TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Nils Tirs 178947IAIB

TERADATA TECHNICAL METADATA AGGEGATION ON A COMMON DASHBOARD

Bachelor's thesis

Supervisor: Martin Rebane
MSc

Tallinn 2020

Author acknowledgement

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Nils Tirs

25.05.2020

Abstract

Teradata Technical Metadata Aggregation on a Common Dashboard

The purpose of this thesis is to create a Teradata technical metadata showing dashboard in a data warehouse for a financial institution, to provide key technical metadata necessary.

The current solution in the financial institution was to create and use long and complicated queries, which took-up a significant amount of time and resources for a database administrator.

To solve this problem, necessary queries were created in association with the financial institutions database administrators and a web-application was created. The common dashboard was created as an ASP .NET MVC with a connection to the financial institutions data warehouse.

The dashboard was able to show all necessary data from key databases and their child tables significantly faster in comparison to executing each SQL statement on its own.

The thesis is written in English and contains 5 chapters, 27 pages and 11 figures.

Annotatsioon

Teradata tehnilise metadata agregeerimine ühtsele koondvaatele

Käesoleva töö eesmärgiks oli luua Teradata tehnilise metaandmeid ülevaade finantsinstitutsiooni andmeaidast. Lahendus koosneb veebirakendusest, mis näitab põhilist tehnilist metaandmeid andme lauda tähtsamates andmebaasides olevatest tabelitest.

Rakenduse eesmärk oli oluliselt lihtsustada finantsinstitutsioonis olevate andmebaasi administraatorite tööd, luues veebirakendus, kus saab näha kõikide tähtsamate andmebaaside tabelite tehnilist metaandmeid. Selle hulka lisandub tabeli vaba ja täitunud maht, mis kasutajatel on vastavale tabelile õigused ja mis tabelid kasutavad või sõltuvad tabelist. Antud metaandmetega saab leida korrelatsioone, et andmelauda tööd hõlbustada.

Töö käigus loodi vastavad SQL laused, rakenduse *back-end* ja kasutajavaade. SQL laused loodi Teradata SQL'is kasutades Teradata's olevat „sõnastik" andmebaasi vastava metaandmetega kättesaamiseks. Veebirakendus loodi finantsinstitutsioonis arenduskäigus oleva andmelauda testimis rakenduse laiendusena ASP .NET MVC's.

Edasiarendusel on võimalik lisada tööle metaandmete ajaloo hoiustamise funktsioon, lisada graafikuid ja filtreerimisvõimalus. Näiteks, et andmebaasi tabelid vaba ruumi järgi, või mis tabelil on kõige rohkem sõltuvusi. Lisaks võib lisada muud andmelauda metaandmeid, et tekitada veel üldisem ülevaade hetkeseisust andmelauda tabelites.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 27 leheküljel, 5 peatükki ning 11 joonist.

List of abbreviations and terms

AMP	*Access Module Processors* are Teradata data base system processors to which tables are divided to.
BI	*Business Intelligence* stands for strategies and technologies used by organizations for data analysis.
CPU	*Central Processing Unit* is considered the brains of the computer and is the part where all the calculations in a computer are done in.
Dashboard	An information management tool used to analyse and track key metrics in a program, organization or a specific process.
Data warehouse	A database management system used mainly in data analysis and Business Intelligence.
Metadata	*Metadata* is the data about data and gives information about other data.
MVC	*Model-view-controller* principle behind ASP .NET web application development.
PE	*Parsing Engine* handles all Teradata queries by orchestrating the AMPs
RDBMS	*Relational Database Management System* is a specific database management system used for relational databases
Teradata	*Teradata* is a RDBMS used mainly in data warehouses and organizations where data analysis over large amount of data is required
Query	A *query* is a request for data or information from a database table or a combination of tables

Table of contents

List of figures

1. Introduction

Metadata. The data about data. Any organization should take the maximum out of its metadata since it can create new understandings or highlight previous problems. This is no truer in data warehouses where there is a massive abundance of data. Data warehouses being built mainly for storage and analysis can be the best platforms of metadata use.

“Metadata helps data warehouse end users to understand various types of information resources available from a data warehouse environment.” [1].

Efficient and proper use of Teradata technical metadata has been a lacking issue at the financial institution the author works at. Currently in the department in the financial institution that the author works at, the process of acquiring qualitative and necessary metadata has been a long and cumbersome process, with each key metric check requiring long and complicated queries.

The dashboard could be used to determine if any table has run out of its allocated space and needs additional resources or if the table is not optimized for Teradata. It can give additional discoveries too if any object in the data warehouse has access to the table when it should not. Since the process of getting this metadata will be improved significantly, the likeliness of discoveries with such cases should increase dramatically if this tool is used.

The author was approached by a financial institution to create a dashboard showing key metadata metrics of tables in the data warehouse, hence the aim of this thesis is to develop an efficient and simple metadata common-view allowing rapid access to key metadata at all times.

The most challenging process in this thesis was aggregating the necessary data from the financial institutions Data Warehouse as creating the queries in order to have the correct and necessary metadata including all the requirements given to the author.

1.1 Contribution

During this thesis, the author created a technical metadata viewing tool called Teradata Common-View for a financial institution where the author is an employee. The user can use this tool to check between all high priority databases and their respective child tables to see the key views of table space, table rights and table references.

For this the author designed appropriate Teradata SQL queries to get all the relevant information, developed a back-end layer to manipulate the key data and created a front-end dashboard that significantly improves metadata accessibility.

1.3 Structure of the thesis

The thesis is composed of six chapters: introduction, theoretical background, analysis, development, result and improvements and conclusion. In theoretical background an overview of Teradata, data warehouses and metadata are given, explaining some key information and necessity of metadata.

In the analysis the requirements and possible solutions of the tool are discussed and chosen upon, reaching a best possible outcome.

For the development chapter the queries, back end and front-end layers of this thesis are discussed, with heavy focus on the queries and how the key metadata was gotten. The back-end chapter describes how the data manipulation from the queries is shown and explains additionally with authors decisions in how to forward it to the front end. In front end the overall structure, fetching and query manipulation is shown in how the user view was created and how it works.

Finally, before concluding the thesis the overall result and limitations of this tool are discussed, with additional improvements suggested. In the first appendix Teradata object types can be viewed, with the following appendix containing the back-end code of this thesis.

The code of this thesis is available publicly on GitHub [2], however as this application is a part of an existing tool in a financial institution a working version is not available.

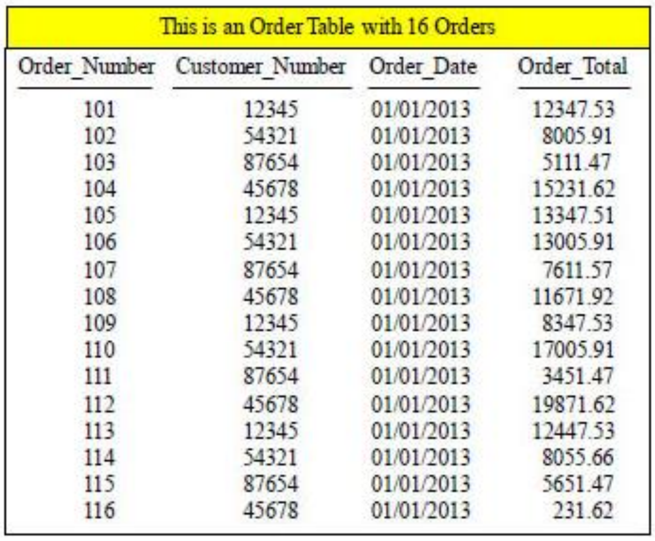
2.Theoretical background

In this chapter there is a brief theoretical overview of Teradata, Data Warehouses and Metadata. This chapter gives a necessary understanding of how Teradata works, how Data warehouses operate and how Teradata is incorporated into Data Warehouses. Additionally, a detailed overview and discussion is presented between Metadata and data warehouses to give all the necessary background information to comprehend the thesis.

2.1 Teradata

A basic single computer works in two parts. It has a hard drive disk with data, and it has a CPU (Central Processing Unit) with memory. Data on a disk does not do anything, when data is requested, the computer moves one block of data to memory, when it is in memory it is processed by the CPU at lightning speed. As said by Tom Coffing The "Achilles heel" of this entire process is that slow process of moving data from disk to memory. [3]

The solution to this is known as parallel processing, which Tom Coffing [3] sums up it in a dry joke:
"Two guys were having fun on a Saturday night when one said, "I've got to go and do my laundry." The other said, "What?!" The man explained that if he went to the laundromat mat the next morning, he would be lucky to get one machine and be there all day. But, if he went on Saturday night, he could get all the machines. Then, he could do all his wash and dry in two hours."
Teradata systems use Access Module Processors (AMP) to which all the rows in a table are spread out to.
Let us look at the table in figure 1. There we have an order Table with 16 orders with their respective attributes. The current Teradata system 16 AMP's, which means that one row is assigned to one AMP. However, if there were 32 rows then each AMP would have 2 rows. Teradata always divides the rows so each AMP can process their portion of the data. [3]



This is an Order Table with 16 Orders			
Order_Number	Customer_Number	Order_Date	Order_Total
101	12345	01/01/2013	12347.53
102	54321	01/01/2013	8005.91
103	87654	01/01/2013	5111.47
104	45678	01/01/2013	15231.62
105	12345	01/01/2013	13347.51
106	54321	01/01/2013	13005.91
107	87654	01/01/2013	7611.57
108	45678	01/01/2013	11671.92
109	12345	01/01/2013	8347.53
110	54321	01/01/2013	17005.91
111	87654	01/01/2013	3451.47
112	45678	01/01/2013	19871.62
113	12345	01/01/2013	12447.53
114	54321	01/01/2013	8055.66
115	87654	01/01/2013	5651.47
116	45678	01/01/2013	231.62

Figure 1 Order Table for parallel process explanation *[3]*

The Parsing Engine (PE) is the brains behind every Teradata system. It guides the AMP's like a conductor to an orchestra. Every time a user logs into Teradata they are directed to a Parsing Engine. The Parsing Engine also handles every query the user executes so if an user wants to execute a query the PE will first check if the user has appropriate security access to the table and once authenticated, the Parsing Engine will take the query and create a plan for the AMP's to follow. Next the Parsing Engine will pass the plan over a BYNET network and each AMP will perform their appointed tasks. Once the task is completed the Parsing engine will receive the data from every AMP and return it to the user. Simply put the PE is the boss and AMP are the workers in a Teradata system. This process is shown in the **Figure 2** with the same OrderTable. [2]

Teradata is especially useful in Data Warehouses. This is because Teradata has mastered two extremes:

1. Teradata can analyse massive amounts of data due to their parallel processing abilities

2. Teradata can find a single record in under a second, no matter the data quantity

   The latter capability is possible since every table in Teradata has column designated as the Primary Index hence when a user wants to look up a single row of data only one AMP is read.[2]

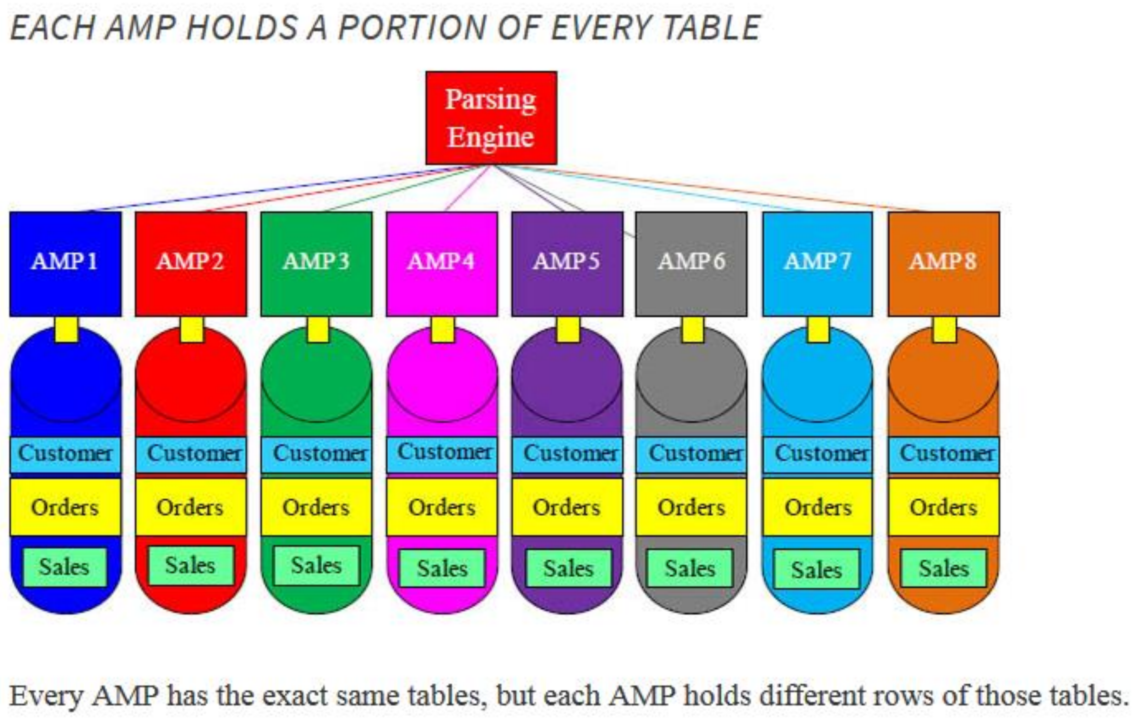

EACH AMP HOLDS A PORTION OF EVERY TABLE
Parsing Engine
AMP 1
AMP 2
AMP 3
AMP 4
AMP 5
AMP 6
AMP 7
AMP 8
Customer
Orders
Sales
Every AMP has the exact same tables, but each AMP holds different rows of those tables.

Figure 2 Example of a Parsing Engine with AMP's *[3]*

2.2 Data warehouse.

“One of the most important assets of any organization is its information “

-Ralph Kimball author of the 3rd edition of the Data Warehouse toolkit [4]

What is a Data warehouse?

According to Oracle, one of the biggest companies on the data warehouse market: “A data warehouse is a type of data management system that is designed to enable and support business intelligence (BI) activities, especially analytics. [5]

The entire purpose of a data warehouse is to perform queries and analysis that contain large amounts of historical data. According to Oracle [5] usually the data within a data warehouse is from a wide range of sources for example log files and transaction applications. Due to data warehouse’s analytical possibilities, they are an excellent tool in giving valuable business insight from their data and improve the decision-making progress. Once it has a historical record the data is invaluable and Oracle quotes that a Data Warehouse can be an organization’s “single source of truth”.

A typical data warehouse generally includes the following elements:

- A relational database for data storage and management
- A solution for extraction, loading and transformation for data analysis
- Statistical analysis, reporting and other data mining capabilities
- Client analysis tools for visualization and a capability to present data for business users
- Some other more powerful analytics tools. [5]

2.3 Metadata

Metadata describes data, it has its own logical structure and generally it is hierarchical. Different information resources can be used as nodes of metadata with various links between these nodes and could be described as relationships between metadata. The changes of a node’s attribute may lead to changes in relationships between nodes. Metadata can be divided into application or core metadata in accordance with the system. Core metadata is used to describe the core resources and application metadata is the integrated metadata over the various information resources. [6]

As said by Ordonez the use of metadata over systems such as data warehouses cannot be undervalued. With the regular checking and historical analysis of metadata a database administrator can easily discover new links between different databases, tables and even schemas. These links could be used to determine which databases are more used, which are more critical and take appropriate steps in order to resolve them. Proper use of metadata could significantly improve the efficiency of a database administrator team as the amount of time and effort wasted in locating potential data could be reduced and the odds of finding valuable information is increased. [7]

An example of this is mentioned by C.Ordonez [7] where he says in a scientific federated database(explanation) containing a centralized metadata repository the user can find all documents referring to the same table representing some data set. The user knows the equations used by other researchers and can investigate who used tables with similar content across databases, enabling better interaction. The user is also able to understand which tasks are and are not commonly performed in the database system and increase understanding in how external programs process such data sets.

However, metadata presentation and quality are incredibly important as they have moderate influence on user attitudes towards the data. The attitude towards the metadata can be changed by the users perception of data quality, the effectiveness of the business intelligence tool used, and the quality of the training received. Collectively these factors have a rather strong influence on user attitudes towards this data. The attitude is important as the perceived usefulness and overall satisfaction increases the use of metadata. [1]

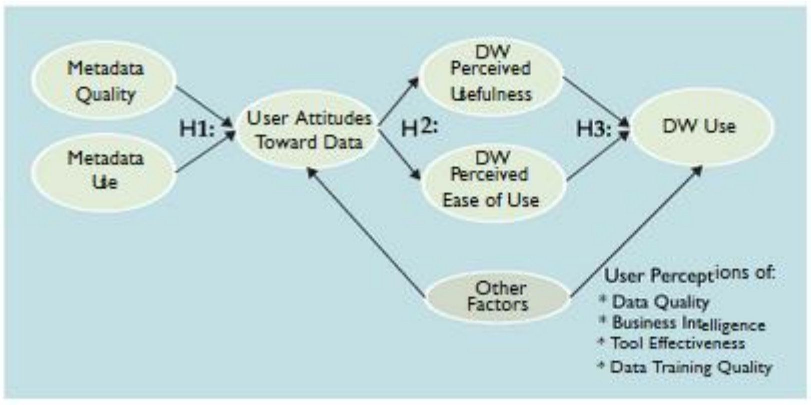


Figure 3: explanation of factors that impact end-user data warehouse use *[1]*

In the figure above we have a graph showing the factors that impact data warehouse use. It shows that Metadata quality and use, which accordingly impacts user attitudes toward data have a significant impact on DW use, showing the importance of metadata in data warehouses.

3 Analysis

This chapter discusses the problem at hand, provides the criteria for the solution, provides possible solutions and specific details about the solution.

3.1 The issue at hand:

Teradata SQL assistant is one of the main ways of executing and creating queries in the financial institution where the author of this paper works. Majority of its functionalities are divided into 3 windows. The query window, where the user creates and executes the query written in Teradata SQL. If the query is successful, an answer set window is shown with the results of the query (if the query does not encounter errors) and lastly there is also a history window that records your query executions whether it was successful or not. These windows, although providing good functionality, have very limited capability in showing technical metadata inside the databases and between the tables, therefore lacking an overall picture of the situation. The only way to gather good quality metadata is to create one's own SQL sentences that are usually long, can be quite complicated and can take up significant amount of time. Even creating and using these queries can be a long and cumbersome process, requiring significant time and other resources.

3.2 Criteria for the solution

In order to create a solution, there must be agreed criteria in what the solution must contain and provide:

1. It should be easy to use. The user should be able to easily access the solution and retrieve data with little to no effort.

2. It must show relevant metadata. The data shown must be useful to various parties in order to give the best understanding possible.

3. It must be secure. Security guidelines need to be kept in mind when creating the solution, even though we cannot see any critical or sensitive data. Only users with the allowed

access should be able to access the metadata and every measure should be taken to prevent use from nefarious third parties.

4. It needs to be simple. The access use and understanding should be as simple as possible, since the entire purpose of this problem is that currently finding metadata is a long and tedious process, so as a priority this point must be remembered.
5. It needs to be creatable. Even if the solution fulfils the four first criteria, it should be excluded, if there are significant legal, ethical or technological issues.

3.3 Possible Solutions:

As is the case with all technological problems, the amount of solutions and possibilities are endless. In this section each potential solution will be assessed and analysed.

1. An extension to the current Teradata SQL assistant.

   In theory an extension to the current Teradata program could be created for the purpose of obtaining the best metadata. If permission would be received from the Teradata company, this would be most optimal as everything would be contained to 1 program.

   The main issue with this solution is that obtaining permission or developing a platform is very unlikely, due to this case being specific to the financial institution the author is developing this for. The most that could be done is a communication to Teradata that this could be a feature in their program. Hence this does not fulfil the 5th criteria meaning that it is not possible to create this solution due to non-technological issues.

2. An independent program

   An independent program could achieve the requirements for the problem if all the criteria would be observed. The main issue with the program is that since due to high security requirements, installing a program in the financial institution is a long and tedious progress that requires approvals and permissions from various parties. Additionally, the security criteria would be difficult to implement into this as the implementation of full cybersecurity requirements would probably exceed the requirements for what is only a bachelor’s thesis.

3. A dashboard of a website

   A dashboard using the financial institutions built-in web-authentication would probably be the simplest for a user to access, use and understand. This is due to in-built web-authentication to the financial institutions work PC’s, making access significantly simpler. Additionally, once a security check for a user is added, this should fulfil the security criteria. Providing that this solution uses and shows relevant metadata as a dashboard on 1 page, all other criteria should be accounted for.

3.4 The Solution

This chapter contains the architecture and the components of the solution created. Different components and requirements have been highlighted that are vital to the system.

After the analysis of the 3 possible solutions, the decision was made to create the Teradata Common view as a dashboard. Additional incentive was given to the dashboard option when, the author was approached for it to be created as an extension to a new testing tool that was being developed for the financial institutions Data Warehouse.

3.4 Technical decisions

First as a disclaimer the author must mention that due to the dashboard being created as a part of an extension of a testing tool that was already in development, the author had to accept a few limitations to the back-end technologies used.

First since this thesis focuses heavily on Teradata the SQL language used was **Teradata SQL.** It is the main communication tool when connecting with Data Warehouses that use Teradata as its Relational Database Management System. [8]

The application uses **ASP.NET MVC** for its back end. It uses a Model-View-Controller (MVC) principle and it provides the creation of clean model classes that are easily bindable to a database. ASP.NET also provides support for many database engines. It must be mentioned that the author suggested creating the application in ASP.NET Core which is a new version of ASP.NET, but due to the testing tool development issues this was not possible. [9]

For front-end the main structural languages used were **HTML**, **CSS** with **Bootstrap 4** for easily being able to create and design websites and web-applications.

Majority of the front-end was written with **jQuery.** jQuery is a fast, small and feature-rich JavaScript library and it mainly used as the principal front-end programming language, performing all the fetch requests and DOM manipulation necessary. [10]

4. Development

This chapter describes the development process of the Teradata Common view. It will cover the topics of the appropriate queries to the Enterprise Data Warehouse (EDW), realizing the back end of the web application and the front-end dashboard. The development process is provided in the chronological sequence of development.

4.1 Realization of Queries for database

In total five queries were required to gather all the necessary data from the financial institutions Enterprise Data Warehouse (EDW). This sub-chapter will go in detail for each query to explain its logic and reasoning. The basic queries for these were created in association with the financial institutions developers and database administrators that were later modified and improved upon by the author. The chapter goes into detail for four out of five of these queries as these are critical for the dashboard.

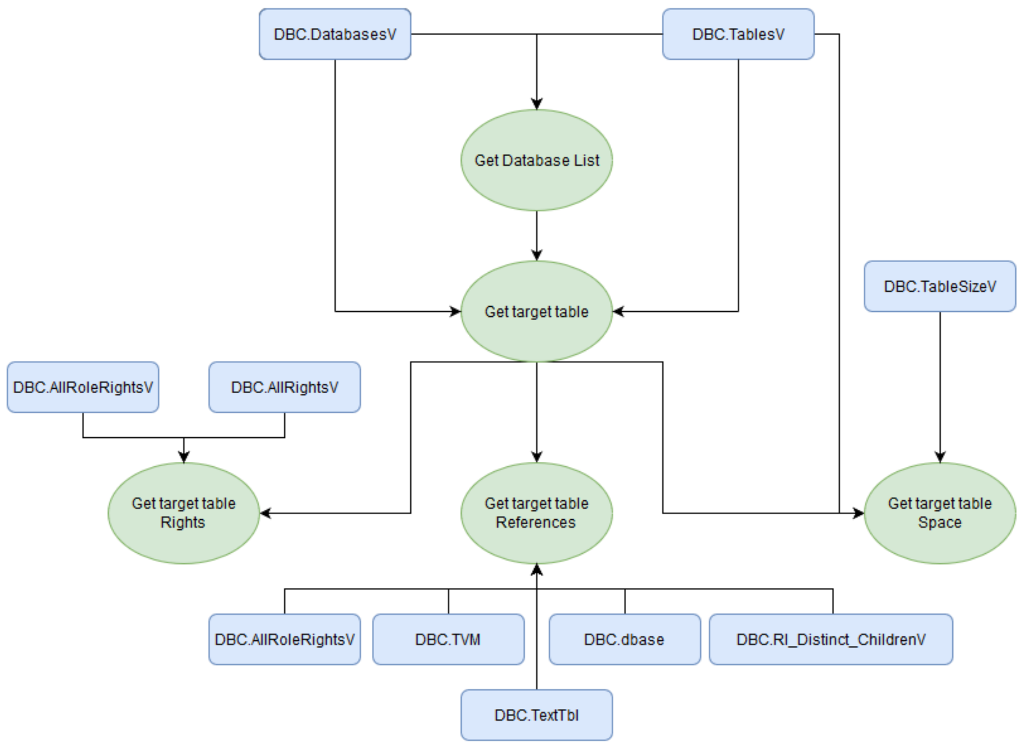


Figure 4 Query model

4.1.1 Databases query

This is the backbone query for the entire development. It returns to us all database names and the number of tables associated with each database.

The query consists of two following parts:

1. The first SELECT of the INNER JOIN goes into *DBC* which is the dictionary database for Teradata containing all data about objects within the data warehouse. Additionally, there are two where clauses present:
    i. First is a WHERE clause which eliminates all databases with the name *DBC* as DBC is the parent database from which we are querying the databases initially. This query also includes a count function that is aggregated through *DataBaseName*. This provides us with all the database names with their table counts. Due to the nature of Teradata this query returns only databases which have tables within them, since there is no necessity to show metadata info of databases which do not have any tables in them.
    ii. The second is to check that the *TableKind* or the type of the object is a table not a view or some other type.

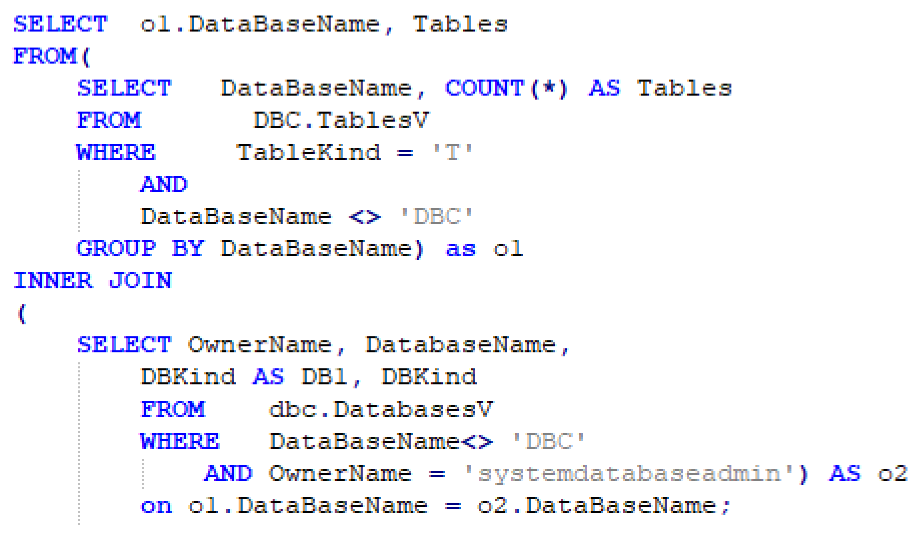
This sub-query returns all databases within DBC with their table counts.

2. The second SELECT of the INNER JOIN goes as well into DBC, but contains two WHERE statements:
    i. First one checks that the OwnerName or the creator of the table is the main administrator account. As a disclaimer I would like to add that *systemdatabaseadmin* is not the actual database administrators account name. This has been changed by the author due to security requirements of the financial institution. The importance of the WHERE clause is to not show any redundant tables which have been created by other users.

ii. The second WHERE statements follows the case with the first part of the database query this sub-query also includes the WHERE clause not returning the databases with the *DBC* parent database.

This sub-query returns all the databases with *systemdatabaseadmin* owner name or creator name.

The final part is INNER JOIN'ing the 2 parts of the queries together and showing the database names with their table count.



```
SELECT  o1.DataBaseName, Tables
FROM(
    SELECT   DataBaseName, COUNT(*) AS Tables
    FROM       DBC.TablesV
    WHERE     TableKind = 'T'
        AND
        DataBaseName <> 'DBC'
    GROUP BY DataBaseName) as o1
INNER JOIN
(
    SELECT OwnerName, DatabaseName,
        DBKind AS DB1, DBKind
        FROM    dbc.DatabasesV
        WHERE   DataBaseName<> 'DBC'
            AND OwnerName = 'systemdatabaseadmin') AS o2
        on o1.DataBaseName = o2.DataBaseName;
```

Figure 5 Get all the databases query.

4.1.2 The Table Space view

The subsequent query uses the database name and its subsequent table. The query is required to get three key principal parameters. *CurrentPerm, PeakPerm* and *SkewFactor*. *CurrentPerm* gets the current amount of space the table occupies. *PeakPerm* gets the total allocated space for the table or the maximum amount it can occupy. *SkewFactor* gets the skew factor for the current table. In chaper **2.1** it is explained that Teradata divides tables into Access Module Processors or AMP's. Skew Factor shows the disparity of occupied space within AMP's. This means that when the skew factor is large, some AMP's have a significant amount of rows within them, while other AMP's have very few. If the skew factor is small, the opposite is true as the difference in row amounts between AMP's is small.

The query does not have any sub-queries, however, it contains large amounts of various statements.

First the *DBC* database containing table sizes is locked in order to calculate the required parameters.

The SELECT statement uses TITLE to assign the table name from *DBC.TableSizeV* as the table name.

Then we have a CASE statement with series of letters being overwritten when the case is true. Essentially the *TableKind* or object type will be changed to optimize the query.

Each of the *TableKind*'s cases that are mentioned are replaced. Each value is referred to in **Appendix 1**. This is done to optimize the query and simplify the results. The *TableKind* value is then renamed as type.

Next the SUM method is used to show *CurrentPerm* and *PeakPerm*. Following this the *SkewFactor* is calculated with the following formula:

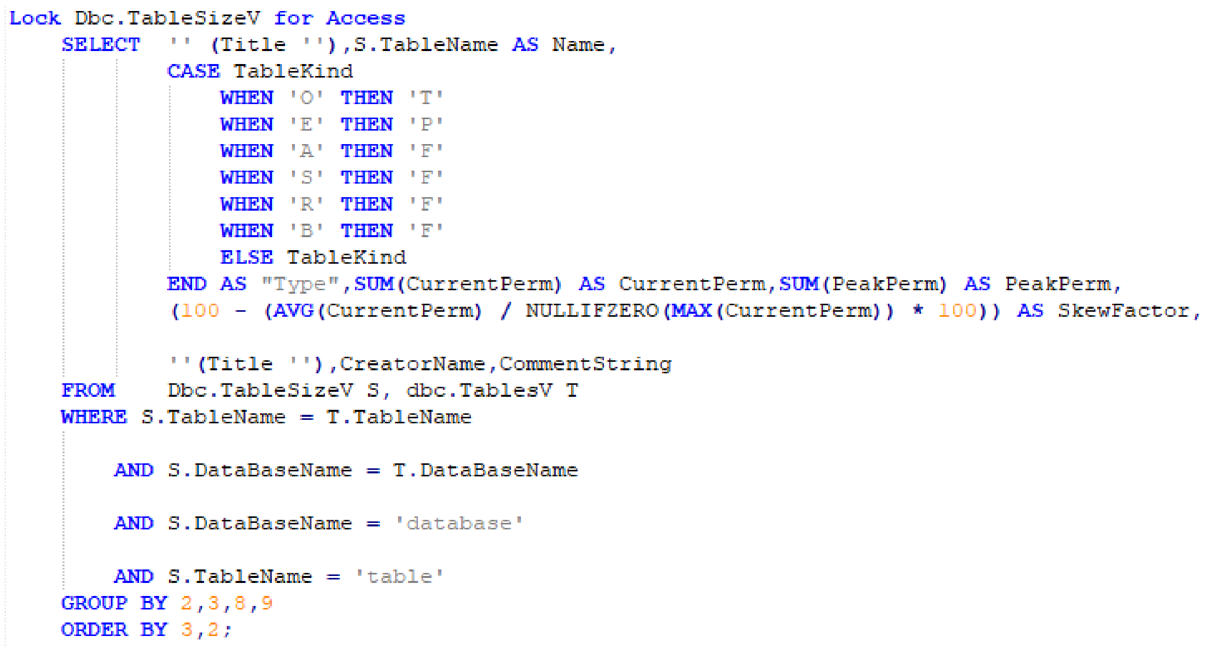
$$\frac{100 - (Average\ value\ of\ (\text{CurrentPerm})}{Maximum\ value\ of\ (CurrentPerm) * 100}$$

In the sql statement NULLIFZERO statement is added to prevent any errors caused by dividing with 0.

Afterwards there are four WHERE statements that are fulfilled. These are:

1. Equalizing the tables from *DBC.TableSizeV* the table size database with *DBC.TablesV* database which is the database containing all tables in the Data Warehouse.
2. Repeating the process of the first where statement, however with database names instead of table names.
3. The database name parameter is added here
4. The table name parameter is added here

Finally, the table is grouped and ordered by various additional columns that are aggregated in order to return the correct results.



```
Lock Dbc.TableSizeV for Access
    SELECT  '' (Title ''),S.TableName AS Name,
            CASE TableKind
                WHEN 'O' THEN 'T'
                WHEN 'E' THEN 'P'
                WHEN 'A' THEN 'F'
                WHEN 'S' THEN 'F'
                WHEN 'R' THEN 'F'
                WHEN 'B' THEN 'F'
                ELSE TableKind
            END AS "Type",SUM(CurrentPerm) AS CurrentPerm,SUM(PeakPerm) AS PeakPerm,
            (100 - (AVG(CurrentPerm) / NULLIFZERO(MAX(CurrentPerm)) * 100)) AS SkewFactor,

            ''(Title ''),CreatorName,CommentString
    FROM    Dbc.TableSizeV S, dbc.TablesV T
    WHERE S.TableName = T.TableName

        AND S.DataBaseName = T.DataBaseName

        AND S.DataBaseName = 'database'

        AND S.TableName = 'table'
    GROUP BY 2,3,8,9
    ORDER BY 3,2;
```

Figure 6 Get table space by table query

4.1.3 Get table rights by table query

The query described in this paragraph returns rows of rights granted for the queried table. Although it returns *Username, ColumnName, AccessRight, GrantAuthority, GrantorName* and *AllnessFlag*. Only three of the results are important enough metadata that should be presented. These are:

1. *UserName* the column for the user the access right is granted.
2. *AccessRight* the type of access right granted to the user.
3. *GrantorName* the user who granted the right to the user.

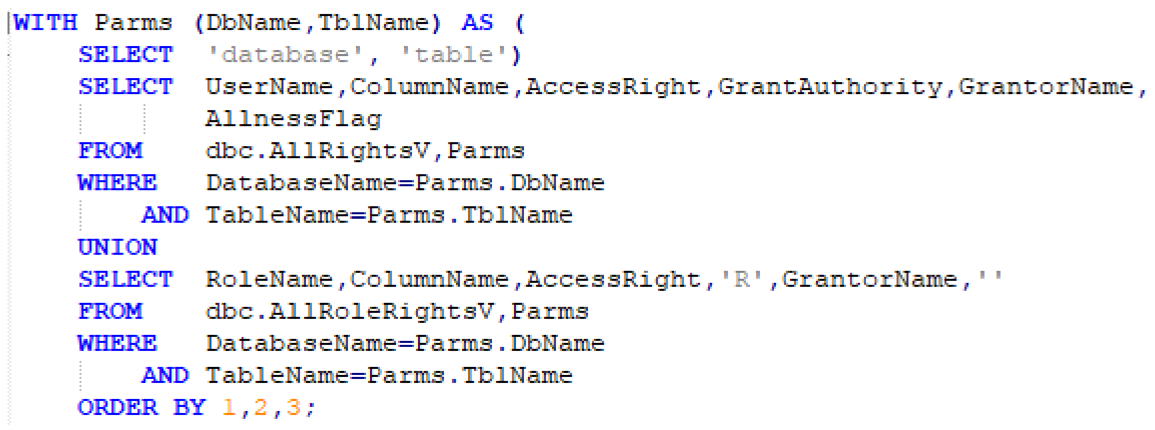
Initially the other columns were not added to the result as the values were entirely the same. However, the author decided not to remove the other column results since these might be necessary in a later development of the Teradata common view.

The query is built using the WITH and SELECT statements which include two parameters *DBName* and *Tb1Name*, which are the database name and table name respectively that is being queried for. It is divided into two sub-queries connected with a UNION statement.

The first sub-query gets all the column names mentioned in the beginning of this query description. Additionally, there are two WHERE statements which include the database name and table name parameters.

The second sub-query queries from *DBC.AllRoleRightsV* is the database containing all rights to different views, tables and other objects. Likewise, for the first sub-query it contains the two parameters for the database and table name. The sub-queries result is also ordered by the first three columns of the result.

Finally, the two queries are UNION'ed and the result is returned.



```
WITH Parms (DbName,TblName) AS (
    SELECT  'database', 'table')
    SELECT  UserName,ColumnName,AccessRight,GrantAuthority,GrantorName,
            AllnessFlag
    FROM    dbc.AllRightsV,Parms
    WHERE   DatabaseName=Parms.DbName
        AND TableName=Parms.TblName
    UNION
    SELECT  RoleName,ColumnName,AccessRight,'R',GrantorName,''
    FROM    dbc.AllRoleRightsV,Parms
    WHERE   DatabaseName=Parms.DbName
        AND TableName=Parms.TblName
    ORDER BY 1,2,3;
```

Figure 7 Get rights by database and table name query

4.1.4 Get table references by table query

In order to get all the references associated with the queried table, the query below was created. This query returns 3 columns with multiple rows if they exist. The three parameters returned are:

1. DatabaseName – The name of the database where the queried table is referenced at.
2. TVMName – The name of the object where the queried table is referenced at.
3. Type – the type of object which references the queried table.

The query uses again the WITH statement to pass in the necessary database and table parameters.

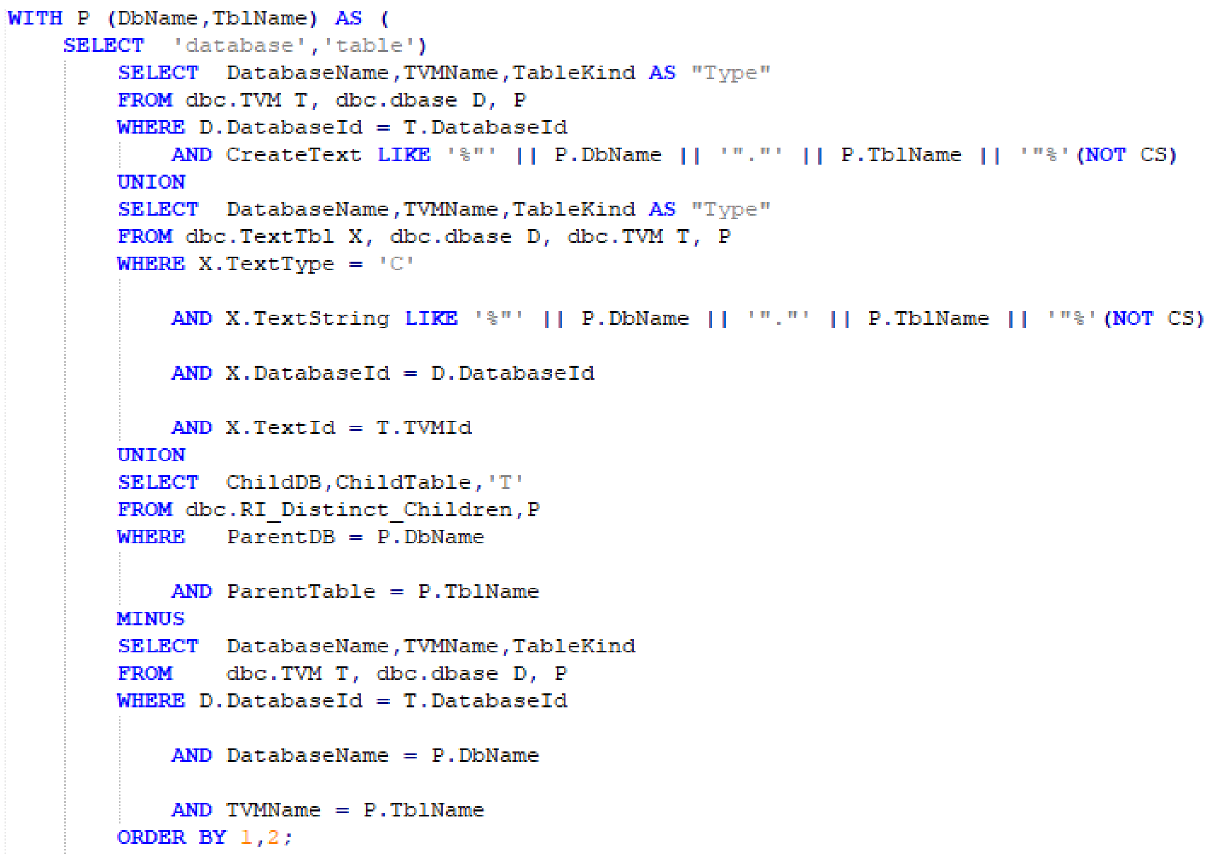
The query is divided into 4 sub-queries.

1. The first sub-query gets the database name, the table/view/object name and the type from *dbc.TVM* which is where all tables, views or other objects reside and from *dbc.dbase* which defines each database and user. The sub-query also contains a WHERE statement which equals both database ID’s. It then creates *CreateText* field which creates from the parameters in the form *database.table* to get the appropriate table.
2. The second sub-query gets as well gets the database name, the table/view/object name and the type from three database tables:
   i. *dbc.TextTbl* - which is the system table containing overflow DDL(data definition language) text,.
   ii. *dbc.dbase* - which is the system table that defines each database and user
   iii. dbc.TVM – which is where all the objects in the database

There are a few WHERE statements in this subquery, which are:

   a. First, *TextType* from *dbc.TextTbl* equaling *C* meaning text was created not requested
   b. Second, a *TextString* from *dbc.TextTbl* is looked for containing the queried for database and its table in the form *DataBaseName.TableName*.
   c. Third, the database ID’s from *dbc.dbase* and *dbc.TVM* are compared to in order to get the same result between them
   d. Finally, the Text ID’st from both *dbc.TextTbl* and *dbc.TVM* are equaled.
3. The third sub-query queries the child database and tables from *DBC.RI_Distinct_Children* which is a view that provides information about tables in child-parent order without duplicates caused from multiple foreign keys. This query contains two WHERE statements which equal to the queried for database name and its respective table name.
4. The final sub-query queries into *dbc.TVM* and *dbc.dbase* to make sure that the database ID’s, queried for database name and object name are the same.

The result is achieved by UNION'ing the first and the second sub-query with the result of the third and fourth sub-query's MINUS or every row that existed in the third sub-query, but not in the fourth. It is then ordered by *DatabaseName and TVMName* alphabetically.



```
WITH P (DbName,TblName) AS (
    SELECT  'database','table')
        SELECT  DatabaseName,TVMName,TableKind AS "Type"
        FROM dbc.TVM T, dbc.dbase D, P
        WHERE D.DatabaseId = T.DatabaseId
            AND CreateText LIKE '%"' || P.DbName || '"."' || P.TblName || '"%'(NOT CS)
        UNION
        SELECT  DatabaseName,TVMName,TableKind AS "Type"
        FROM dbc.TextTbl X, dbc.dbase D, dbc.TVM T, P
        WHERE X.TextType = 'C'

            AND X.TextString LIKE '%"' || P.DbName || '"."' || P.TblName || '"%'(NOT CS)

            AND X.DatabaseId = D.DatabaseId

            AND X.TextId = T.TVMId
        UNION
        SELECT  ChildDB,ChildTable,'T'
        FROM dbc.RI_Distinct_Children,P
        WHERE   ParentDB = P.DbName

            AND ParentTable = P.TblName
        MINUS
        SELECT  DatabaseName,TVMName,TableKind
        FROM    dbc.TVM T, dbc.dbase D, P
        WHERE D.DatabaseId = T.DatabaseId

            AND DatabaseName = P.DbName

            AND TVMName = P.TblName
        ORDER BY 1,2;
```

Figure 8 Get references by table query

4.2 Common-view Back-end

This chapter details the details of the back-end development process of this thesis.

The back end is created in **ASP .NET MVC** which is an open-source software from Microsoft used for web development. It combines features of MVC(Model-View-Controller) in order to promote agile development [7]

The back end is divided into four models, one view and two controllers.

The four models used are:

TDDatabase – The following model contains the database name, a list of tables the database contains and an *IsOpen* boolean which is explained **in chapter 4.3**.

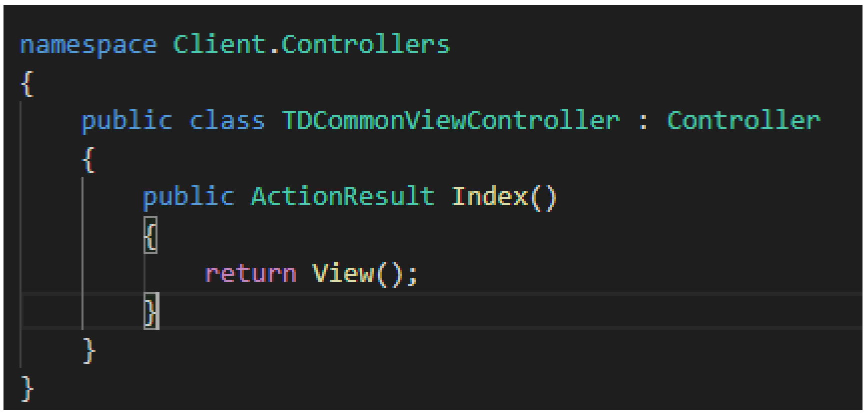
1. *TDTable* - This model has its parent database name, the table name, it is column count and an *IsOpen* Boolean which is used in the front-end layer and explained more in detail in **chapter 4.3**.

2. *TDRights* – the following model contains the *Username*, *RightType* and *GrantorName* arguments that are gotten from the get rights query by table in **chapter 4.2.4.**

3. *TDReference* – the model here contains *DataBaseName*, *TVMName* and *Type* arguments gotten from the get references query by table in **chapter 4.2.5**

The view will be described in the Front-End **Chapter 4.3**

This thesis contains two controllers. First is a basic ASP .NET controller in the client returning a *View()* or the index page referenced to.



```
namespace Client.Controllers
{
    public class TDCommonViewController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

Figure 9 ASP .NET Client returning a View

The second controller, the *TDCommonViewApiController* is an API controller which manages all the queries to the data warehouse and returns the result. The goal of this API controller is to perform the necessary GET requests in order to provide all the data from the queries.

It includes a method for each query hence there are five methods in total.

Throughout the *TDCommonViewApiController* the methods use one built-in classes called *DbUtility* and two classes provided by Teradata called *DbCommand* and *DbConnection. DbC*onnection creates the connection to the data warehouse which is saved by *DbUtility*. Then the query is prepared using *DbCommand* and the *DButiltity.GenerateSQL* method and finally is its executed using the *ExecuteReader* method and saved in the variable *reader.*

GetDatabases() – this method executes the query in **Chapter 4.2.1** and creates and returns an *TDdatabase* list named *databases.* Initially the query is executed using the three database connection classes and saved in the variable *reader*.

Then an empty list called *databases* is initialised and while loop is created that runs if *reader* has rows to read. For each new row, a new *TDDatabase* object is created with only the *Name* field being filled with the query result since arguments *IsOpen* and *Tables* are used in the front-end part of the development. *IsOpen* is used to check if the database needs to have its tables rendered and *Tables* are the databases child tables.

GetTables() – The following method executes a query to get tables with the *database* parameter using the *DbUtility* class and returns a list containing *TDTable* objects. Then a list called *tables* of *TDTable* objects is initialized. Then a while loop runs if there are available tables and *TDtable* objects are added to *tables* with the table name and database added to the class. Argument *ColumnCount* is also set, however this is not used in the scope of this thesis but kept in case of need for future developments.

GetTableSpace() – Here with two parameters *database* and *table* the table space query in **Chapter 4.2.3** is executed. The method returns a list of three elements containing the necessary results. However, the difference with the query being executed is that the author uses *((char)34)* to insert the double quote (“) symbol without creating errors. It again uses the *DbUtility* class to execute the query and the three datapoints are added to *tableSpace* list which is then returned.

GetRights() – This method uses the *database* and *table* parameters to execute the table rights query in **Chapter 4.2.4**. The method returns a list of *TDRights* objects. After being executed using the *DbUtility* class the *tableRights* list is created to which each table right row is added as an *TDRights* object containing the username, access right and the grantor name of the target table.

GetReferences() – The get references method gets all the references of the necessary table using the query in **Chapter 4.2.5**. It uses the *database* and *table* parameters to get a list of *TDReference objects*. The query was required significant use of the *((char)34)* method, which was necessary for execution. Using *DbUtility* the *tableReferences* list was initialized and a while loop containing rows of references added *TDReference objects* if they exist. The *tableReferences* list is then returned.

4.3 Front-end

This sub-chapter describes the development and creation of the front-end.

4.3.1 Technologies used

The main technologies used were **Jquery**, **Html, CSS** and **Bootstrap 4**. **Jquery** is a Javascript library that follows the moto "Write less, do more". It is a fast, small and a feature-rich JavaScript library. It includes html document traversal, manipulation, event handling, animation, Ajax and has an easy-to-use API that works across multiple browsers [10]

HTML (HyperText Markup Language) is the basic block in the Web. Defines meaning and web content and is used in addition with CSS and JavaScript to create websites. [11]

CSS (Cascading Style Sheets) is a stylesheet language which describes the presentation a document in HTML or XML. It describes how elements are rendered on screen, paper or in other media [12]

Bootstrap is a CSS framework that allows to quickly design and customize responsive web sites. It is the most popular front-end open source toolkit with powerful JavaScript plugins [13]

4.3.2 HTML structure description

The dashboard of this thesis is divided into half vertically. The left part contains all the databases and if clicked on its child tables. The Databases and their child tables have been made scrollable with CSS in order to increase ease of use.

The right side of the page is reserved for the Tablespace, Table Rights and Table References views. The three views are also created as tables with their respective column names and data (when data is fetched with jQuery).

4.3.3 Data fetch and DOM element manipulation with jQuery

The page is first initialized with all its html elements including a pre-built structure for the necessary views which are hidden. A fetch request is first sent out to get all the appropriate databases from the data warehouse which are then rendered in an unordered list. The left side of the page initially contains a message telling the user to click on a table to see the data.

The databases are rendered clickable so if clicked the child tables are fetched and then rendered as a sub-unordered list underneath the parent database.

When the necessary table has been selected three fetch requests to get the Table Space, Table Rights and Table References have been sent to each of these are rendered in a table form. Since Table Rights and references contain multiple rows of data the data fields have been made scrollable using CSS. If the table is clicked again the data will disappear with a message showing that to get data click a table in a database.

The dashboard has been built in a way that refreshes the results every time its closed and opened, to provide up-to-date metadata. The data model of the entire process can be found in **Figure 10**

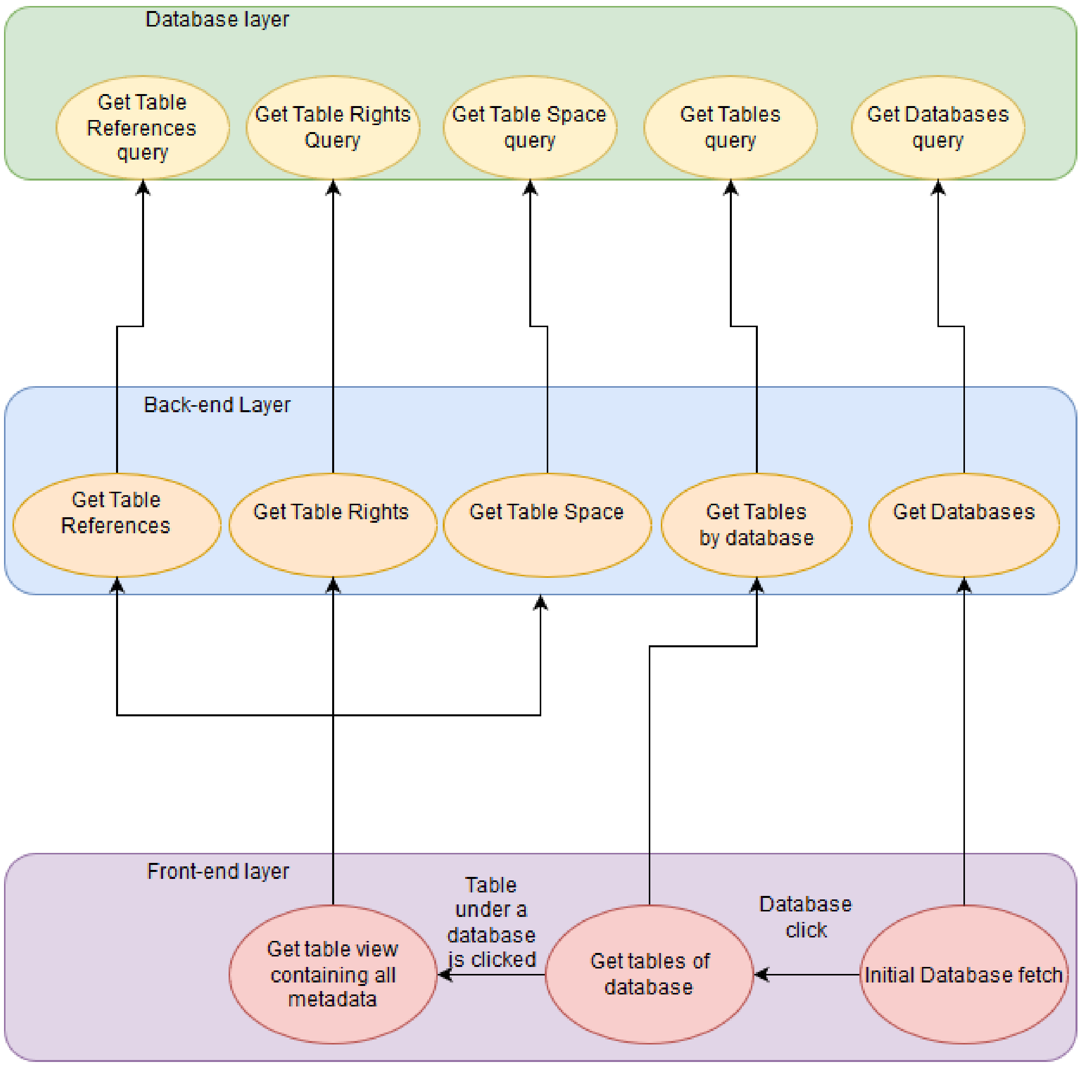
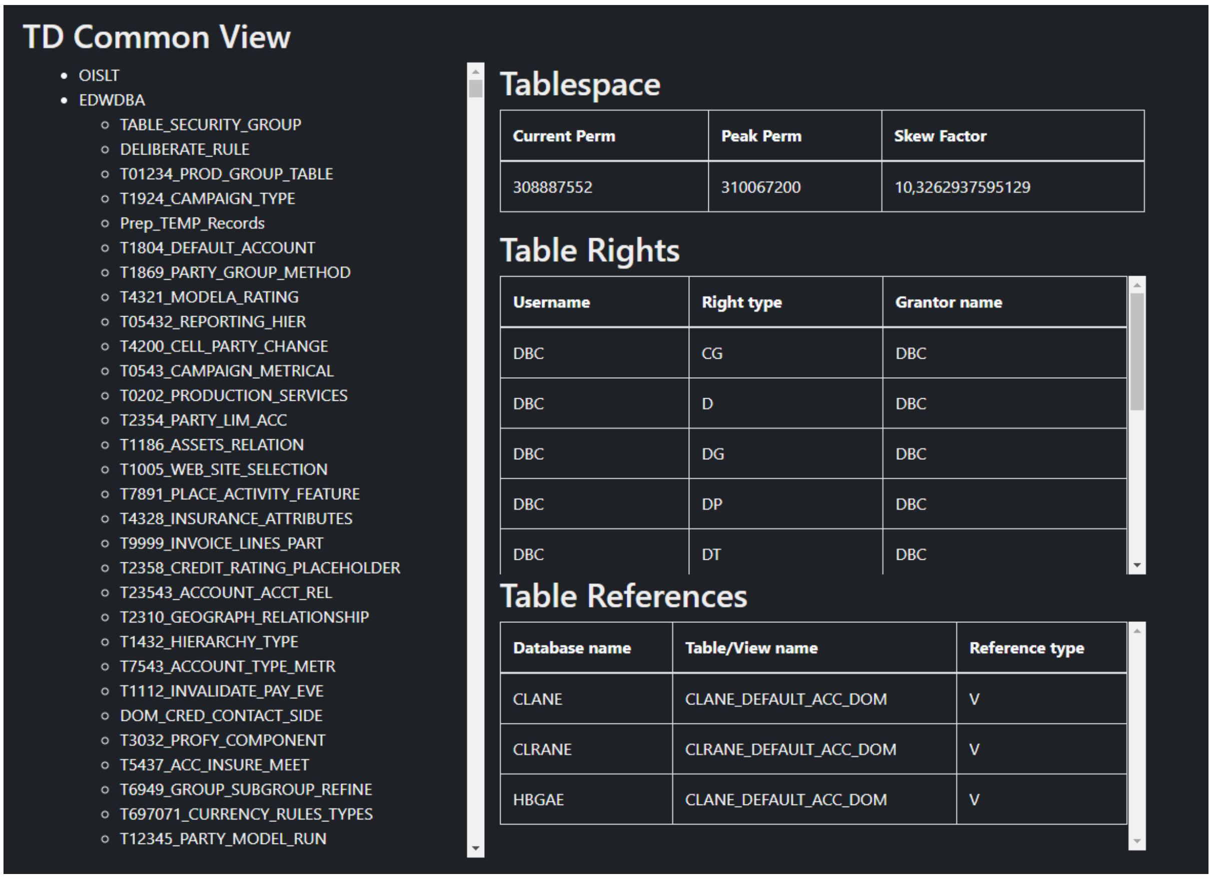


Figure 10 Data model of the application

4.4 Result

The result shown is a redacted version of the final metadata common view. All the database names and table names were changed as requested by the financial institution. The dashboard can be summed up in three simple pictures. Initially when the page is opened the user is greeted with the initial view asking for a table to be opened to see additional data as is shown in **Figure 11**. In the following figure it can be noted that the *T1804_DEFAULT_ACCOUNT* table still has some space but can run out soon if populated with too much data. In the rights table access rights from the Teradata dictionary database *DBC* can be seen. With the few references this table has it is not too in demand, but still any changes must be considered with the referenced tables/views.



TD Common View

- OISLT
- EDWDBA
  - TABLE_SECURITY_GROUP
  - DELIBERATE_RULE
  - T01234_PROD_GROUP_TABLE
  - T1924_CAMPAIGN_TYPE
  - Prep_TEMP_Records
  - T1804_DEFAULT_ACCOUNT
  - T1869_PARTY_GROUP_METHOD
  - T4321_MODELA_RATING
  - T05432_REPORTING_HIER
  - T4200_CELL_PARTY_CHANGE
  - T0543_CAMPAIGN_METRICAL
  - T0202_PRODUCTION_SERVICES
  - T2354_PARTY_LIM_ACC
  - T1186_ASSETS_RELATION
  - T1005_WEB_SITE_SELECTION
  - T7891_PLACE_ACTIVITY_FEATURE
  - T4328_INSURANCE_ATTRIBUTES
  - T9999_INVOICE_LINES_PART
  - T2358_CREDIT_RATING_PLACEHOLDER
  - T23543_ACCOUNT_ACCT_REL
  - T2310_GEOGRAPH_RELATIONSHIP
  - T1432_HIERARCHY_TYPE
  - T7543_ACCOUNT_TYPE_METR
  - T1112_INVALIDATE_PAY_EVE
  - DOM_CRED_CONTACT_SIDE
  - T3032_PROFY_COMPONENT
  - T5437_ACC_INSURE_MEET
  - T6949_GROUP_SUBGROUP_REFINE
  - T697071_CURRENCY_RULES_TYPES
  - T12345_PARTY_MODEL_RUN

Tablespace

Current Perm	Peak Perm	Skew Factor
308887552	310067200	10,3262937595129

Table Rights

Username	Right type	Grantor name
DBC	CG	DBC
DBC	D	DBC
DBC	DG	DBC
DBC	DP	DBC
DBC	DT	DBC

Table References

Database name	Table/View name	Reference type
CLANE	CLANE_DEFAULT_ACC_DOM	V
CLRANE	CLRANE_DEFAULT_ACC_DOM	V
HBGAE	CLANE_DEFAULT_ACC_DOM	V

Figure 11 Opened table showing *T1804_DEFAULT_ACCOUNT* data with changed database and table names by author.

This chapter discusses the possible and necessary future developments together with a conclusion.

4.5 Possible future developments

The primary future development on this dashboard is to add more metadata. Other necessary metadata information could be shown, for example: total space in the database or ordered tables according to data usage or skew factor. Additionally, a historical view could be created to show the changes over a period.

Additional clarity could be added as currently the final view does not show which table data is rendered and overall design could be improved.

Finally, an automatic refresh of the table data could be added to make the user experience more optimal.

5 Conclusion

The goal of this thesis was to create a Teradata metadata visualisation tool that allows users to access key table metadata with ease. This was done to improve metadata access in the financial institution's department where the author works at. This tool had to be easy to use and show relevant information rapidly.

The possible solutions for this were a Teradata SQL program extension, an independent program or a web-application. The author chose to create a web-application as the author was offered to create this dashboard as an extension of a testing tool that was in development. The tool ended up being called Teradata Common View, it was built as an ASP .NET MVC web-application in association with Teradata SQL queries.

The main problems of this thesis were in designing the queries, particularly creating a query for getting the initial key databases of the data warehouse. More precisely the part which filters out the databases which have no tables in them. This required clever use of the COUNT statement with additional appropriate thesis. Additionally, the entire get table references query was rather challenging to design in order to get correct results.

Teradata Common View allows users to access used table space, allocated table space and the skew factor with ease. It gives an overview of all the tables given rights to the target table and shows all the views and tables in which a target table was referenced in. The tool is easy to use and gets relevant information rapidly. Considering that beforehand this info had to be queried by the long, challenging and cumbersome queries, the Teradata Common View offers a much better solution to this issue.

Possible future improvements for Teradata Common View is first, to add more relevant metadata. It is important that these additional metrics are carefully chosen in order not to overpopulate the dashboard. Additionally, a historical view could be created to recognize historical patterns and finally additional clarity to the dashboard could be added.

Bibliography

[1] A. Taylor, N. Foshay and A. Mukherjee, “DOES DATA WAREHOUSE END-USER METADATA ADD VALUE?,” in *Communications of the ACM*, 2007.

[2] N. Tirs, “Teradata Common view code,” [Online]. Available: https://github.com/nillu21/TeradataCommonView.

[3] T. Coffing and Nolander.L, Teradata for executives, Coffing Publishing, 2013.

[4] Kimball.R and M. Ross, The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modelling, 3rd edition, Wiley, 2013.

[5] “What is a Data Warehouse?,” [Online]. Available: https://www.oracle.com/database/what-is-a-data-warehouse/. [Accessed 11 May 2020].

[6] W. Yanzhang, F. Tianwei, Y. Xin and W. Chunyou, “Metadata Management Model for Emergency Information Resources,” in *International Joint Conference on Web Intelligence and Intelligent Agent Technology*, 2009.

[7] C. Ordonez, Z. Chen and J. Garcia, “Metadata Management for Federated Databases,” in *Proceeding of the ACM first workshop on CyberInfrastructure: information management in eScience*, New York, 2007.

[8] “TERADATA Database SQL documentation,” Teradata, [Online]. Available: https://docs.teradata.com/reader/1DcoER_KpnGTfgPinRAFUw/URMgPkXpBvFL~jOlmMLmkQ. [Accessed 9 May 2020].

[9] “ASP .NET MVC Pattern: .NET,” Microsoft, [Online]. Available: https://dotnet.microsoft.com/apps/aspnet/mvc. [Accessed 12 May 2020].

[10] “What is jQery?,” JS Foundation, [Online]. Available: https://jquery.com/. [Accessed 9 May 2020].

[11] “HTML: Hypertext Markup Language,” Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML. [Accessed 13 May 2020].

[12] “What is CSS?,” Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS. [Accessed 14 May 2020].

[13] “Getting started with Bootstrap,” Boostrap team, [Online]. Available: https://getbootstrap.com/docs/4.5/getting-started/introduction/. [Accessed 14 May 2020].

Appendix 1 DBC.TableKind Column values

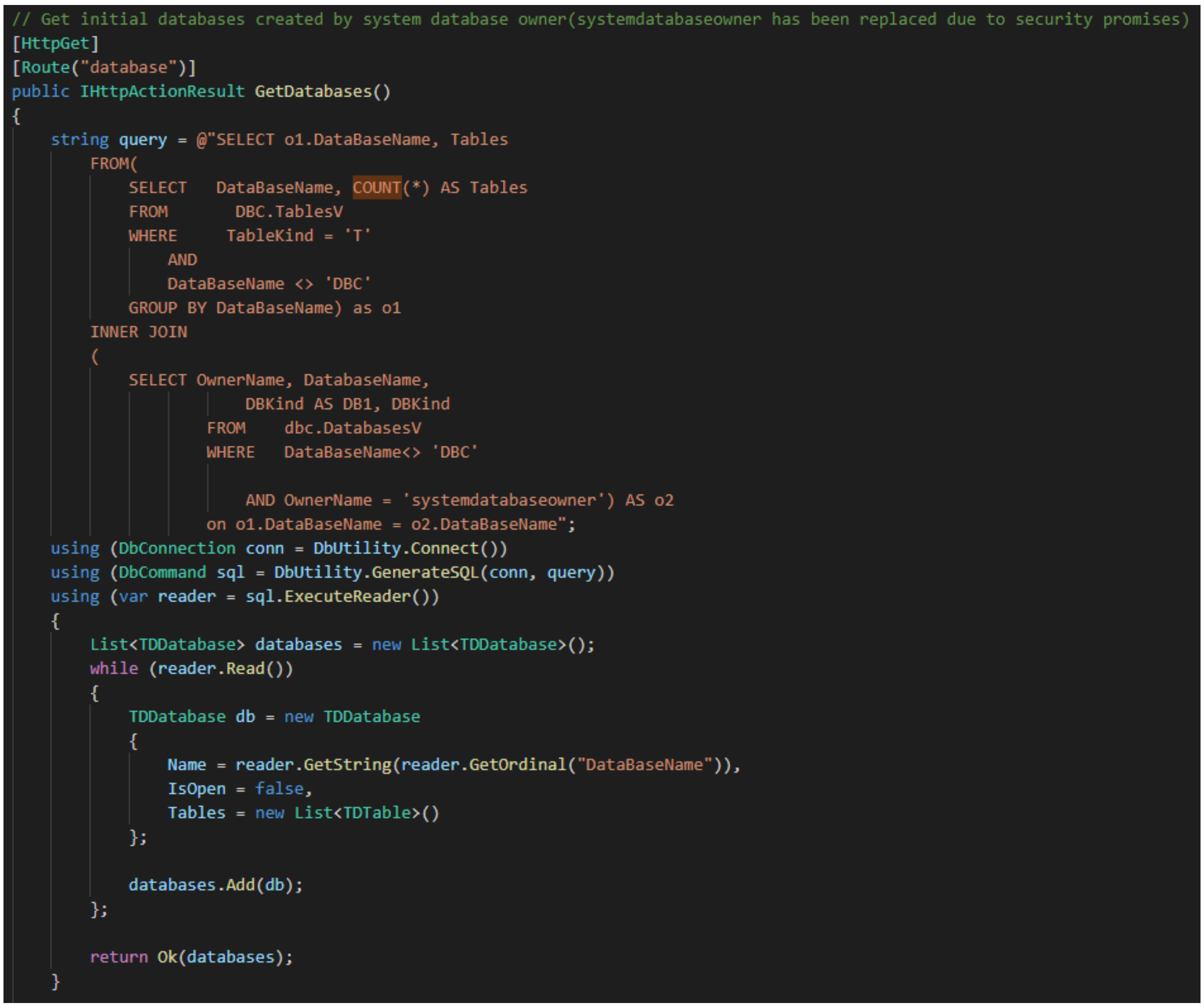
TableKind Column values[1]:

Value	Description
A	Aggregate function
B	Combined aggregate and ordered analytical function
C	Table operator parser contract function
D	JAR
E	External stored procedure
F	Standard function
G	Trigger
H	Instance or constructor method
I	Join index
J	Journal
K	Foreign server object. **Note:** K is supported on the Teradata-to-Hadoop connector only
L	User-defined table operator

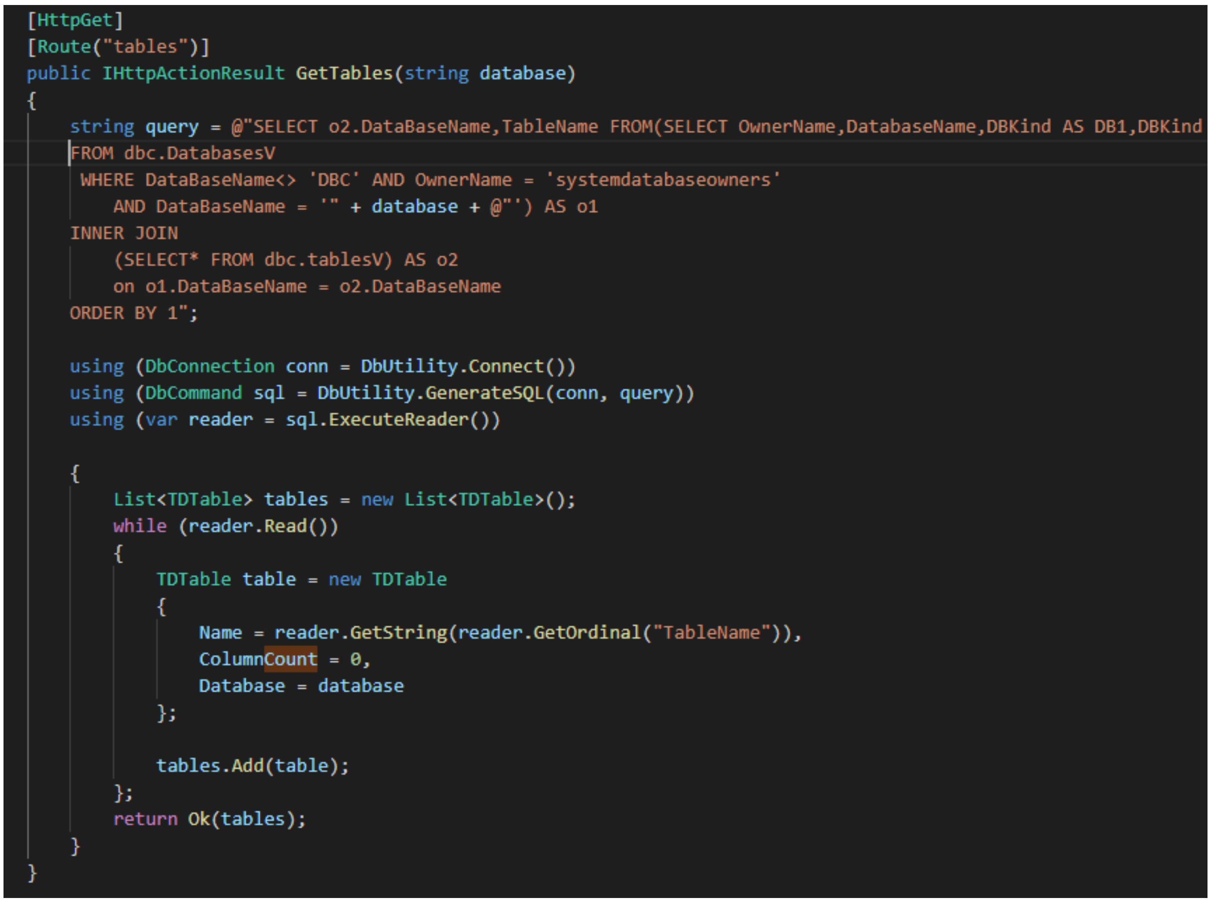
[1] https://docs.teradata.com/reader/oiS9ixs2ypIQvjTUOJfgoA/MhLWBwDX7EKjxwM83TD5bA

M	Macro
N	Hash index
O	Table with no primary index and no partitioning
P	Stored Procedure
Q	Queue table
R	Table function
S	Ordered analytical function
T	Table with a primary index or primary AMP index, partitioning, or both. Or a partitioned table with NoPI
U	User-defined type
V	View
X	Authorization
Y	GLOP set
Z	UIF

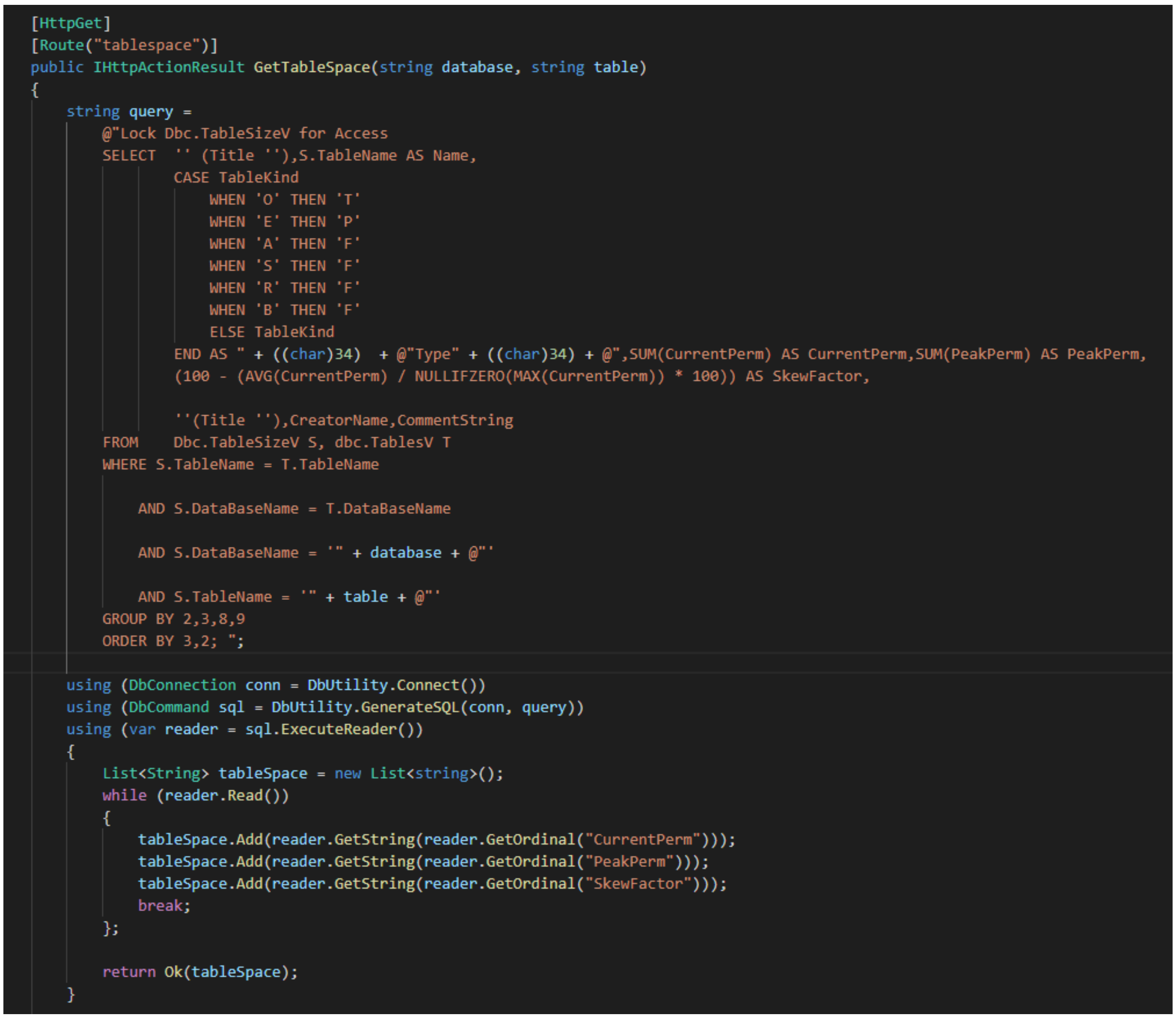
Appendix 2 Back-end of the application



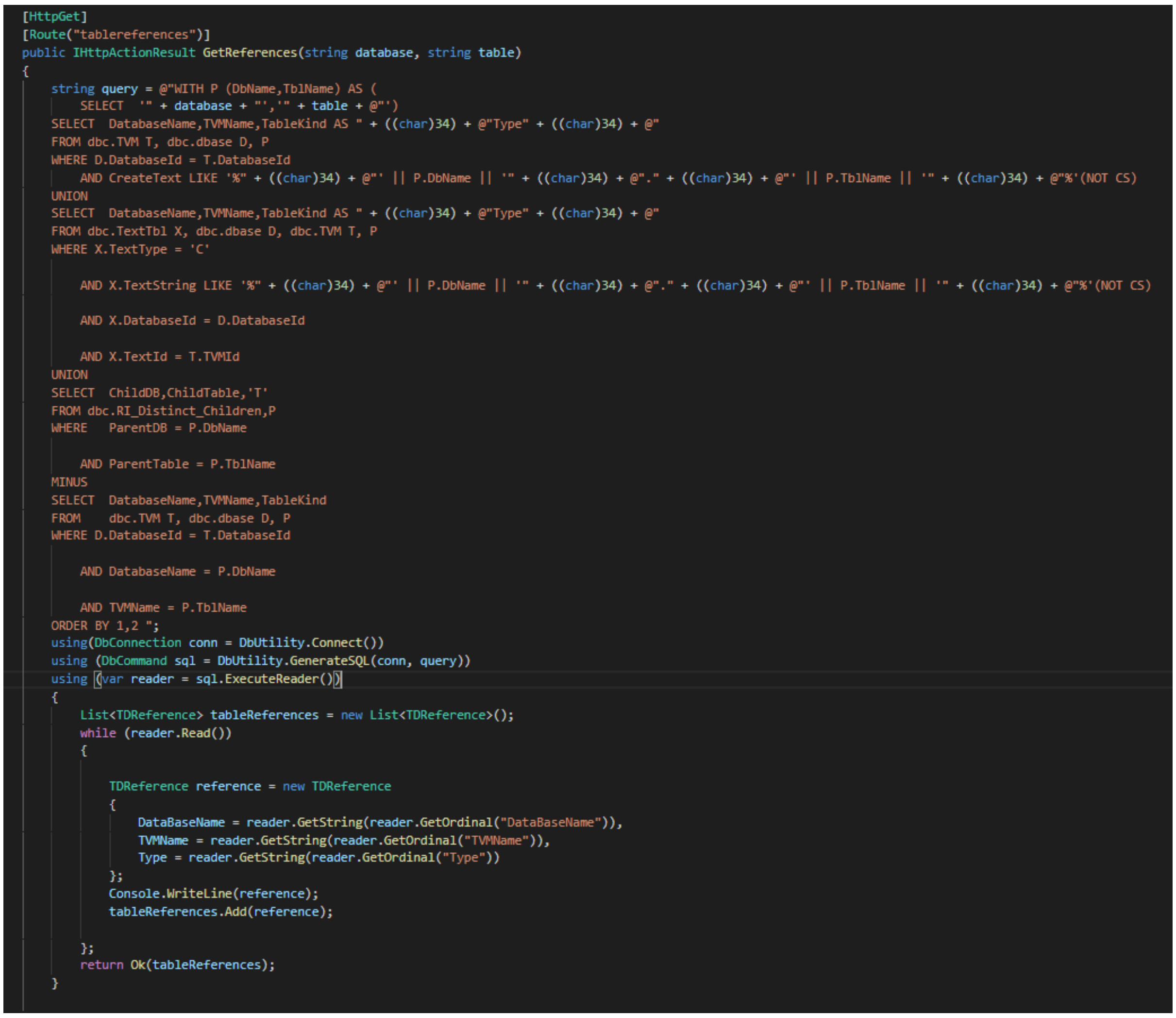
```
// Get initial databases created by system database owner(systemdatabaseowner has been replaced due to security promises)
[HttpGet]
[Route("database")]
public IHttpActionResult GetDatabases()
{
    string query = @"SELECT o1.DataBaseName, Tables
        FROM(
            SELECT   DataBaseName, COUNT(*) AS Tables
            FROM       DBC.TablesV
            WHERE     TableKind = 'T'
                AND
                DataBaseName <> 'DBC'
            GROUP BY DataBaseName) as o1
        INNER JOIN
        (
            SELECT OwnerName, DatabaseName,
                        DBKind AS DB1, DBKind
                    FROM    dbc.DatabasesV
                    WHERE   DataBaseName<> 'DBC'

                        AND OwnerName = 'systemdatabaseowner') AS o2
                    on o1.DataBaseName = o2.DataBaseName";
    using (DbConnection conn = DbUtility.Connect())
    using (DbCommand sql = DbUtility.GenerateSQL(conn, query))
    using (var reader = sql.ExecuteReader())
    {
        List<TDDatabase> databases = new List<TDDatabase>();
        while (reader.Read())
        {
            TDDatabase db = new TDDatabase
            {
                Name = reader.GetString(reader.GetOrdinal("DataBaseName")),
                IsOpen = false,
                Tables = new List<TDTable>()
            };

            databases.Add(db);
        };

        return Ok(databases);
    }
```



```
[HttpGet]
[Route("tables")]
public IHttpActionResult GetTables(string database)
{
    string query = @"SELECT o2.DataBaseName,TableName FROM(SELECT OwnerName,DatabaseName,DBKind AS DB1,DBKind
    FROM dbc.DatabasesV
     WHERE DataBaseName<> 'DBC' AND OwnerName = 'systemdatabaseowners'
        AND DataBaseName = '" + database + @"') AS o1
    INNER JOIN
        (SELECT* FROM dbc.tablesV) AS o2
        on o1.DataBaseName = o2.DataBaseName
    ORDER BY 1";

    using (DbConnection conn = DbUtility.Connect())
    using (DbCommand sql = DbUtility.GenerateSQL(conn, query))
    using (var reader = sql.ExecuteReader())

    {
        List<TDTable> tables = new List<TDTable>();
        while (reader.Read())
        {
            TDTable table = new TDTable
            {
                Name = reader.GetString(reader.GetOrdinal("TableName")),
                ColumnCount = 0,
                Database = database
            };

            tables.Add(table);
        };
        return Ok(tables);
    }
}
```



```
[HttpGet]
[Route("tablespace")]
public IHttpActionResult GetTableSpace(string database, string table)
{
    string query =
        @"Lock Dbc.TableSizeV for Access
        SELECT  '' (Title ''),S.TableName AS Name,
                CASE TableKind
                    WHEN 'O' THEN 'T'
                    WHEN 'E' THEN 'P'
                    WHEN 'A' THEN 'F'
                    WHEN 'S' THEN 'F'
                    WHEN 'R' THEN 'F'
                    WHEN 'B' THEN 'F'
                    ELSE TableKind
                END AS " + ((char)34)  + @"Type" + ((char)34) + @",SUM(CurrentPerm) AS CurrentPerm,SUM(PeakPerm) AS PeakPerm,
                (100 - (AVG(CurrentPerm) / NULLIFZERO(MAX(CurrentPerm)) * 100)) AS SkewFactor,

                ''(Title ''),CreatorName,CommentString
        FROM    Dbc.TableSizeV S, dbc.TablesV T
        WHERE S.TableName = T.TableName

            AND S.DataBaseName = T.DataBaseName

            AND S.DataBaseName = '" + database + @"'

            AND S.TableName = '" + table + @"'
        GROUP BY 2,3,8,9
        ORDER BY 3,2; ";

    using (DbConnection conn = DbUtility.Connect())
    using (DbCommand sql = DbUtility.GenerateSQL(conn, query))
    using (var reader = sql.ExecuteReader())
    {
        List<String> tableSpace = new List<string>();
        while (reader.Read())
        {
            tableSpace.Add(reader.GetString(reader.GetOrdinal("CurrentPerm")));
            tableSpace.Add(reader.GetString(reader.GetOrdinal("PeakPerm")));
            tableSpace.Add(reader.GetString(reader.GetOrdinal("SkewFactor")));
            break;
        };

        return Ok(tableSpace);
    }
```



```
[HttpGet]
[Route("tablereferences")]
public IHttpActionResult GetReferences(string database, string table)
{
    string query = @"WITH P (DbName,TblName) AS (
        SELECT  '" + database + "','" + table + @"')
    SELECT  DatabaseName,TVMName,TableKind AS " + ((char)34) + @"Type" + ((char)34) + @"
    FROM dbc.TVM T, dbc.dbase D, P
    WHERE D.DatabaseId = T.DatabaseId
        AND CreateText LIKE '%" + ((char)34) + @"' || P.DbName || '" + ((char)34) + @"." + ((char)34) + @"' || P.TblName || '" + ((char)34) + @"%'(NOT CS)
    UNION
    SELECT  DatabaseName,TVMName,TableKind AS " + ((char)34) + @"Type" + ((char)34) + @"
    FROM dbc.TextTbl X, dbc.dbase D, dbc.TVM T, P
    WHERE X.TextType = 'C'

        AND X.TextString LIKE '%" + ((char)34) + @"' || P.DbName || '" + ((char)34) + @"." + ((char)34) + @"' || P.TblName || '" + ((char)34) + @"%'(NOT CS)

        AND X.DatabaseId = D.DatabaseId

        AND X.TextId = T.TVMId
    UNION
    SELECT  ChildDB,ChildTable,'T'
    FROM dbc.RI_Distinct_Children,P
    WHERE   ParentDB = P.DbName

        AND ParentTable = P.TblName
    MINUS
    SELECT  DatabaseName,TVMName,TableKind
    FROM    dbc.TVM T, dbc.dbase D, P
    WHERE D.DatabaseId = T.DatabaseId

        AND DatabaseName = P.DbName

        AND TVMName = P.TblName
    ORDER BY 1,2 ";
    using(DbConnection conn = DbUtility.Connect())
    using (DbCommand sql = DbUtility.GenerateSQL(conn, query))
    using (var reader = sql.ExecuteReader())
    {
        List<TDReference> tableReferences = new List<TDReference>();
        while (reader.Read())
        {

            TDReference reference = new TDReference
            {
                DataBaseName = reader.GetString(reader.GetOrdinal("DataBaseName")),
                TVMName = reader.GetString(reader.GetOrdinal("TVMName")),
                Type = reader.GetString(reader.GetOrdinal("Type"))
            };
            Console.WriteLine(reference);
            tableReferences.Add(reference);

        };
        return Ok(tableReferences);
    }
```