UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Data Science Curriculum

Roland Pajuleht

A comparative study of database schema on query performance in data warehousing

Master's Thesis (15 ECTS)

Supervisor(s): Eduard Ševtšenko, PhD

Tartu 2024

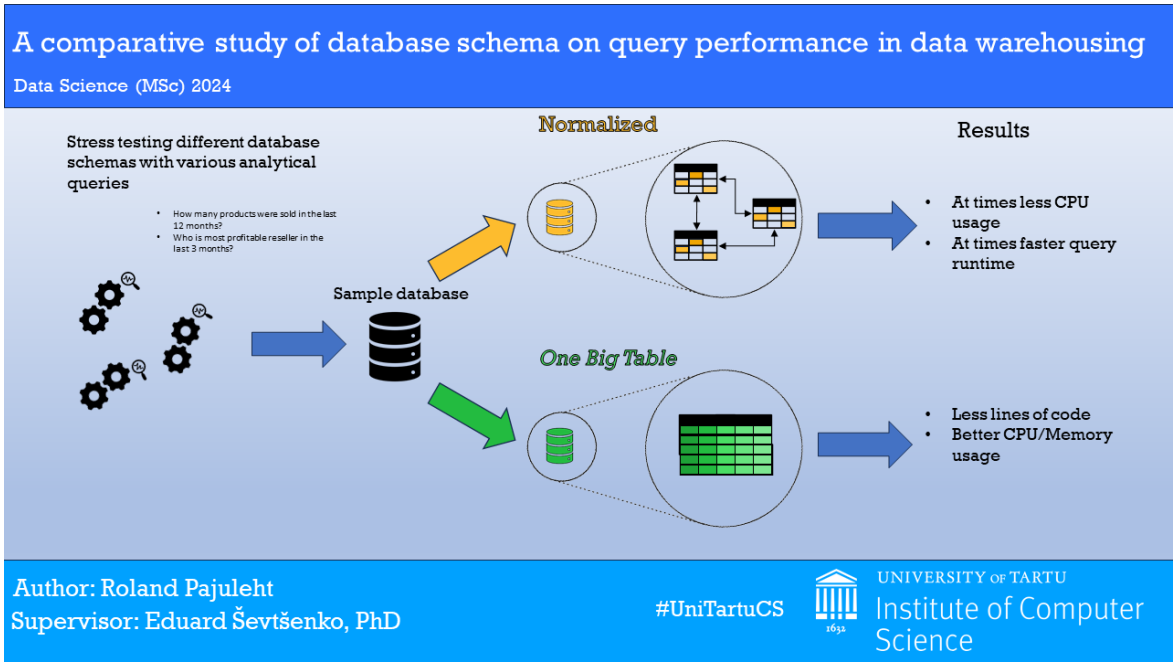
Abstract:

This paper compares differences in query performance and execution plans between two different approaches to data modeling. Dimensional modeling, a standard procedure for building data warehouses, is compared with a less standardized model that starts to emerge as a consequence when concrete data architectural procedures are not in place. Several analytical queries are run against a standard, normalized star schema and a table with a more relaxed form, often called One Big Table. It was found that while readability improved when constructing queries for the wide table, performance issues quickly emerged. When operating in traditional data warehouses, data engineers must adhere to established architectural practices in order to maintain an efficient database.

Keywords:
Data warehouse, OLAP, SQL query performance

CERCS: P175 Informatics, systems theory

Graphical abstract:



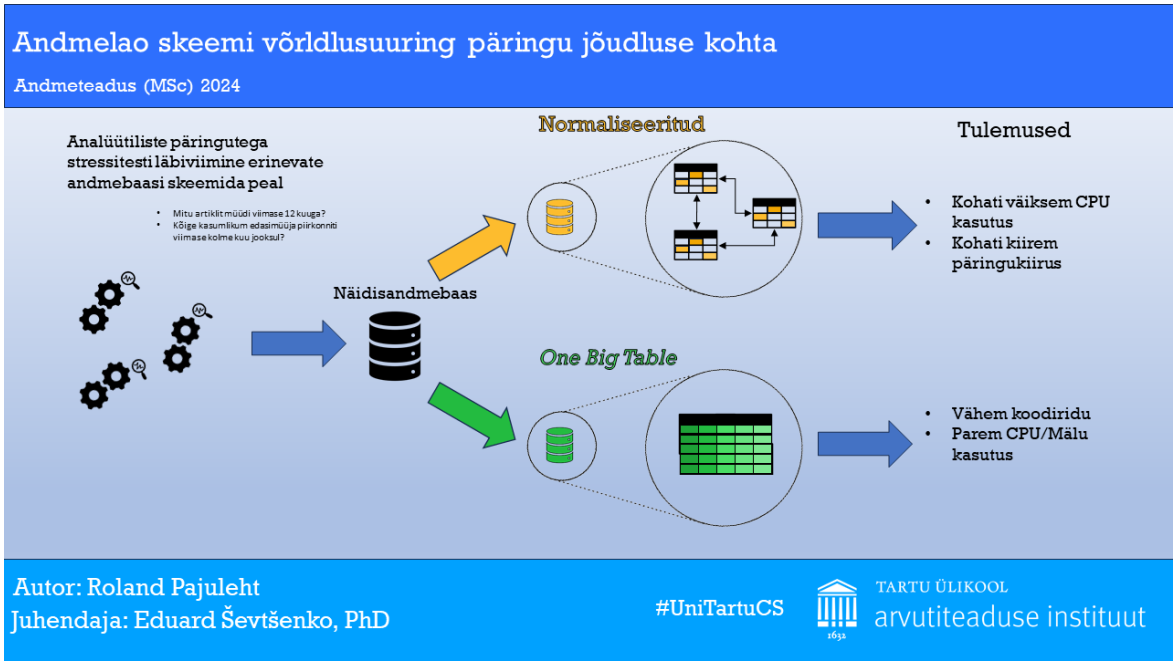
Lühikokkuvõte:

Käesolevas töös võrreldakse päringute jõudluse ja täitmisplaanide erinevusi kahe erineva andmemodelleerimise lähenemisviisi vahel. Mõõtmelist modelleerimist, mis on andmelao ehitamise üheks standardlahenduseks, võrreldakse vähem standardiseeritud mudeliga, mis hakkab tekkima, kui konkreetsed andmearhitektuuriprotseduurid ei ole paigas. Nende skeemide peal viiakse läbi mitmed analüütilised. Selgus, et kuigi loetavus laia tabeli puhul on parem, tekivad kiiresti jõudlusprobleemid. Traditsioonilistes andmeladudes töötades peavad andmetöötlejad järgima väljakujunenud arhitektuuripraktikaid, et hoida üleval tõhusat andmeladu.

Võtmesõnad:
Andmeladu, OLAP, SQL päringu jõudlus

CERCS: P175 Informaatika, süsteemiteooria

Visuaalne kokkuvõte:



Contents

Acronyms and abbreviations

Acronym/abbreviation	Full term
DBMS	**D**ata**b**ase **M**anagement **S**ystem
OLAP	**O**n**l**ine **A**nalytical **P**rocessing
DW/BI	**D**ata **W**arehousing / **B**usiness **I**ntelligence
OTB	**O**ne **B**ig **T**able
EDA	**E**xploratory **d**ata **a**nalysis
FOSS	**F**ree and **O**pen-**s**ource **s**oftware
SSMS	**S**QL **S**erver **M**anagement **S**tudio

1 Introduction

Numerous state and private Estonian organizations have data warehouses built in the pre-European Union times. Inevitably, the underlying data models in these warehouses have become obsolete, causing high requirements for maintenance and making it harder to adapt to new technologies and tools that today's data requires. Many of these organizations are in the process of updating their data models. This is no easy task, however, because of the unique nature of the organization, the amount and nature of data have to be considered.

In the rapidly evolving landscape of data management and analytics, the structure and organization of data within a data warehouse play a crucial role in determining query performance and overall system efficiency. As data volumes grow and analytical needs become more complex, selecting an appropriate data schema becomes even more critical. This thesis delves into a comparative analysis of two distinct data modeling approaches: dimensional modeling, specifically the star schema, and the "One Big Table" approach.

Nowadays, when data engineers are faced with a data ingestion problem, they have the option to rigidly model the data, or it is also possible to throw the data into a single wide table. A wide table is highly denormalized (very redundant in the data it holds) and a vast collection of many fields, typically created in a columnar database. Additionally, it is possible to simply store the data in the cloud. The popularity of the cloud means that storage is very cheap. It is cheaper to store data than to agonize over the optimum way to represent it in storage. Also, the popularity of nested data (JSON and similar) means schemas are flexible in source and analytical systems.

The first part introduces the problem, explains the concepts needed to answer the research question, sets the scope, and defines the limitations of such work. It also provides a review of the literature on the concepts introduced. The second part reviews benchmarks that have been done in the past that answer a similar research question, though in a much broader context. The third part explains the procedure for building up the necessary systems to answer the research question. It also describes the sample database in this paper, its properties, the selected database engine that will be used on this sample database, and arguments for focusing on specific parts of the database. The fourth part will take a look at the analytical queries and perform them on the schemas, compare performance and execution plans proposed by the engine, and look at ways to improve them. The final part discusses and concludes the findings.
The purpose of this paper is to provide data engineers who are in the process of building,

maintaining, evaluating, or improving their data warehouses, to become aware of the potential effects and trade-offs that might affect the performance of analytical queries that are performed on the warehouse. Also to help them consider some of the properties that are discussed in the paper in their own processes

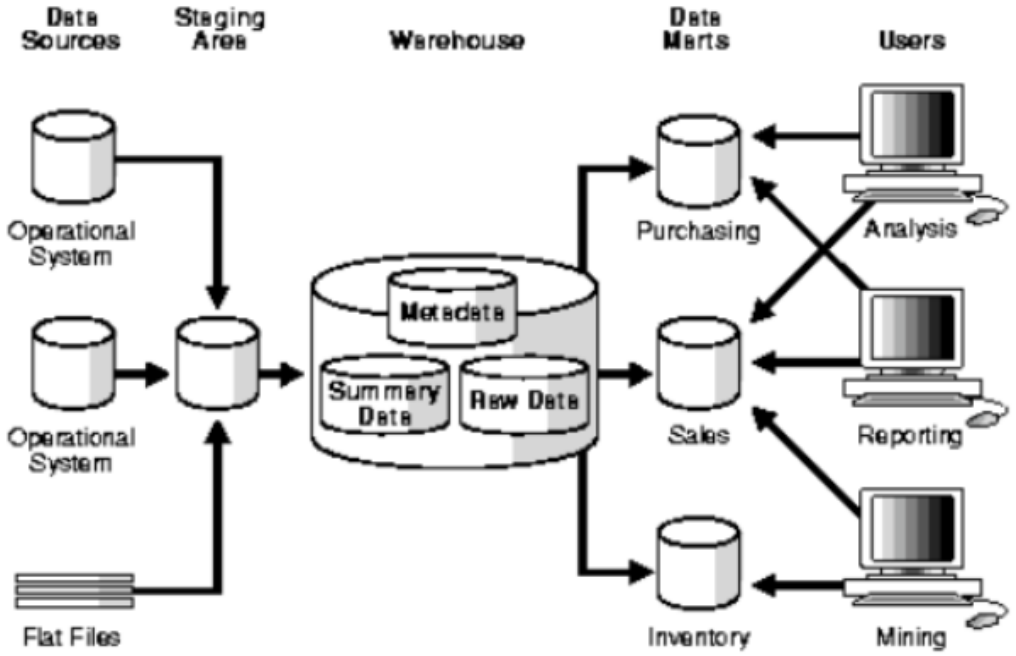
1.1 Data modeling

Data modeling has been a practice for decades in one form or another. For example, various types of normalization techniques have been used to model data since the early days of RDBMs; data warehousing modeling techniques have been around since at least the early 1990s and arguably longer. As pendulums in technology often go, data modeling became unfashionable in the early to mid-2010s. The rise of data lake 1.0, NoSQL, and BD systems allowed engineers to bypass traditional data modeling, sometimes for legitimate performance gains. Other times, the lack of rigorous data modeling created data swamps and lots of redundant, mismatched, or simply wrong data. (Reis & Housley, 2022) (Brackenbury, Liu, Mondal, & Elmore, 2018)

Nowadays, the pendulum seems to be swinging back toward data modeling. The growing popularity of data management (in particular, data governance and data quality) pushes the need for coherent business logic. The fast rise of data's prominence in companies creates a growing recognition that modeling is critical for realizing value at the higher levels of the Data Science Hierarchy of Needs pyramid[1].

[1] https://medium.com/@hugh_data_science/the-pyramid-of-data-needs-and-why-it-matters-for-your-career-b0f695c13f11

Figure 1. Typical data warehouse architecture. (Hariharaputran, 2010)



A data warehouse is a central data hub used for reporting and analysis. It is a collection of decision support technologies, aimed at enabling the knowledge worker (executive, manager, analyst) to make better and faster decisions (Reddy, Srinivasu, Rao, & Rikkula, 2010). Data in a data warehouse is typically highly formatted and structured for analytics use cases. It is among the oldest and most well-established data architectures. Traditionally, a data warehouse pulls data from application systems by using ETL. The extract phase pulls data from source systems. The transformation phase cleans and standardizes data, organizing and imposing business logic in a highly modeled form.

The tests conducted in this paper are only relevant to a specific domain in the database. A specific unit inside a data warehouse, focused on a single subject or line of business, is called a data mart.

1.1.1 Dimensional modeling

Traditionally, the two main approaches to data warehouse schema design have been dimensional modeling and One Big Table (OBT) format. Bill Inmon and Ralph Kimball developed two main approaches for dimensional modeling, respectively. Due to its ease of use and better performance, Kimball's dimensional modeling has prevailed as the most

popular one and is thus one of the main topics of this paper. A comparison of the models is described in the table below. Mary Breslin's article provides an excellent high-level comparison of the models. (Breslin, 2004)

	Inmon	Kimball
Methodology and architecture		
Overall approach	Top-down	Bottom-up
Architectural structure	Enterprisewide (atomic) data warehouse "feeds" departmental databases	Data marts model a single business process; enterprise consistency achieved through data bus and conformed dimensions
Complexity of the method	Quite complex	Fairly simple
Comparison with established development methodologies	Derived from the spiral methodology	Four-step process; a departure from RDBMS methods
Discussion of physical design	Fairly thorough	Fairly light
Data modeling		
Data orientation	Subject- or data-driven	Process oriented
Tools	Traditional (ERDs, DISs)	Dimensional modeling; a departure from relational modeling
End-user accessibility	Low	High
Philosophy		
Primary audience	IT professionals	End users
Place in the organization	Integral part of the Corporate Information Factory (CIF)	Transformer and retainer of operational data
Objective	Deliver a sound technical solution based on proven database methods and technologies	Deliver a solution that makes it easy for end users to directly query the data and still get reasonable response times

Table 1. Comparison of Essential Features of Inmon's and Kimball's Models.

The table above summarizes the basic properties of both models. As Breslin has characterized, Kimball's dimensional modeling is generally easier to use and implement than Inmon's model; this is also confirmed by the industry and academia by being the go-to model when implementing or teaching "traditional" databases.

Dimensional modeling, first developed by Ralph Kimball, is a widely adopted methodology for structuring data warehouses (Kimball, 1996). Kimball emphasizes simplicity and ease of use in his book, which encourages efficient query execution and reporting through well-defined dimensions and facts. The star schema is a data model that organizes data into fact tables linked to dimension tables, which helps to optimize query performance and provides a clear, intuitive structure for end-users.

1.1.2 OBT modelling

Denormalized database schemas, often called OBTs (One Big Table), not to be confused with Bigtables, which are different data storage mechanisms, are in terms of normalization on the other end of the spectrum. The strict modeling approaches previously described were developed when data warehouses were expensive, on-premises, and heavily resource-constrained with tightly coupled computing and storage. While batch data modeling has

traditionally been associated with these strict approaches, more relaxed approaches are becoming more common due to storage becoming cheaper and the ability to pay by byte when communicating transactions in the cloud, where affordability of storage can be utilized (Reis & Housley, 2022).

Wide tables can contain thousands of columns, unlike typical relational databases, which usually have fewer than 100 columns. These wide tables tend to be sparse, with most entries in many fields being null. This sparsity is costly in traditional relational databases because a fixed amount of space is allocated for each field entry, making nulls occupy a large amount of space. In contrast, nulls take up negligible space in columnar databases. Reading from a wide schema in a relational database is slow because each row must allocate space for all specified columns, requiring the database to read the entire row.
Conversely, a columnar database reads only the columns needed for a query, with nulls being virtually free to read. Wide tables often emerge through schema evolution, where engineers add fields over time. This process is slow and resource-intensive in relational databases, whereas in columnar databases, adding a field initially only requires a metadata update. As data gets added to the new field, the column expands accordingly.

Analytics queries on wide tables often run faster than equivalent queries on highly normalized data requiring many joins. Removing joins can have a significant impact on scan performance. The wide table contains all of the data an engineer would have joined in a more rigorous modeling approach. Facts and dimensions are represented in the same table. The lack of data model rigor also means not a lot of thought is involved. Storing this nested data in an analytical storage has many benefits. OBT schema approach can be implemented in a production database when there is a lot of data that needs more flexibility than traditional data modeling rigor provides. While the OBT model can enhance query readability and reduce the need for complex joins, it often introduces significant performance challenges, particularly as the dataset scales.

1.1.3 Indexes

Indexing plays a pivotal role in optimizing query performance within a star schema, significantly enhancing the efficiency of data retrieval operations. Indexing involves creating data structures that improve the speed of data access on tables, particularly fact and dimension tables in a star schema. By facilitating quicker search and join operations, indexes help reduce the time complexity of queries. For example, bitmap indexes are particularly effective in star schemas, as they are well-suited for columns with low cardinality typically found in dimension tables.

Properly designed indexes can drastically reduce the query execution time by allowing the database engine to locate and join relevant rows quickly. (Kimball, 1996) Also, indexes on foreign key columns in fact tables and primary key columns in dimension tables are crucial for optimizing the performance of complex analytical queries. Thus, strategic indexing is a critical optimization technique for ensuring efficient query performance in star schema-based data warehouses. (Adamson, 2017) (O'Neil, 2009)

1.2 OLAP Databases

OLAP system is built to run large analytics queries and is typically inefficient at handling lookups of individual records. For example, modern column databases are optimized to scan large volumes of data, dispensing with indexes to improve scalability and scan performance. Any query typically involves scanning a minimal data block, often 100 MB or more in size. Trying to look up thousands of individual items per second in such a system will bring it to its knees unless combined with a caching layer designed for this use case.

OLAP databases excel in handling numerous read requests, a capability attributed in part to their utilization of columnar serialization. This feature enables databases to selectively scan only the columns relevant to a given query, significantly minimizing the volume of data retrieved from disk (Abadi, Boncz, & Harizopoulos, 2009). Furthermore, the arrangement of data by columns clusters similar values together, leading to good compression ratios with minimal overhead. Consequently, data retrieval from disk and across networks is faster.

While columnar databases demonstrate suboptimal performance in transactional scenarios, such as the asynchronous lookup of numerous individual rows, they are very effective in scenarios requiring extensive data scanning. This includes tasks such as complex data transformations, aggregations, statistical calculations, or evaluating intricate conditions on substantial datasets.

Historically, columnar databases struggled with join operations, prompting data engineers to denormalize data by favoring wide schemas, arrays, and nested data structures wherever feasible. However, recent years have witnessed a significant improvement in join performance for columnar databases. Consequently, while there may still be performance benefits to denormalization, it is no longer deemed a necessity.

1.3 Objectives of the study

This paper aims to systematically compare the query performance and execution plans of these two model designs by running several analytical queries against a normalized star schema and an OBT schema. This study attempts to find potential trade-offs by examining query execution times and resource utilization.

The study aims to find the quantitative benefits of transforming an analytical data warehouse to a denormalized view. The benefits to be measured are:

1. Storage – does the physical storage size on the disk decrease, increase, or stay roughly the same after the transformation?
2. Query speed – how are queries of different sizes, complexity, and nature affected by schema design?
3. Human readability, lines of code – how is the general readability of the data model? Is it readable and understandable for an analyst?

1.4 Scope and limitations

There is a plethora of properties that go into database design. This thesis only considers (but acknowledges other forms where appropriate) only the essential requirements to run a query and the context of an analytical database. This thesis works under the premise of an analytical database. This means that any other forms of databases are, for the most part, not discussed as they require different settings and parameters to operate. Graph, NoSql or cloud technologies/databases are not discussed.

Several critical aspects of database technology are also beyond the scope of this study. For example, database security is not addressed. Also, big data considerations, which involve handling and processing extremely large datasets, have not been explored. Techniques such as partitioning, which divides a database into smaller, more manageable pieces, and compression, which reduces the storage space required for data, are also not discussed.

2 Literature review

This section provides sufficient background for the existing professional benchmarks in the field. Two of the main benchmarks are TPS-DS by the TPC organization and APB-1 created by the OLAP Council. Apart from the two mentioned in this paper, there are more benchmarks that have been developed, but none are as comprehensive and standardized as the TPC-DS benchmark. APB-1 was also well maintained and a considerable benchmark during its time when it was maintained.

2.1 TPS-DS Benchmark

The Transaction Processing Performance Council's Decision Support benchmark (TPC-DS) is a widely recognized and used benchmarking standard designed to evaluate the performance of data warehouses. TPC-DS provides a comprehensive framework for testing and comparing the efficiency of database systems under various complex, business-oriented query workloads. Introduced to address the growing complexity of data warehousing environments, TPC-DS serves as a critical tool for researchers and practitioners aiming to assess the performance capabilities of different database systems in handling analytical workloads. The most recent version was tested by Databricks with a database size of 100,000GB. One of the main metrics in the benchmark is The Performance Metric QphDS®. In general, it tests the time to perform various Tests, such as Power Test and Data Maintenance Test.

Figure 2: The Performance Metric (QphDS@SF)

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

The primary objective of TPC-DS is to simulate a real-world decision support system, including tasks such as data loading, data extraction, complex query processing, and report generation. The benchmark encompasses a broad range of operations, reflecting the diverse

requirements of modern analytical environments. The TPC-DS benchmark suite includes 99 queries, each designed to test different aspects of database performance, such as aggregations, joins, and sub-queries. These queries are derived from real-world business scenarios across different industries, with the organization ensuring that the benchmark remains relevant and applicable.

One of the distinguishing features of TPC-DS is its emphasis on scalability. The benchmark is designed to be executed on different system sizes, from small to large-scale data warehouses, allowing for comparative analysis across various hardware and software configurations. This scalability is achieved through the use of a scale factor, which adjusts the size of the dataset, enabling tests on databases ranging from a few gigabytes to multiple terabytes.

TPC-DS also places a strong emphasis on workload diversity. The benchmark includes a mix of short-running and long-running queries, simple and complex query structures, and varying degrees of data selectivity and cardinality. This diversity ensures that the benchmark thoroughly evaluates the performance characteristics of a database system, including its query optimization capabilities, indexing strategies, and parallel execution efficiency. (TPC, 2021)

2.2 APB-1 OLAP Benchmark

Developed by the OLAP Council in 1998, the goal of the APB-1 is to measure a server's overall OLAP performance rather than the performance of specific tasks. In order to stay relevant and imitate an actual business environment, the operations the benchmark does are chosen so that they reflect common business operations. The operations can be ad-hoc queries, data aggregation, incremental data loading from operational systems, or data loading from external or internal sources. (OLAP Council)

A standard benchmark metric called AQM (Analytical Queries per Minute) was defined to capture the performance of both hardware and software. It represents the number of analytical queries processed per minute, including data loading and computation time. It aims to include data loading performance, calculation performance, and query performance into a single metric.

The OLAP Council recognizes that there is no generally accepted design criteria among data warehouses. Therefore, for the benchmark, there are no structural requirements for the database design. The database that is tested can be either multidimensional or relational, and storing calculated values is neither strictly forbidden nor encouraged.

3 Methodology

The main methodology will be conducted as follows: Dimensional modeling and OBT database schemas will be tested against various analytical queries to see whether the schema design has any effect on the querys' performance.

3.1 Choice of database management system

Acknowledging that the DBMS landscape goes far and wide, certain limitations must be considered. This paper will perform the queries in a sample database that has been built up as an OLAP database. This means that the sample database is designed to handle analytical queries. The chosen approach supports the business in making decisions and providing an overview of its operations.

All of the operations in this paper will be done using free and open-source software (FOSS). The database is running on a MySQL engine and is running on a local machine. The database is accessed through DBeaver, which is a SQL client software application and a database administration tool[2]. Microsoft SQL Server Management Studio is also used as a database client to view detailed query statistics and estimated and actual execution plans.

One feature of the MySql Server 2022 is to apply rowstore or columnstore indexes. Rowstore indexes perform best on queries that seek into the data when searching for a particular value, or for queries on a small range of values. With the given settings, SSMS applies rowstore indexing natively and by default on the tables. Columnstore indexes have to be specified in order to make use of them.

Columnstore indexes give high performance gains for analytic queries that scan large amounts of data, especially on large tables. Columnstore indexes are used on data warehousing and analytics workloads, especially on fact tables, because they tend to require full table scans rather than table seeks.[3]

[2] https://www.dbeaver.io/

[3] https://learn.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview?view=sql-server-ver16#benefits

Some relational DBMS provide columnar storage natively, some support it through indexing. For example, Vertica, a SQL dialect, natively stores data in columns rather than in rows, reducing disk I/O and query processing time. MySQL Server 2022 supports the benefits of columnar storage through specific columnar store indexing, which is invoked with functions that can be applied to tables. By default, MySQL Server 2022 uses B-trees as its primary storage mechanism (Fritchey, 2018).

3.2 Performance metrics

To comprehensively evaluate the query performance of different schemas within the AdventureWorks2022DW database, a series of analytical queries will be executed against the star schema and a denormalized "One Big Table" schema. These queries will cover a range of typical data warehouse operations, including aggregations, joins, and filtering.

Query execution time will be recorded to measure the speed of data retrieval and processing under each schema. In addition to execution time, other performance metrics will be assessed to provide a holistic view of database efficiency. These metrics include disk I/O operations, which will indicate the amount of data read from and written to disk; CPU utilization, which will reflect the computational resources consumed during query execution; memory usage, which will show the amount of RAM required; and query plan complexity, which will provide insights into the optimization strategies employed by the database engine. By analyzing these metrics, we aim to identify the strengths and weaknesses of each schema in handling analytical workloads.

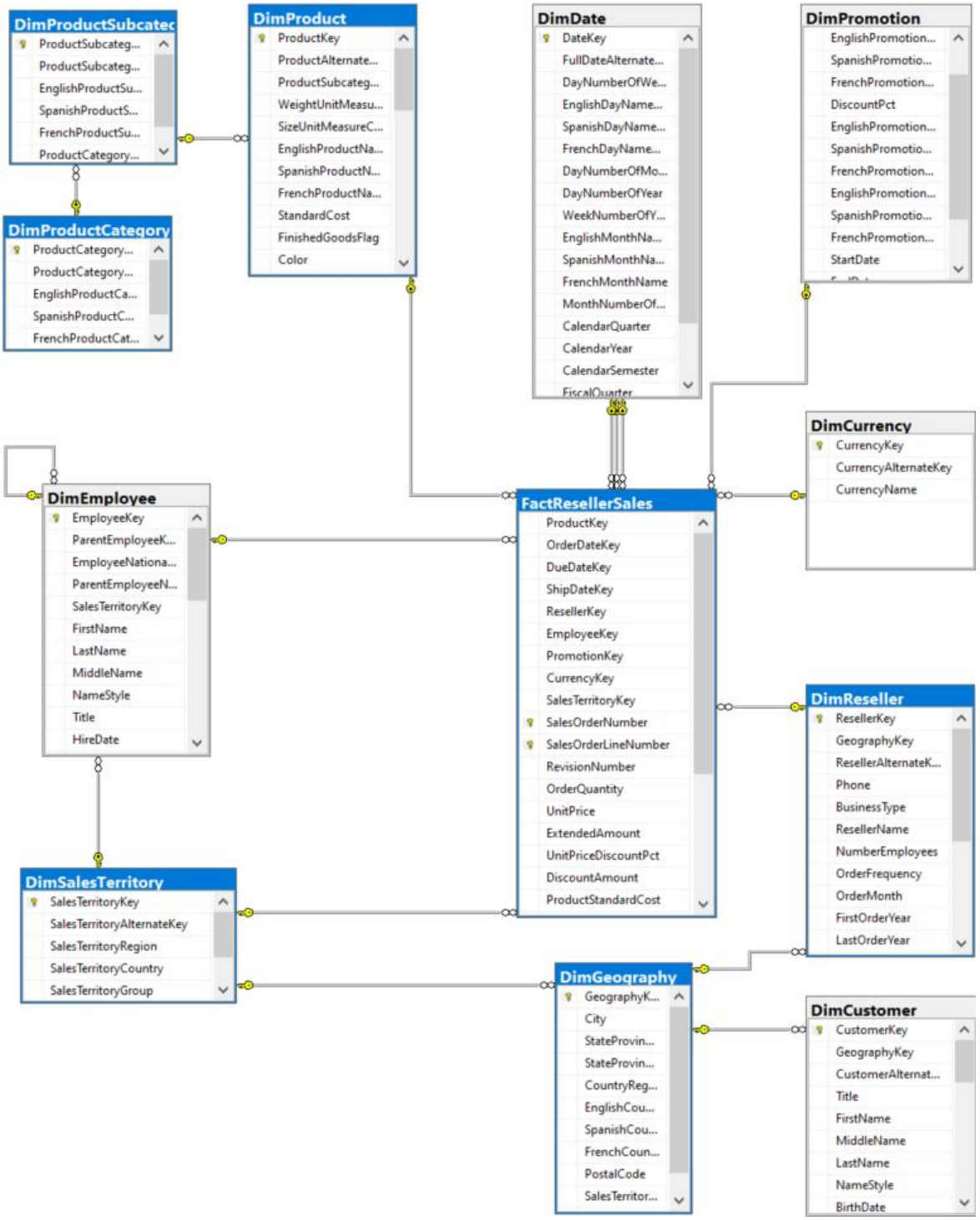
3.3 AdventureWorks sample Database

The AdventureWorks databases are sample databases that were originally published by Microsoft to show how to design a SQL Server database using SQL Server 2008 and demonstrate its features. The database has been constantly updated every couple of years or so to demonstrate the newly added features of SQL Server. AdventureWorks is the OLTP database, and AdventureWorksDW (onwards referenced as the database) is the data warehouse version of it. The latest version was released in 2022, hence the name AdventureWorksDW2022. The database originally has 31 tables, the backup file to restore the database from is 97 MB and when restored, inflates to around 7 GB. The whole physical diagram of the database can be seen in Appendix I.

This paper will deal with the fact table FactResellerSales and its relevant dimensional tables. The star schema of the database can be seen in Table 2. This schema covers and, therefore, provides an overview of this data mart, in this case, a reseller's sales. In the middle is the fact table, holding all of the factual data about an event or a sale. All relevant information is distributed to the dimensional tables, such as promotion application, product(s) being sold, reseller information, reseller's geographical information, and so on. The tables are connected with primary and foreign keys, which help with data integrity and readability.

The following table displays the star schema of the databases' data mart. The highlighted tables are the ones that have been included in the denormalization process of the fact table. Queries related to customer data still have to be joined because there can not be a direct relationship between the OBT and DimCustomer because DimCustomer and the fact table FactResellerSales would be joined through the primary key of DimGeography, as can be seen from the diagram. This would cause an explosion of rows because DimGeography and DimCustomer have a many-to-one relationship, meaning many customers exist in one geographic location. For example, querying for product info while joining customer data would cause the rows in the returned resultset to cross-join product and customer information, effectively multiplying the table counts and, therefore, halting the query performance.

Figure 3. Star Schema of FactResellerSales data mart. Highlighted tables used in OBT table.



In the table, the logic of proper dimensional modeling can be seen. The fact table 'FactResellerSales,' which holds all of the quantitative data for the sales, is in the middle, with complimentary dimensional tables, helping to describe the factual events (for example, reseller sales). Visually, it can be seen that it represents a star schema. Connections between tables are defined with keys, where the table touching a key is being used as a foreign key in another table.

4 Results

4.1 Translation of standard queries to OBT queries

A straightforward method to translate the initial queries to OBT queries would be to remove joins and calculate and name the columns in the final select statement. This is not possible, however, due to the nature of the SELECT statement. When a SELECT statement is executed, the first operation is to look for the tables in the FROM clause. Since this clause is after the SELECT clause, the query processor does not know how to calculate the columns in the SELECT statement (Weinberg, 1999). The formal rules and order of execution for the SELECT statement are detailed in the ISO/IEC 9075 SQL standard, but this document is not freely available. MySQL follows the SQL Standard.

A solution to this problem would be either to use Common Table Expressions (CTEs) or sub-queries. For the sake of readability and assuming a data analyst's requirement for a quick data analysis, CTEs will be used.

It should also be noted that Standard SQL is not meant to perform highly complex analytical queries. Some functions are only for simpler exploratory data analysis queries, such as basic mathematical operations and date/time conversions.[4]

4.2 Analytical queries

In the following section, analytical queries are performed on both schemas. Execution plans of the queries are viewed and compared to each other. First, a query against the original "standard" (referred to as Standard query onwards) data schema is run, then against the OBT schema. A discussion of the execution plan comparison will follow afterward. Visual graphs of the execution plans can be seen in the appendix.

The queries used in this paper are heavily inspired by the queries run by the TPC-DS benchmark. The business questions and the queries' parameters can be viewed from the official documentation in Appendix B. [5] The queries have been fully uploaded to GitHub as well for viewing. [6]

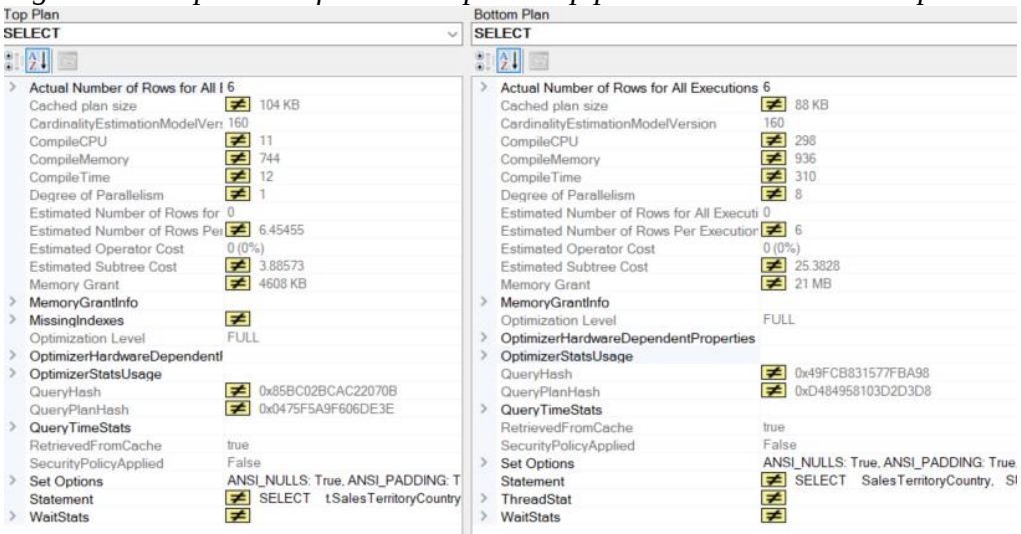
[4] https://dev.mysql.com/doc/refman/8.0/en/functions.html
[5] https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-DS_v3.2.0.pdf
[6] https://github.com/Agirish/tpcds

To capture different nature of queries, three computationally different queries are performed. First, a comprehensive sql select with some filtering that involves as many tables as possible is done. Second, a complex join operation is performed, and thirdly, a query that uses a number of aggregate functions will be used.

4.3 Microsoft SQL Server Management Studio

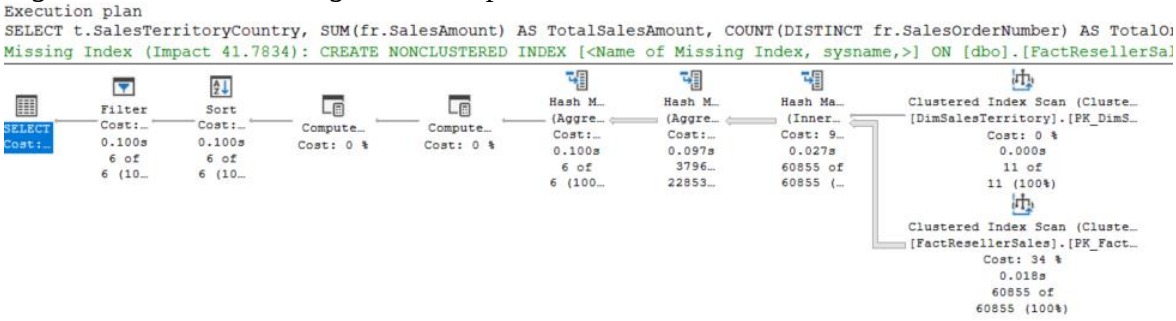
In terms of analyzing and optimizing queries and database properties, SSMS provides excellent depth natively. There are many variables that do not fit into the scope of this paper that could affect and explore query performance in greater detail, such as query hints. Hints give the database administrator the ability to store query text, query plans, and query performance history at the database scope for troubleshooting and performance analysis. Below is a detailed view of the execution plan comparison for queries. Standard and OBT queries could be compared conveniently.

Figure 4: Comparison of execution plan. Top plan – Standard. Bottom plan – OBT.



SSMS also actively evaluates the queries through the query optimizer and offers suggestions on constructing non-clustered indexes for faster data retrieval should the same query run on the database again.

Figure 5: SSMS indicating an index optimization



4.3.1 Query optimizer

SQL Server Management Studio includes functionality that implements some degree of automation in the task of actual execution plan analysis, especially for large and complex plans. The goal is to make it easier to find scenarios of inaccurate Cardinality Estimation and get recommendations on which possible mitigations may be available.

The SQL Server Query Optimizer is a cost-based Query Optimizer. This means it selects query plans with the lowest estimated processing cost to execute. The Query Optimizer determines the cost of executing a query plan based on two main factors:
The total number of rows processed at each level of a query plan is referred to as the cardinality of the plan. The cost model of the algorithm is dictated by the operators used in the query.

The first factor, cardinality, is used as an input parameter of the second factor, the cost model. Therefore, improved cardinality leads to better estimated costs and, in turn, faster execution plans. Cardinality estimation (CE) in SQL Server is derived primarily from histograms created when indexes or statistics are created manually or automatically. Sometimes, SQL Server also uses constraint information and logical rewrites of queries to determine cardinality.

4.4 Construction and comparison of queries

The following queries are built so that their nature reflects something that a data analyst

might query in the database. For the purpose of this paper, it is not important to answer any business question, meaning the result that the query provides might not provide any analytical value. It is important to look at the query's performance instead. SQL code for the queries can be found in Appendix II.

The performance metrics of 3 queries are presented below. Query 1 is listed three times to show the effects of an applied non-clustered index on the OBT dataset. Non-clustered indexes were applied where they were suggested and marked with the _cis suffix.

Description of the metrics used for measuring query performance:

1. Runtime – time, in ms, it took for the query to finish. This includes all of the time involved in executing the query, including coming up with an execution plan from the query engine, latency, and a few others.
2. LoC – lines of code, rough metric for readability and ease of use. All queries are compiled in a similar, standard format.
3. CompileCPU – Amount of CPU resources needed to process the query.
4. CompileMemory – Amount of memory (in Kb) to process the query.
5. RwAff – Rows affected. This number indicates how many rows of data the query processed. Does not account for I/O operations.

	Query 1			Query 2		Query 3	
	q1std	q1obt	q1_obt_cis	q2std	q2obt	q3std	q3obt
Runtime	5.353ms	5.022ms	4.574ms	1.405ms	1.608ms	0.233ms	0.189ms
LoC	56	45	45*	33	27	48	40
Compile							
CPU	316	191	117	9	30	19	16
Memory	6888	2640	2424	1008	504	1464	952
RwAff.	175995	175959	175959	28540	30748	65	28

Table 2. Query performance results on selected metrics.

For Query 1, moving from the standard schema (q1std) to the OBT schema (q1obt) results in a reduced runtime from 5.353ms to 5.022ms, indicating a modest performance improvement. Further optimization using a non-clustered index (q1_obt_cis) enhances performance significantly, reducing the runtime to 4.574ms. This trend is mirrored in the compile-time resource consumption, where CPU usage drops from 316 to 191 and further to 117. Memory usage also decreases substantially, from 6888 to 2640 and then to 2424, reflecting the efficiency of the indexed OBT approach. The constant Read/Write Affinity (175995) suggests that the I/O operations remain consistent, implying that the performance gains are mainly due to better query processing and indexing. High numbers for the standard query in resource consumption come from join operations. The denormalized schema, particularly when combined with indexing (q1_obt_cis), offers clear advantages in runtime and resource usage. The benefits stem from reduced join complexity and efficient index utilization.

For Query 2, the standard schema (q2std) outperforms the OBT schema (q2obt) in terms of runtime (1.405ms vs. 1.608ms). This indicates that for this specific query, the overhead of

managing a larger denormalized table in OBT outweighs the benefits of simplified joins. The standard schema also shows lower CPU usage (9 vs. 30) but higher memory usage (1008 vs. 504), suggesting that the query benefits from the normalized structure in terms of CPU efficiency but requires more memory to handle the joins. The Read/Write Affinity is slightly higher for the OBT version (30748 vs. 28540), indicating more I/O operations, likely due to the larger size of the denormalized table. The standard normalized schema performs better, suggesting that the normalized approach can be more efficient for complex queries involving multiple joins. The increased memory requirement in the standard version indicates that joins are memory-intensive but still more efficient CPU usage than handling a large denormalized table.

Query 3 shows the most significant performance improvement when using the OBT schema. The runtime drops from 0.233ms (q3std) to 0.189ms (q3obt), and both CPU (19 vs. 16) and memory (1464 vs. 952) usage decrease, reflecting the efficiency of the denormalized schema for this query. The lower Read/Write Affinity in the OBT version (28 vs. 65) suggests fewer I/O operations, which can be attributed to the reduced need for join operations in the OBT schema. This highlights that the denormalized schema can provide substantial performance gains for simpler queries. The denormalized schema significantly outperforms the normalized schema, demonstrating its effectiveness for simpler queries that benefit from the absence of join operations.

5 Discussion

5.1 One Big Table rationale

In terms of security, it is possible but difficult to separate different business users from various areas of the table. For example, suppose an employee table with sensitive personal information is leaked into the OBT to simplify queries for the annual results of sales representatives. In that case, there might be people from different areas of business that might have access to this source information, which they should not have.

The paradigm of data modeling through OBT logic might have been reasonable and not too costly in the past. However, with the requirements and the 4 V's (Velocity, Volume, Variety, and Veracity) of present days' data, this method of modeling will no longer do. When the characteristics of data is discussed, Big Data technologies will soon be mentioned, for example Apache Spark or Hadoop.

If the database should receive updates, then data redundancy problems will arise. It is difficult to maintain a denormalized database due to the code complexity because changes need to be propagated to multiple places where redundant data is stored. Over time, this will significantly increase storage requirements and data consistency issues.

Additionally, as could be seen from OBT query execution plans, every time the OBT query executes, it has to do a full scan of the table.

Arguments for using the OBT data model:

1. By avoiding joins, queries can be executed more quickly and with less computational overhead. This is particularly beneficial in read-heavy analytical workloads where speed is crucial. Using denormalized tables (like One Big Table) simplifies query execution since all necessary data is in a single table, reducing the need for complex join operations.
2. Predictability in performance. With constant data loadings, resources will be utilized at a more constant rate in a denormalized table where joins are reduced. This happens because the query execution plan is less complex. When resources are tight, this might be an argument to consider.
3. Database storage has become remarkably affordable in recent years, driven by technological advances and the widespread adoption of cloud computing. Companies no longer need to invest in extensive on-premises infrastructure; instead, they can leverage cloud services to access scalable storage solutions. These services offer flexible pricing models, allowing businesses to pay on-demand or precisely for the amount of storage they utilize. This pay-as-you-go approach reduces upfront

costs and enables companies to adjust their storage needs dynamically, ensuring cost efficiency and resource optimization. As a result, organizations of all sizes can afford robust data storage solutions, fostering greater innovation and agility in managing their data assets.

6 Conclusion

In conclusion, the decision to use a normalized versus a denormalized schema should be guided by the nature of the queries being run. For queries that benefit from reduced join complexity and can leverage indexing effectively, the denormalized OBT schema provides substantial performance improvements. However, the normalized schema may still offer better performance for more complex queries where join operations are central. Consequently, a hybrid approach may yield the best overall performance, where the schema design considers the specific workload and query patterns.

6.1 Suggestions for future research

For future research, exploring the potential optimizations or adaptations that could be made to the less standardized data modeling approach to improve query performance without sacrificing readability would be valuable. SSMS offers those solutions, but in the context of this paper, they could not be implemented. This could involve investigating different indexing strategies, partitioning techniques, or data organization methods made for such schemas. Additionally, further comparative studies could be conducted using larger and more diverse datasets to confirm the findings of this paper. Also, exploring the impact of different database management systems and query optimization techniques on the performance of these alternative data models could provide valuable insights for practitioners seeking to optimize their data warehousing solutions. The development of more efficient and adaptable architectural practices.

Also, the additional dimension of adding temporal data and analyzing that is an endeavor in itself. This adds additional complexity and possibilities to data warehousing schema, which often, if not always, exist in databases.

References

Abadi, D. J., Boncz, P. A., & Harizopoulos, S. (2009). Column-oriented database systems. *Proceedings of the VLDB Endowment*, p. 1.

Adamson, C. &. (2017). Star Schema: The Complete Reference.

Brackenbury, W., Liu, R., Mondal, M., & Elmore, A. J. (2018). Draining the Data Swamp: A Similarity-based Approach. *HILDA '18: Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, 1.

Breslin, M. (2004). Data Warehousing Battle of the Giants: Comparing the Basics of the model Kimball and Inmon Models. *Business Intelligence Journal*, 12-19.

Council, O. (n.d.). *Olap Council.* Retrieved from http://www.olapcouncil.org/research/resrchly.htm

Fritchey, G. (2018). *SQL Server Execution Plans.* Redgate Publishing.

Hariharaputran, S. (2010). BioSilicoSystems - A Multipronged Approach Towards Analysis and Representation of Biological Data. *Nature Precedings*, 33.

Kimball, R. (1996). *The Data Warehouse Toolkit.*

Kotiranta P, J. M. (2022). Performance of Graph and Relational Databases in Complex Queries. *Applied Sciences*.

O'Neil, P. O. (2009). The Star Schema Benchmark and Augmented Fact Table Indexing. *Technology Conference on Performance Evaluation and Benchmarking*, (pp. 237-252).

OLAP Council. (n.d.). Retrieved 05 03, 2024, from http://www.olapcouncil.org/research/bmarkly.htm

Reddy, G. S., Srinivasu, R., Rao, M. P., & Rikkula, S. R. (2010). Data warehousing, Data mining, OLAP and OLTP technologies are essential elements to support decision-making process in industries. *International Journal on Computer Science and Engineering*, 2.

Reis, J., & Housley, M. (2022). *Fundamentals of Data Engineering.*

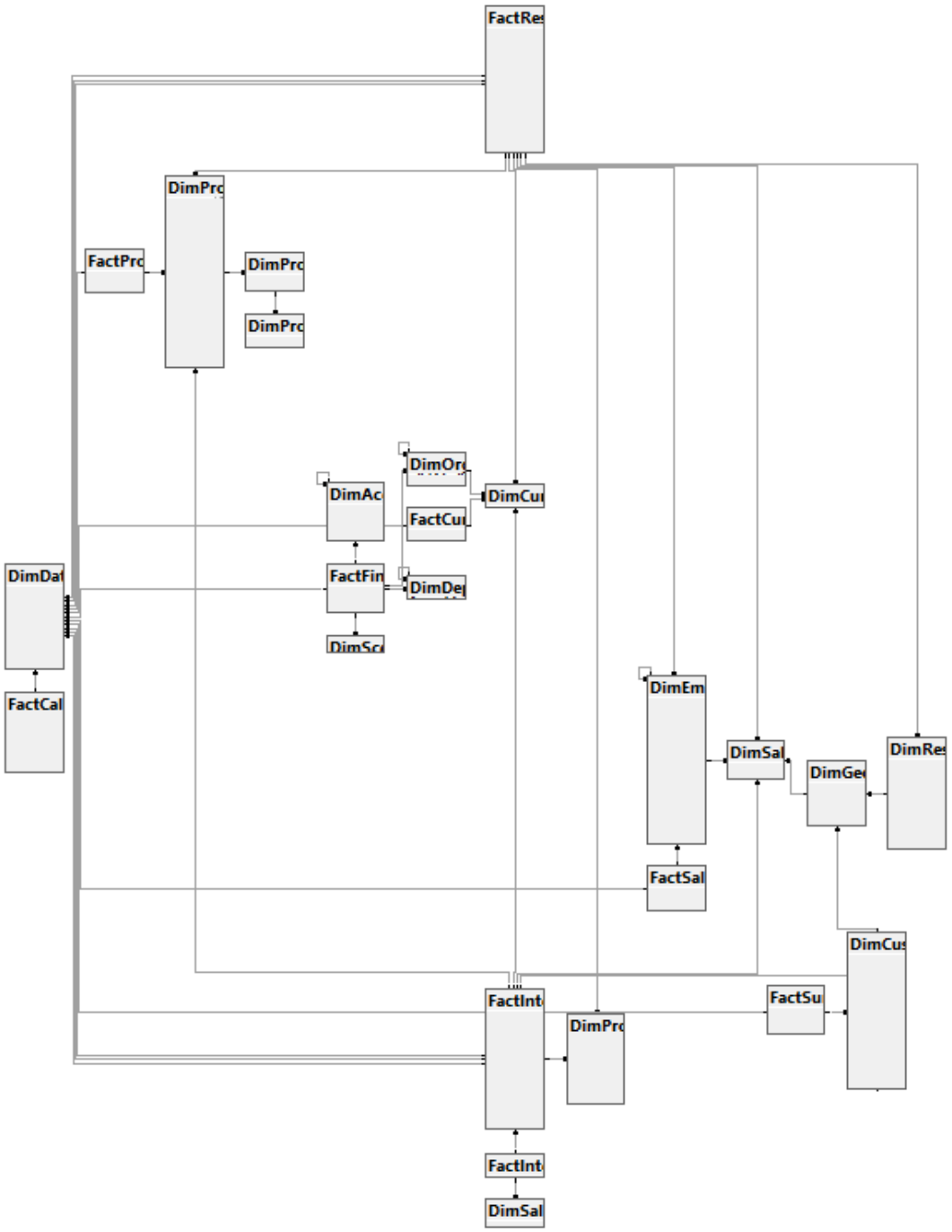
TPC, T. P. (2021, June). *tpc.org.* Retrieved from https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-DS_v3.2.0.pdf

Weinberg, J. R. (1999). *SQL - The Complete Reference.* Retrieved from

http://160592857366.free.fr/joe/ebooks/ShareData/SQL%20-%20The%20Complete%20Reference.pdf

Appendix

I. Physical design of the database



II. SQL Queries

Query 1 Standard:

```sql
SELECT
       fs.SalesOrderNumber,
       fs.OrderDateKey,
       fs.DueDateKey,
       fs.ShipDateKey,
       fs.SalesAmount,
       fs.TaxAmt,
       fs.Freight,
       fs.TotalProductCost,
       fs.UnitPriceDiscountPct,
       c.FirstName AS CustomerFirstName,
       c.LastName AS CustomerLastName,
       c.EmailAddress AS CustomerEmail,
       c.Phone AS CustomerPhone,
       e.FirstName AS SalesPersonFirstName,
       e.LastName AS SalesPersonLastName,
       e.Title AS SalesPersonTitle,
       p.ProductKey,
       p.EnglishProductName,
       ps.EnglishProductSubcategoryName AS ProductSubcategoryName,
       pc.EnglishProductCategoryName,
       g.City AS ShipToCity,
       g.StateProvinceName AS ShipToState,
       d.FullDateAlternateKey AS OrderDate,
       d.CalendarYear AS OrderYear
FROM
       FactResellerSales fs
JOIN
    DimEmployee e ON
       fs.EmployeeKey = e.EmployeeKey
JOIN
    DimProduct p ON
       fs.ProductKey = p.ProductKey
JOIN
    DimProductSubcategory ps ON
       p.ProductSubcategoryKey = ps.ProductSubcategoryKey
JOIN
    DimProductCategory pc ON
       ps.ProductCategoryKey = pc.ProductCategoryKey
JOIN
    DimReseller dr ON
       fs.ResellerKey = dr.ResellerKey
JOIN
    DimGeography g ON
       dr.GeographyKey = g.GeographyKey
JOIN
    DimDate d ON
       fs.OrderDateKey = d.DateKey
JOIN
    DimCustomer c ON
       g.GeographyKey = c.GeographyKey
WHERE
       fs.OrderDate >= '2012-12-29 00:00:00.000'
       AND fs.SalesAmount > 1000
```

```
ORDER BY
	fs.SalesOrderNumber;
```

Query 1 OBT:

```
SELECT
	SalesOrderNumber,
	ShipDateKey,
	SalesAmount,
	TaxAmt,
	Freight,
	TotalProductCost,
	UnitPriceDiscountPct,
	c.FirstName AS CustomerFirstName,
	c.LastName AS CustomerLastName,
	c.EmailAddress AS CustomerEmail,
	c.Phone AS CustomerPhone,
	e.FirstName AS SalesPersonFirstName,
	e.LastName AS SalesPersonLastName,
	e.Title AS SalesPersonTitle,
	ProductKey,
	EnglishProductName,
	EnglishProductSubcategoryName AS ProductSubcategoryName,
	EnglishProductCategoryName,
	City AS ShipToCity,
	StateProvinceName AS ShipToState,
	FullDateAlternateKey AS OrderDate,
	CalendarYear AS OrderYear
FROM
	FactResellerSales_OBT fs
JOIN
    DimEmployee e ON
	fs.EmployeeKey = e.EmployeeKey
JOIN
    DimReseller dr ON
	fs.ResellerKey = dr.ResellerKey
JOIN
    DimGeography g ON
	dr.GeographyKey = g.GeographyKey
JOIN
    DimDate d ON
	fs.ShipDateKey = d.DateKey
JOIN
    DimCustomer c ON
	g.GeographyKey = c.GeographyKey
WHERE
	fs.OrderDate >= '2012-12-29 00:00:00.000'
	AND fs.SalesAmount > 1000
ORDER BY
	fs.SalesOrderNumber;
```

Query 2 Standard:

```
WITH SalesSummary AS (
    SELECT
        fs.OrderDateKey,
        fs.SalesAmount,
        pc.EnglishProductCategoryName,
        SUM(fs.SalesAmount) OVER (PARTITION BY pc.EnglishProductCategoryName ORDER
BY fs.OrderDateKey) AS RunningTotalSales,
        AVG(fs.SalesAmount) OVER (PARTITION BY pc.EnglishProductCategoryName ORDER
BY fs.OrderDateKey) AS RunningAverageSales,
        ROW_NUMBER() OVER (PARTITION BY pc.EnglishProductCategoryName ORDER BY
fs.OrderDateKey) AS RowNum
    FROM
        FactResellerSales fs
    JOIN
        DimProduct p ON fs.ProductKey = p.ProductKey
    JOIN
        DimProductSubcategory ps ON p.ProductSubcategoryKey =
ps.ProductSubcategoryKey
    JOIN
        DimProductCategory pc ON ps.ProductCategoryKey = pc.ProductCategoryKey
)
SELECT
    OrderDateKey,
    EnglishProductCategoryName,
    SalesAmount,
    RunningTotalSales,
    RunningAverageSales,
    RowNum,
    MAX(SalesAmount) OVER (PARTITION BY EnglishProductCategoryName) AS
MaxSalesAmount,
    MIN(SalesAmount) OVER (PARTITION BY EnglishProductCategoryName) AS
MinSalesAmount
FROM
    SalesSummary
WHERE
    OrderDateKey > '20121229'
ORDER BY
    EnglishProductCategoryName, OrderDateKey;
```

Query 2 OBT:

```
WITH SalesSummary AS (
    SELECT
            ShipDateKey,
        SalesAmount,
        EnglishProductCategoryName,
        SUM(SalesAmount) OVER (PARTITION BY EnglishProductCategoryName ORDER BY
ShipDateKey) AS RunningTotalSales,
```

```
        AVG(SalesAmount) OVER (PARTITION BY EnglishProductCategoryName ORDER BY
ShipDateKey) AS RunningAverageSales,
        ROW_NUMBER() OVER (PARTITION BY EnglishProductCategoryName ORDER BY
ShipDateKey) AS RowNum
    FROM
        FactResellerSales_OBT fs
)
SELECT
    ShipDateKey,
    EnglishProductCategoryName,
    SalesAmount,
    RunningTotalSales,
    RunningAverageSales,
    RowNum,
    MAX(SalesAmount) OVER (PARTITION BY EnglishProductCategoryName) AS
MaxSalesAmount,
    MIN(SalesAmount) OVER (PARTITION BY EnglishProductCategoryName) AS
MinSalesAmount
FROM
    SalesSummary
WHERE
    ShipDateKey > '20121229'
ORDER BY
    EnglishProductCategoryName, ShipDateKey;
```

Query 3 Standard:

```
WITH MonthlySales AS (
    SELECT
        pc.EnglishProductCategoryName,
        YEAR(d.FullDateAlternateKey) AS SalesYear,
        MONTH(d.FullDateAlternateKey) AS SalesMonth,
        SUM(fs.SalesAmount) AS MonthlySales,
        AVG(SUM(fs.SalesAmount)) OVER (PARTITION BY pc.EnglishProductCategoryName,
YEAR(d.FullDateAlternateKey)) AS AvgMonthlySales
    FROM
        FactResellerSales fs
    JOIN
        DimProduct p ON fs.ProductKey = p.ProductKey
    JOIN
        DimProductSubcategory ps ON p.ProductSubcategoryKey =
ps.ProductSubcategoryKey
    JOIN
        DimProductCategory pc ON ps.ProductCategoryKey = pc.ProductCategoryKey
    JOIN
        DimDate d ON fs.OrderDateKey = d.DateKey
    WHERE
        YEAR(d.FullDateAlternateKey) = 2012
    GROUP BY
        pc.EnglishProductCategoryName, YEAR(d.FullDateAlternateKey),
MONTH(d.FullDateAlternateKey)
),
```

```
Deviations AS (
    SELECT
        EnglishProductCategoryName,
        SalesYear,
        SalesMonth,
        MonthlySales,
        AvgMonthlySales,
        ABS(MonthlySales - AvgMonthlySales) / AvgMonthlySales AS SalesDeviation
    FROM
        MonthlySales
    WHERE
        ABS(MonthlySales - AvgMonthlySales) / AvgMonthlySales > 0.1
)
SELECT
    EnglishProductCategoryName,
    SalesYear,
    SalesMonth,
    MonthlySales,
    AvgMonthlySales,
    SalesDeviation
FROM
    Deviations
ORDER BY
    SalesDeviation DESC;
```

Query 3 OBT:

```
WITH MonthlySales AS (
    SELECT
        EnglishProductCategoryName,
        YEAR(d.FullDateAlternateKey) AS SalesYear,
        MONTH(d.FullDateAlternateKey) AS SalesMonth,
        SUM(SalesAmount) AS MonthlySales,
        AVG(SUM(SalesAmount)) OVER (PARTITION BY EnglishProductCategoryName,
YEAR(FullDateAlternateKey)) AS AvgMonthlySales
    FROM
        FactResellerSales_OBT fs
    JOIN
        DimDate d ON fs.ShipDateKey = d.DateKey
    WHERE
        YEAR(d.FullDateAlternateKey) = 2012
    GROUP BY
        EnglishProductCategoryName, YEAR(d.FullDateAlternateKey),
MONTH(d.FullDateAlternateKey)
),
Deviations AS (
    SELECT
        EnglishProductCategoryName,
        SalesYear,
        SalesMonth,
        MonthlySales,
        AvgMonthlySales,
```

```
        ABS(MonthlySales - AvgMonthlySales) / AvgMonthlySales AS SalesDeviation
    FROM
        MonthlySales
    WHERE
        ABS(MonthlySales - AvgMonthlySales) / AvgMonthlySales > 0.1
)
SELECT
    EnglishProductCategoryName,
    SalesYear,
    SalesMonth,
    MonthlySales,
    AvgMonthlySales,
    SalesDeviation
FROM
    Deviations
ORDER BY
    SalesDeviation DESC;
```

III. Github repository

The link to the Github code repository is: https://github.com/RolandP97/masterthesis. The instructions on how to reproduce the results is provided in the "README.md" of the main branch.

IV. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Roland Pajuleht**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

   "A comparative study of database schema on query performance in data warehousing"

   supervised by Eduard Ševtšenko.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons license CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Roland Pajuleht
15/05/2024