a break of two minutes was scheduled meaning executing both workloads more than 15 times would take too long. Table 6.1 is an example of the output and are not real execution times, and these will be explained in the following Section.

Database 1	Database 2	Database 3
55.30	45.19	42.85

Table 6.1: Output format containing the execution times of the queries, which are in seconds.

6.2 Testing the Benchmark

This section will discuss the different execution times after implementing the essential properties. This implementation of the essential properties was done in three different phases, and the order of implementation was not in a particular fashion. The execution times shown in the different tables are the total averages in seconds after 15 executions.

6.2.1 Phase 1

Five essential properties were implemented in the first phase when generating the synthetic workload. These five essential properties were implemented first otherwise a synthetic workload could not be generated. These properties were `correlations` of the data, the `query operators` and `query clauses` of the queries, the `amount of data` and the `number of queries` for the benchmark execution. In this phase and the other phases the `amount of data` and the `number of queries`, which are 1700 queries, for the original and synthetic stays the same. After this implementation the benchmark on both original and synthetic workloads was executed as discussed in the above section for the first time. The results of the execution times are shown in Table 6.2.

	Database 1	**Database 2**	**Database 3**
Original	131.66	18.12	30.82
Synthetic	68.98	6.88	15.04

Table 6.2: The average execution times in seconds of the original and synthetic data and queries in the three databases.

As one can observe from Table 6.2, the difference between the original and synthetic workload is not within the desired 20% difference. Another observation are the fast execution times for both original and synthetic workload. This occurs because not all essential properties were implemented yet and was an initial execution.

6.2.2 Phase 2

In the second phase, all essential properties were implemented. Also, one of the practical constraints was incorporated. This practical constraint was the limitation in downloading the whole dataset, which was partly discussed in Section 5.2. Since a part of the dataset could be downloaded, tables outside the scope are empty but could still be mentioned in the query log. Queries containing these tables would return empty results and could be a reason for the fast execution times shown in Table 6.2 . Therefore the tables outside this scope were removed from the query log.

In phase one of the benchmark execution, the property that was not yet implemented was the `cross-relationships` of the original data. When downloading the data, the cross-relationships were taken into account by trying to download joinable data. However, even though this was considered, not all data was joinable since only a part of the data was downloaded. A solution was to use the primary keys and joining keys `SpecObjID` of the downloaded data of the main table `PhotoObjAll`. The joining key `SpecObjID` was used for tables that are not joinable using the primary key. In each table, these two joining keys of `PhotoObjAll` were compared to the keys available in that table. When a joining key in the table was not present in `PhotoObjAll`, this joining key would be replaced with a key of `PhotoObjAll` not yet present in the table. After taking care of these joining keys, the benchmark was executed for a second time, and the execution times are shown in Table 6.3.

	Database 1	**Database 2**	**Database 3**
Original	907.24	76.35	153.48
Synthetic	757.92	45.98	143.66

Table 6.3: The average execution times in seconds of the original and synthetic data and queries in the three databases of phase 2.

In the table above, the average execution times seem more realistic. This means that by removing table names, that were not filled with data, from the query log had a positive effect on the benchmark. The execution times are also closer to each which means the property `cross-relationships` introduced in Section 3.1 are an essential property for making a synthetic workload look like the original. However, the difference in execution times is not yet the desired difference, as mentioned at the beginning of this chapter.

6.2.3 Phase 3

In Table 6.3 all essential properties were incorporated however, another practical constraint was not yet discussed which was the subqueries that could influence the execution times. After looking at the execution of queries containing subqueries, `Database 3` did not handle subqueries optimally. Therefore in phase three, the subqueries in the original query log were flattened, and the benchmark was again executed. The execution times of phase three could be seen in Table 6.4.

	Database 1	**Database 2**	**Database 3**
Original	785.92	57.19	142.85
Synthetic	709.88	48.46	142.14

Table 6.4: The average execution times in seconds of the original and synthetic data and queries in the three databases of phase 3.

An observation that could be made from Table 6.4 are the different execution times compared to the execution times shown in Table 6.3 of Phase 2. There are two reasons for this difference in execution times. The first is the removal of subqueries of the original query log and regenerating the synthetic query log based on this altered log. The second reason is the local execution of the benchmark even though as much applications were closed during the execution. Even tough the execution times are different compared to Phase 2, the execution times shown in Table 6.4 are closer to each other compared to Phase 2. The difference between the original and synthetic data and queries is within the 20% difference. This means all implemented properties and incorporated practical constraints can make the synthetic workload look like the original.

Chapter 7

Conclusions

A benchmarking tool was created and used to create a synthetic workload that looks like the original workload in this work. Making a synthetic workload look like the original entails that this workload should have the same properties. Therefore essential properties were introduced and could be categorized into data, queries, and benchmark execution. These essential properties should be captured from the original workload and used to create a synthetic version. The implementation of the essential properties was discussed, as well as the dataset used to capture these properties. A benchmark was created, executed and tested holistically by focusing on the benchmark's performance. The goal is to have a minimal difference in execution times between the original and synthetic workload. When this difference is within 20%, the conclusion could be made that the synthetic workload behaves like the original workload. The benchmark was executed in three different phases, and in each phase, different essential properties were implemented, or practical constraints were taken into account. After each phase, the execution times were closer until all essential properties were implemented in the third phase. There was a minimal difference in execution times in this last phase, and the conclusion could be made that the synthetic workload looks like the original. Although the benchmark was executed on a specific use case to conclude the latter, the identified essential properties could be implemented in different use cases.

7.1 Limitations

Even though the synthetic workload looks like the original, there are some limitations to the method used and discussed in Chapter 4. This chapter discusses how a correlation matrix was used to capture the essential property `correlations`. However, this correlation matrix was created using pairwise correlation, which is not optimal since two attributes are compared. A better alternative would be to compare one attribute with multiple attributes; pairwise is limited to finding correlations between two attributes. An example is three attributes `A`, `B`, and `C`, where `A` is correlated with the combination `B` and `C`. Such a correlation could not be found with a pairwise correlation.

Another limitation is related to the correlations and the distribution of the tables. In the current implementation, the correlation of the attributes between tables is taken into account but not the distribution of the tables. Due to time constraints, this was not implemented but could improve the quality of the synthetic data.

The last limitation is the generation of the synthetic queries, precisely the conditions used in the `WHERE`-clauses. Currently, the conditions of the original queries are transferred to the synthetic queries; this does not cause errors since the median was taken into account when generating synthetic data based on the correlation matrix. This correlation matrix was created using Query Column Sets (QCS), which captured attributes stated in the `WHERE`-clauses. This limitation is also part of future work.

7.2 Future Work

As mentioned above, the conditions in the `WHERE`-clauses are a limitation and could be improved by using conditions based on generated data. Another way to improve the synthetic data generation is by using Kramer's coefficient to generate the data types String or varchar. These data types were generated using the constraints defined in the table schema. Using these constraints, a random String length is generated; however, instead of generating random Strings, more representative Strings could be generated using Kramer's coefficient. Next to improving the synthetic data generation, the synthetic queries could also be improved by using a generative adversarial network (GAN). Using the machine learning model GAN more representative synthetic queries could be generated.

Bibliography

[1] Furtado P. Barata M., Bernardino J. An Overview of Decision Support Benchmarks: TPC-DS, TPC-H and SSB. *Advances in Intelligent Systems and Computing*, 2015. 1, 4

[2] M. Olma, O. Papapetrou, R. Appuswamy, and A. Ailamaki. Taster: Self-tuning, elastic and online approximate query processing. *Proceedings - International Conference on Data Engineering*, 2019. 1

[3] Nambiar R. Poess M. The Making of TPC-DS. *Proceedings of the 32nd International Conference on Very Large Data Bases*, 2006. 1, 4

[4] S. Jain, D. Moritz, D. Halperin, B. Howe, and E. Lazowska. SQLShare: Results from a Multi-Year SQL-as-a-Service Experiment. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2016. 1, 2

[5] P. Eichmann, E. Zgraggen, C. Binnig, and T. Kraska. IDEBench: A Benchmark for Interactive Data Exploration. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020. 4

[6] A. Vogelsgesang, M. Haubenschild, J. Finis, A. Kemper, V. Leis, T. Muehlbauer, and et al. Get real: How benchmarks fail to represent the real world. *Proceedings of the Workshop on Testing Database Systems*, 2018. 4

[7] Your data has a story. share it with the world. `https://public.tableau.com/s/`. 4

[8] S. Deep, A. Gruenheid, P. Koutris, J. Naughton, and S. Viglas. Comprehensive and Efficient Workload Compression. *roceedings of the VLDB Endowment*, 2020. 5

[9] L. Battle, P. Eichmann, M. Angelini, T. Catarci, G. Santucci, Y. Zheng, C. Binnig, JD. Fekete, and D. Moritz. Database benchmarking for supporting real-time interactive querying of large data. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020. 5

[10] I. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. CORDS: Automatic discovery of correlations and soft functional dependencies. *Proceedings of the 2004 ACM SIGMOD international conference on Management of data - SIGMOD '04*, 2004. 6

[11] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: Queries with bounded errors and bounded response times on very large data. *Proceedings of the 8th ACM European Conference on Computer Systems - EuroSys '13*, 2013. 10

[12] M. R. Blanton, M. A. Bershady, B. Abolfathi, F. D. Albareti, C. Allende Prieto, A. Almeida, J. Alonso-García, F. Anders, S. F. Anderson, B. Andrews, and et al. Sloan Digital Sky Survey IV: Mapping the Milky Way, Nearby Galaxies, and the Distant Universe. *aj*, 154:28, jul 2017. 14

[13] R. Ahumada, C. Prieto, A. Almeida, and et al. The 16th Data Release of the Sloan Digital Sky Surveys: First Release from the APOGEE-2 Southern Survey and Full Release of eBOSS Spectra. *The Astrophysical Journal Supplement Series*, 249:3, 2019. 14