Pedro Henrique Marangoni

Comparing Performance of Data Warehouse Designs: Implementing Generic Data Warehouse, Star, and Snowflake Schemas using TPC-H as Data Source

Master Thesis

at the Chair for Information Systems and Information Management
(Westfälische Wilhelms-Universität, Münster)

Supervisor: Dr. Friedrich Chasin

Presented by: Pedro Henrique Marangoni
Friedhofsweg 6a
48324 Sendenhorst
+49 176 69792014
pedro.marangoni@uni-muenster.de

Date of Submission: 2020-04-20

Content

Figures

Tables

Abbreviations

3NF	Third Normal Form
ADAPT	Application Design for Analytical Processing Technologies
BI	Business Intelligence
BPMN	Business Process Model and Notation
DB	Database
DBMS	Database Management System
DDL	Data Definition Language
DFM	Dimension Fact Model
DML	Data Manipulation Language
DQL	Data Query Language
DSA	Data Staging Area
DW	Data Warehouse
DWD-Q	Data Warehouse Designs Query
ER	Entity-Relation
ETL	Extract Translate Load
FK	Foreign Key
GB	Gigabyte
GDWH	Generic Data Warehouse
HDD	Hard Disk Drive
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoC	Inversion of Control
IS	Information Systems
JPQL	Java Persistence Query Language
JSP	JavaServer Pages
KPI	Key Performance Indicator
ME/RM	Multidimensional Entity Relationship Model
MVC	Model-View-Controller
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
OOP	Object-Oriented Paradigm
ORM	Object-Relation Mapping
OS	Operating System
PK	Primary Key
POM	Project Object Model
RAM	Random Access Memory
RDMBS	Relational Database Management System
SCSI	Small Computer System Interface
SF	Scale Factor
SQL	Standard Query Language
SSD	Solid-State Drive
TPC	Transaction Processing Performance Council
UML	Unified Modeling Language
WWU	Westfälische Wilhelms-Universität Münster

1 Introduction

Data analysis support organizations to understand their weakness, strengths, and opportunities. A Business Intelligence (BI) system is a set of tools designed to discover knowledge through past data to enhance the assertiveness of strategic decisions. A BI environment consists of a Data Warehouse (DW) architecture, an Online Analytical Processing (OLAP) tool, and dashboards. As a core element of BI systems, a DW represents an integrated repository containing structured and cleaned data extracted from diverse sources. OLAP operations are used to change the level of data aggregation in real-time querying. BI dashboards offer to DW/BI users tools to generate reports and analyze data (Gupta and Singh 2014; Harrison et al. 2015).

The most know DW architectures are the star, snowflake, and constellation (Inmon 2005; Kimball and Ross 2002). Becker and Winkelmann (2019) posit that star and snowflake schemas are complex structures to maintain, and the stored data are inconsistent and redundant. To overcome these issues, they proposed another DW architecture called Generic Data Warehouse (GDWH).

The problem arises when a huge volume of data (billions of records, or petabytes) must be read and loaded into BI environments. A process called Extracting-Transform-Loading (ETL) has to be executed to extract, clean, and standardize data from different sources, and load into a DW architecture. Each DW architecture has its advantages and drawbacks on ETL and query performance (Darmont et al. 2007). Efforts on DW performance comparison have been made with star and snowflake schemas (Darmont 2009; Dehdouh et al. 2014; O'Neil et al. 2006). According to Campbell et al. (2017, p. 2), comparing performance in DW "remains largely unexplored from a technical perspective in data management and analytics."

Measuring performance is important to support system engineers in choosing the most suitable DW architecture, and to evaluate the results of using optimization techniques (Darmont et al. 2007). With this target, the present work contributes to the DW performance comparison debate in novel and meaningful ways by evaluating the performance of three DW architectures: GDWH, star, and snowflake. The ETL processes specification and performance measurement for each architecture is also a contribution from this study.

The essential criterion of the DW performance evaluation is the query running time. However, other criteria will be used to evaluate each DW architecture: data redundancy and consistency, ETL performance, and metadata management. For the GDWH architecture, three ETL variations will be designed as a trial to improve query

performance. The results of this study can be used to support IT decision-makers and DW/BI software architects in choosing the most suitable DW architecture for their needs.

The Data Warehouse design concepts and applications will be utilized to transform and analyze synthetic data generated with the TPC-H Benchmark tool. The TPC-H data model represents a typical retail company that buys and sells products over the world. The retail industry is a prominent area where BI systems are often used. Sales performance indicators, customer behavior, and logistic analysis are typical examples of BI analysis on retail companies (Gang et al. 2008).

Business questions like "what is the total sales turnover in Germany in 1998?" will be answered using the three DW architectural choices. As no particular application was found to query the selected DW architectures, a Java application called Data Warehouse Designs-Query[1] (DWD-Q) was developed. With DWD-Q, users can select a scale factor and a DW architecture to be queried.

The next sections of this study are organized as follows: Chapter 2 presents the concepts used in this study and the related research found in the literature review. Chapter 3 describes the research approach applied and the process of data source selection. Chapter 4 contains the steps used to design the DW architecture using TPC-H as the data source. Chapter 5 shows the IT infrastructure and the software utilized to support development and data management. The ETLs designed for GDWH, star, and snowflake will be shown in chapter 6. The process of business question definition will be shown in chapter 7. The corresponding SQL queries for each business question and DW architecture and query running times will be presented in chapter 8. Chapter 9 covers the discussion about performance aspects related to each DW architecture analyzed. Chapter 10 concludes this study.

[1] Available on http://dwh-architectures.uni-muenster.de

2 Research Background

2.1 Data Warehouse Fundamentals

A typical retail company has two different architectures of information systems: Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP). OLTP are systems responsible for supporting day-to-day operations and transactions. The main objective is to create and update information efficiently, keeping consistency and availability (Garani and Helmer 2012).

Reports are generated for managers who wish to analyze weekly or monthly sales performance, control product stock, or analyze supplier data. With the constant use of web applications in the retail industry, data generated from OLTP systems can achieve massive levels, and even regular reports can take hours to be generated. Analyzing data in such architecture has performance drawbacks. That is why a second system architecture named OLAP has been developed to answer analytical questions. OLAP is a cube representation of data that has better query performance compared to traditional OLTP systems (Gupta and Singh 2014).

An OLAP data cube has dimensions and facts. Dimensions "contain the textual context associated with a business process measurement event. They describe the 'who, what, where, when, how and why' associated with the event" (Kimball et al. 2013, p. 13). Dimensions contain attributes that constrain a query by grouping data, e.g., sales volume grouped by market segment. The market segment is an attribute of the customer dimension. Sales volume represents a fact. A fact is "the performance measurement resulting from an organization's business process events" (Kimball et al. 2013, p. 10). A business measure usually is numeric and additive, e.g., revenue, cost, profit, sales volume. The word "ratio" was used in this study with the same meaning as a business measure.

There are four basic types of OLAP operations (J. Gray et al. 1996):

- Roll-up: it consists of aggregation or consolidation of a query, e.g., if a query returns the sales volume grouped by state, a roll-up operation aggregates the sales volumes by country or continent.

- Drill-down: the opposite of roll-up, e.g., if a query returns sales volume by year, a drill-down operation goes down to month or week level.

- Slice and dice: on slice operation, only one dimension is selected to create a new cube, e.g., the first query represents sales volume by year; a new cube would slice

the result to the year 1999. On dice operations, more dimensions and filters can be added, e.g., sales volume for products A and B, and the years 1998 and 1999.

- Pivot: cube rotation (changing axes position) to see data from another perspective.

The use of those OLAP operations has improved query performance on transactional systems. But data on those systems usually are inconsistent, fragmented, and not standardized. Data must be first transformed and stored efficiently to generate reports with reliable information and with fast query response time to decision-making users. Therefore, a Data Warehouse is the most suitable architecture which supports the process of collecting and storing data to be efficiently queried by decision-making users (Chaudhuri and Dayal 1997).

Inmon (2005, p. 29) defines Data Warehouse as a "subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions." Subject-oriented means that a DW represents data related to the business entities. For a retailer, the main subject is the sales process because it generates the company revenue. Integrated is the most crucial aspect. It is related to the standardization of data that comes from different sources. Once data is fed to a DW, it must be converted to a single type to guarantee accuracy in answering business questions. Nonvolatile means that data in DW are stored once and rarely updated. Time-variant indicates that DW stores data related to a certain point in time.

Moving from OLTP to a DW architecture is not a trivial task. A crucial process is the DW design process. Designing a DW is a complex task that must be carefully conducted to produce reliable information and assure performance in answering analytical business questions. According to Malinowski and Zimányi (2009), there are two approaches to design a DW:

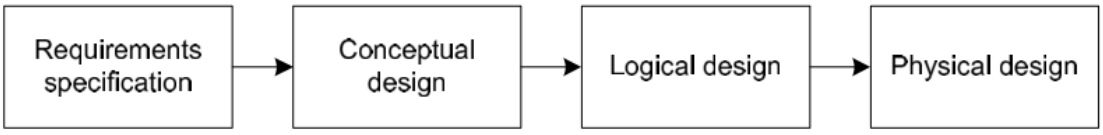
- Top-down design: first, the requirements from different users are collected and analyzed; then, a single DW schema is created for the entire company. Data Marts can be implemented after the integrated DW to meet the needs of different decision-makers users or departments. A Data Mart is a piece of a DW that stores compact and aggreged information, different from DW, where data is very granular.

- Bottom-up design: in this approach, proposed by Kimball and Ross (2002) as a *data warehouse bus architecture*, Data Marts should be first built in a star schema structure based on user's requirements, and later on, incrementally combined to create a global DW.

Inmon et al. (2008) postulate that the bottom-up approach can lead to data inconsistency and redundancy problems. Designing separated Data Marts may result in different query results creating discrepancies between the same business analysis. Maintaining an environment with many data marts is also another issue because it increases the efforts to insert or update data, and increases the chances of error propagation. On the other hand, the bottom-up design is more flexible and can be implemented faster because a Data Mart is a limited representation of the entire company; i.e., it has a lower volume of data.

Malinowski and Zimányi (2009) proposed four phases in DW design that can be applied independently of the chosen approach (top-down or bottom-up). The two essential processes of DW design are the requirements analysis and conceptual design because they represent the connection between the user's requirements and what the software will deliver. The requirements specification phase is divided in four approaches: (i) user-driven considers that all users from different levels and departments are important and should be included on the requirements specification; (ii) business-driven is more related to business requirements or business processes than the users itself; (iii) source-driven (or data-driven) considers the operational source systems as the main component of the requirements and user participation is secondary; (iv) combined approach (top-down/bottom-up analysis) is the combination of business- or user-driven and data-driven approaches.

After defining the requirements, the next phase consists of creating a multidimensional conceptual model where dimensions, hierarchies, and ratios can be visualized. The third and fourth phases in DW design are more technical. They consist of implementing the DB logical design, which will be used to create the corresponding physical DB model to be fulfilled with data and queried later on. An overview of all phases is shown in Figure 2.1.



Source: Malinowski and Zimányi (2009)

Figure 2.1 Phases in Data Warehouse design

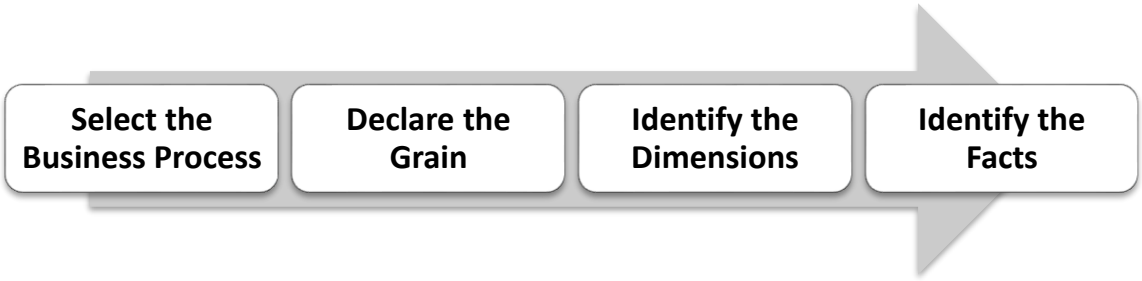
The conceptual design is typically an evolutionary process where users and DW designers discuss together the best solution. User feedbacks are used to improve the conceptual model. What should be avoided at all is starting a DW project without first modeling the dimensions, dimension attributes, and ratios in a conceptual language, independent of a

DB technology physical schema. It is bad practice to start a DW design with a star schema definition (Sapia et al. 1998).

To support the conceptual design phase, Kimball et al. (2013) proposed four steps to design dimensions and facts efficiently. The first step consists of identifying and selecting the business process the company wants to analyze. Attention shall be given to business processes that produce business performance metrics, such as processing orders, registering items, delivering goods.

The second step is the grain declaration. It means defining exactly the atomic level of a fact which represents a single row in the fact table. In the delivery process, for instance, a clerk delivers a product to a customer at a specific time. A corresponding row in the fact table would have a clerk, a product, a customer, and a delivery time as an individual record.

The next step is the dimension identification. If the business process and the grain were correctly defined, dimensions should be easily identified as the textual description of each ratio. Customer, product, clerk, location, supplier, and time are typical dimension examples in retail companies (Becker and Schütte 2004). The last step is the identification of facts, which are the true representation of a grain defined in the second step. Figure 2.2 shows the four steps sequentially.



Source: Kimball et al. (2013)

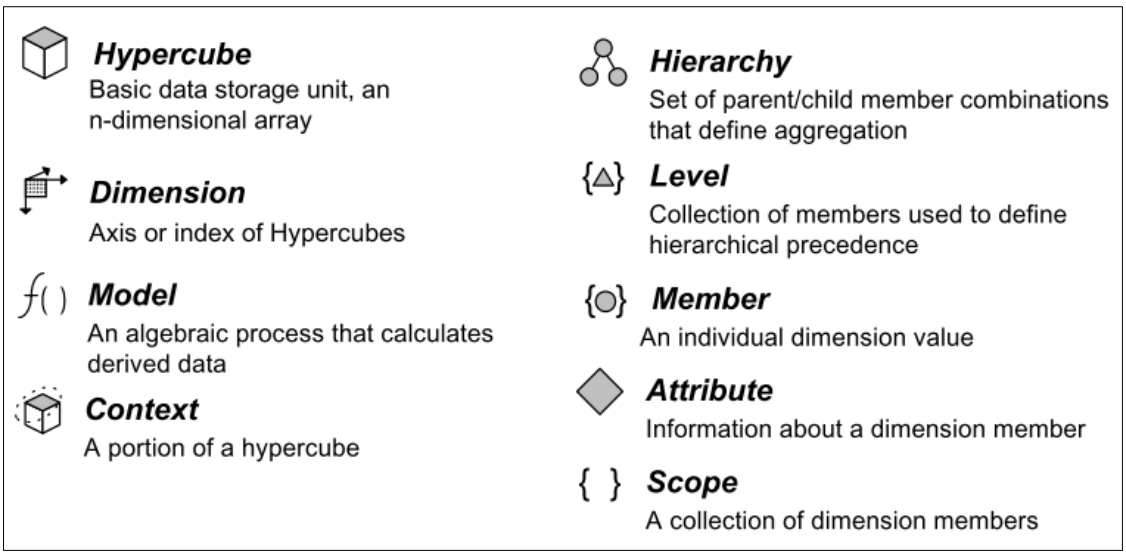
Figure 2.2 Four-Step dimensional design process

After the identification of facts, dimensions, and ratios, a multidimensional model approach needs to be selected. To date, there is no standard on multidimensional modeling language. A broad discussion about the different approaches (DFM, ME/RM, Star E/R, UML) can be found on Manuel Serrano et al. (2004) and Saroop and Kumar (2011).

According to Becker et al. (2006), only three approaches contain specific elements for designing a multidimensional model: ME/RM, DFM, and ADAPT. The Multidimensional Entity-Relationship Model (ME/RM) is an adaptation of classic ER models. The aim is to represent hierarchies and their relation with facts maintaining the

basic structure of ER diagrams. The Dimension Fact Model (DFM) is a dimensional schema where the fact is at the diagram's center. Dimensions and hierarchies are connected to the fact, which contains the fact measures.

The Application Design for Analytical Processing Technologies (ADAPT) is a specific approach for modeling multidimensional data that offers the most significant number of elements needed to represent ratios, dimensions, and hierarchies. The ADAPT model represents an abstraction of a cube with dimensions and hierarchies without a physical model prescription, i.e., it can be used independently of the chosen physical model. Figure 2.3 shows the nine ADAPT objects.



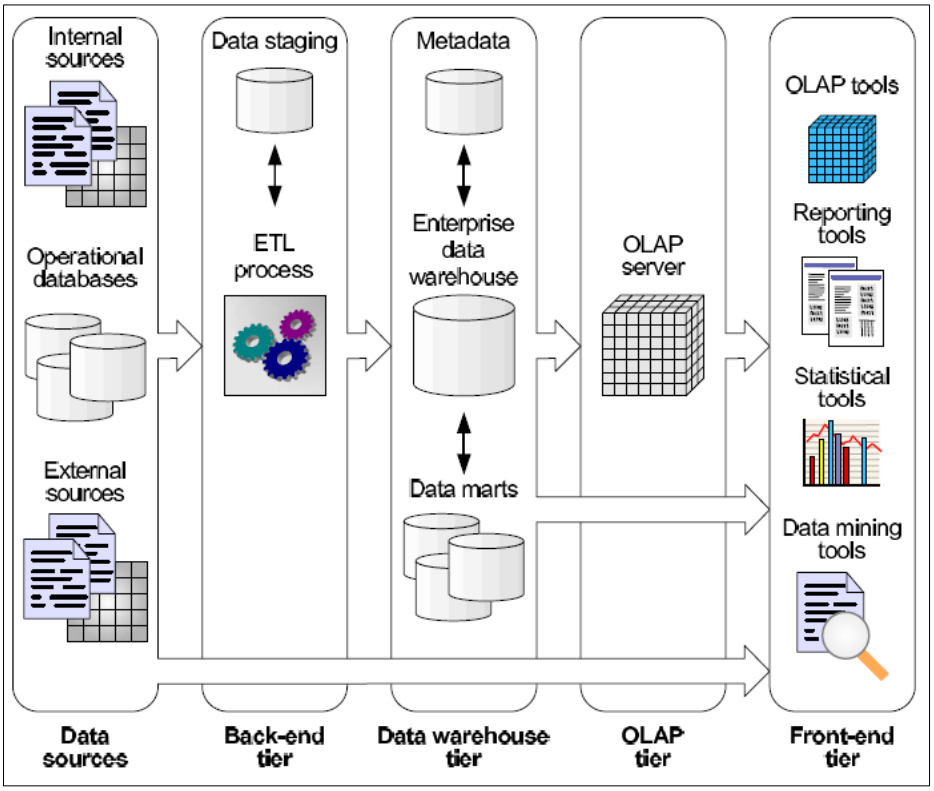
Source: Bulos and Forsman (2006)

Figure 2.3 ADAPT objects notation

The core objects of ADAPT are hypercubes and dimensions. A hypercube contains n-dimensions related to a specific business process analysis (sales, delivering, claims). A hierarchy includes parents and children levels related to a dimension, and it shows how a dimension can be aggregated (a dimension can have more than one hierarchy). Member, attribute, and scope are elements related to a dimension. A member represents a single dimension value, e.g., Germany is a member of the Location dimension. Attributes describe the member's characteristics of a dimension. The scope object is a subset of dimension members. It is useful to store calculations or limit the analysis of a particular group. The model object represents a set of formulas that can be applied to the data. A context represents a specific part of a cube, e.g., splitting a cube into national and international sales (Bulos and Forsman 2006).

Once requirements were set, and conceptual models were built, the next step on DW design aims to translate the abstracted models to real data mapping from source data to a DW database. This process is called Extracting-Transformation-Loading (ETL). Extracting is the first task where source data were read from different data providers (usually files and databases). Next, data are stored in a Data Staging Area (DSA). The transformation tasks consist, for instance, of cleaning, removing duplicates, standardization of categories, and removing domain conflicts. The transformation step assures that data will be sent to a DW system without inconsistencies. Once the transformation step is done, data are finally loaded into the DW database, which corresponds to the Loading task. (Kimball et al. 2013; Vassiliadis et al. 2002).

Malinowski and Zimányi (2009) employed the notion of *tiers* (layers) to describe a typical DW architecture. The back-end tier contains ETL tools that read data from internal or external data sources, and stores transformed data into the DW tier. The DW tier has a single DW database or several Data Marts, and the metadata repository responsible for maintaining the information about the DW. The OLAP tier contains the OLAP server that offers the computation power for OLAP operations. The front-end tier represents the user interface where analytical tools are available to DW/BI users query data. An overview of all these concepts and their connections is shown in Figure 2.4.



Source: Malinowski and Zimányi (2009)

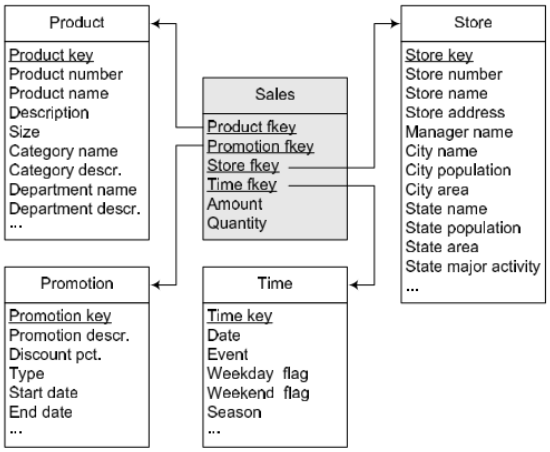
Figure 2.4 Data Warehouse Architecture

The next three sections will describe three different architectural choices in the DW design process: star, snowflake, and Generic DWH. The choice of one or another will impact on the ETL design process and the way of building SQL queries. These different architectural choices are an essential part of this study because they will be tested against each other in terms of ETL and query performance.

2.2 Star Schema

The star schema is a logical DB model that consists of one fact table in the center and dimensions tables on the border. Its name is related to the star or asterisk form. A star schema is entirely different from an OLTP DB because the star is not on the third normal form[2] (3NF). An OLTP on 3NF would store the relation "state – city – store" in three different tables named accordingly. Three joins have to be performed to group stores by state.

In a DW architecture with a huge volume of data, joining many tables reduces query performance significantly. The rationale of using a star schema lies in reducing the number of joins through denormalization to improve query performance. However, repetitions in dimension tables will occur; hence, more space is needed to store data. Star schemas are also more complex to read because they did not represent hierarchies as users think. A star schema can be combined, forming a galaxy schema (or fact constellation, where dimensions are connected to one or more facts). Figure 2.5 shows an example of a star schema.



Source: Malinowski and Zimányi (2009)

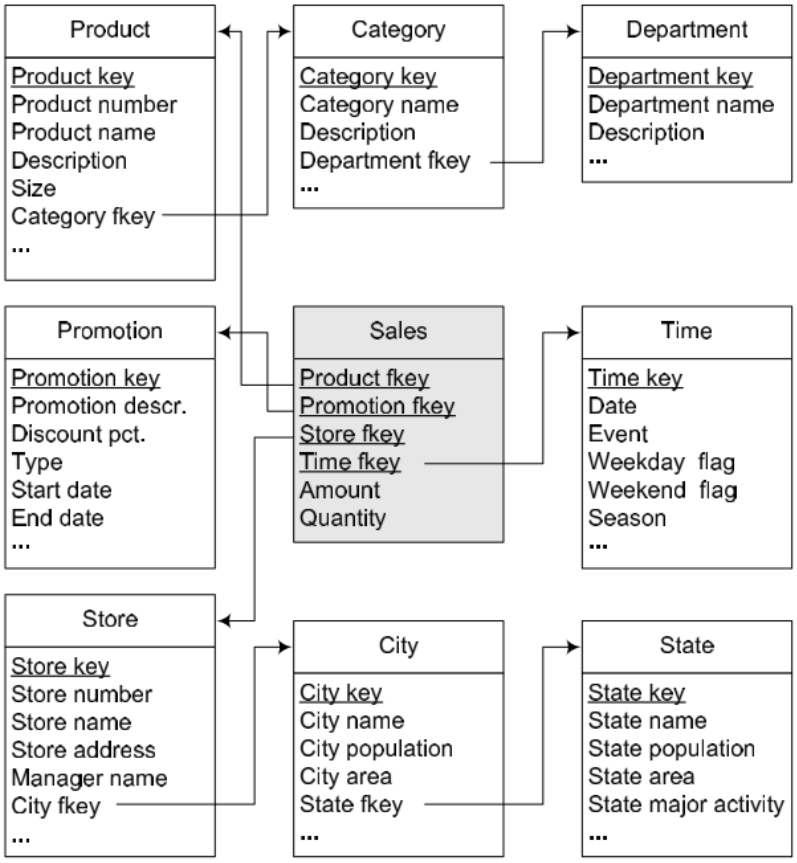
Figure 2.5 Star schema example

[2] The term is used in relational databases to ensure referential integrity, avoiding duplications and anomalies on data. A database logical model on the third normal form is also easier to understand because the relations are clearly identifiable.

Kimball is the principal responsible for popularizing the star schema. He is using the star schema on his DW architecture since the 1990s. Due to his close relation to the star schema, criticism has arisen upon his architecture through the drawbacks of a star schema. However, his architecture suggests the use of star, but it is not only made of a star schema. In ETL tasks, for instance, tables can be used in the third normal form on the staging area. This is an acceptable procedure, according to Kimball's dimensional DW design (Adamson 2010).

2.3 Snowflake Schema

The snowflake schema can be understood as a normalized extension of a star schema. The term *snowflaking* is often used to represent the act of normalizing star dimensions to the 3NF. The transformation of the star example from Figure 2.5, into a snowflake, is shown in Figure 2.6. In the snowflake, the table store has its hierarchy levels city and state normalized into two new tables named accordingly. The same occurred to the product table: category and department fields were normalized (extracted) to separated tables, and FKs were added to refer the corresponding hierarchy levels.



Source: Malinowski and Zimányi (2009)

Figure 2.6 Snowflake schema example

The advantages of the snowflake are the disadvantages of the star. First, snowflake stores fewer data as a result of denormalized tables; nevertheless, more tables mean more joins and higher query time. Snowflake represents hierarchies as users think, which is helpful in the translation from conceptual to logical models. Martyn (2004) suggests that it is not always the case that a star schema has a better performance on querying than a snowflake. He shows that, if a dimension table on star schema stores a huge volume of textual information, the query performance will reduce drastically. In such a case, the snowflake has better performance.

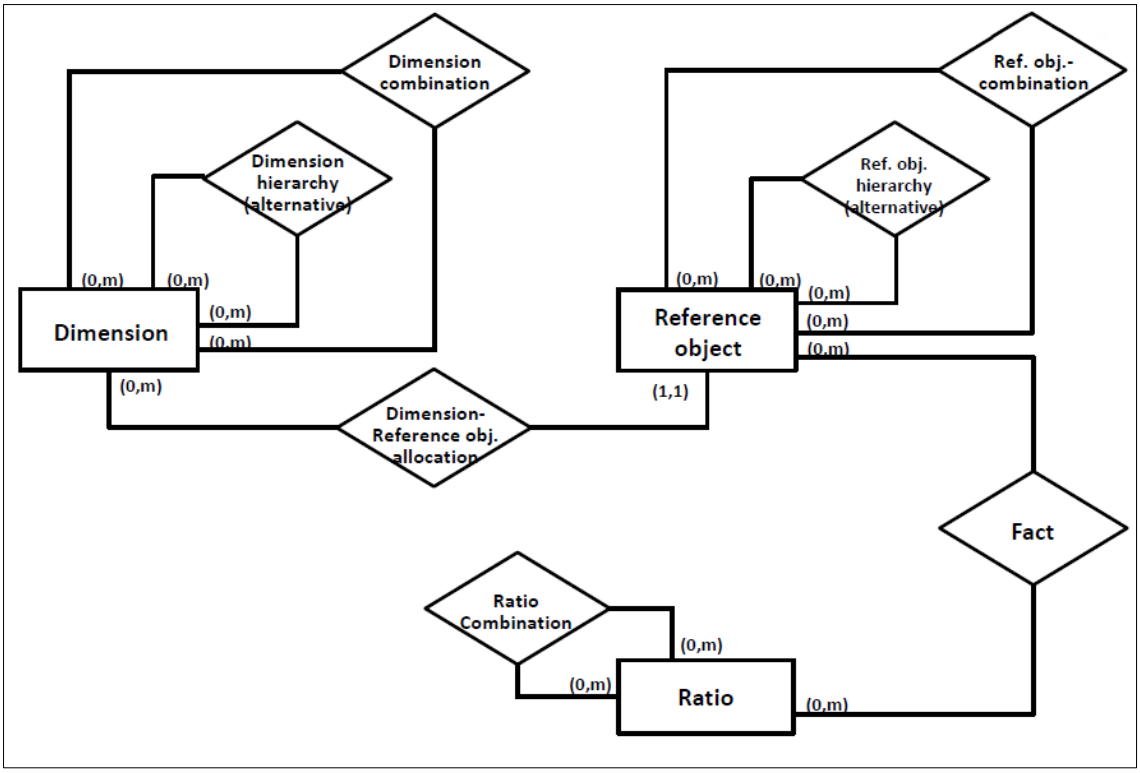
2.4 Generic Data Warehouse Schema

Becker and Winkelmann (2019) revisited the issues related to star and snowflake schemas and proposed a new multidimensional schema. They argue that star and snowflake grow fast and disorderly because, for each new dimension or ratio entity insertion, new tables must be created or updated. Consequently, database size grows to hundreds of tables in which even experts can quickly lose control. Maintaining such a vast and complex database incurs an intense administrative effort. Data stored in a star or snowflake schema are also inconsistent and plenty of redundancies.

To work around the disadvantages of star and snowflake, Becker and Winkelmann proposed a new multidimensional schema named Generic Data Warehouse (GDWH) represented as an ER model in Figure 2.7. The same model was also found in Becker and Schütte (2004, p. 631) with the German name *Komprimiertes Data Warehouse Modell*.

The GDWH schema consists of nine tables that can represent any DB as a multidimensional model. This schema is in a high level of abstraction, where dimension, reference object, and fact are the core elements. A reference object is related to only one dimension, but a dimension has zero or more reference objects. With the (0,m):(0,m) relationship for hierarchy (alternative) and combination, it is possible to store all hierarchies and combinations for the same atomic level. For time dimension, it is common to store different hierarchies; for instance, a quarter or a semester refers to a month.

The dimension table, which corresponds to the dimension entity, saves the dimensions of a DW as rows. The dimension store and its hierarchy levels city and state, for instance, are rows in the dimension table. The dimension hierarchy relation represents the hierarchies for each dimension level, e.g., state is recorded as parent and city as a child element. It is also possible to save dimension combinations relations, for instance, city and product (in this case, a new dimension city-product will be saved in the dimension table as well).



Source: Becker and Winkelmann (2019)

Figure 2.7 Generic Data Warehouse ER model

The instantiation of dimensions is saved in the reference object table, e.g., Münster (city dimension), NRW (state dimension). The reference object hierarchy holds the information of hierarchies for each reference object, e.g., NRW is the parent of Münster. The reference object combination represents the dimension combinations allocations; for instance, Münster – Bicycle is the instantiation of the city-product dimension combination.

A fact is the result of a relationship between a ratio and a reference object. Ratios are business performance indicators, e.g., revenue, sales volume, cost. The ratio combination relationship corresponds to formulas used to calculate derived ratios, e.g., profit is equal to revenue minus costs. The fact table in GDWH stores a row for each ratio, different from star and snowflake, where each ratio constitutes a column in the fact table.

The administration of DW metadata in a GDWH implementation is straightforward. If new dimensions or facts have to be added to an existing GDWH database, only new rows in the corresponding tables have to be inserted. That means maintaining a GDWH database requires less effort if compared to star and snowflake schemas. The risk of generating inconsistent data is reduced. GDWH is also suitable for decision support

systems implemented as a database application without the complexity of a typical DW architecture.

2.5 Related Research

This work is a sheer performance comparison between DW architectures. It is not a benchmark standard specification because the goal is not related to hardware or environment performance comparisons. Nevertheless, the related research literature is about decision support benchmarks. The process of building a good benchmark is also connected to this study because it highlights the main aspects that need to be taken into account to produce consistent performance results and guarantee reproducibility. Researches about ETL modeling languages and ETL performance improvements are also connected to this study because ETL will be designed and implemented for each DW architecture.

Building a good benchmark is not a trivial task. Huppler (2009) proposes a guideline consisting of five key aspects to support the development of a benchmark. The first is about relevance. A benchmark should reflect something important, connected to a real industry problem, and consists of an objective metric that must be easily understood by its target audience. Second, a benchmark needs to be repeatable, that means if one runs it, over and over, the results should remain the same. The third point is about being fair/portable. Portability is not a real problem anymore. With the use of Object-Oriented programming languages and SQL, benchmarks can be implemented in different environments without issues. To be fair, a benchmark needs to be able to use the different technologies available on the market (column storage, in-memory environments, database accelerator). The fourth point is about to be verifiable. A benchmark needs to have a high degree of confidence, to guarantee that, it must be possible to verify if the results change once inputs have changed. Simply benchmarks are easily verifiable, but more complex benchmarks have to be evaluated from certified auditors. The last point is economical. A benchmark needs to be economically affordable in terms of hardware and software resources.

DW benchmarks should also follow these guidelines to achieve its goal, but the lack of standards to design a DW makes it a complicated task. The Star Schema Benchmarking (SSB) proposed by O'Neil et al. (2006) transformed the TPC-H schema into a star schema following the recommendations of Kimball and Ross (2002). The SSB contains a set of four queries and their variations (using roll-up and drill-down operations). The query definition on SSB does not refer to any specific business question; instead, it follows only technical aspects regarding "functional coverage (different common types of Star Schema queries) and selectivity coverage (varying fractions of the LINEITEM table that must be

accessed to answer the queries)" (O'Neil et al. 2006, p. 4). SSB is a DW benchmark standard definition and does not contain any performance results of implemented tests. An implementation of SSB and its results can be found in Michael Stonebraker et al. (2007) and Sanchez (2016).

The study of Bizarro and Madeira (2002) proposes a performance-oriented perspective on DW design. They claim that DW performance improvements (indexes, new types of joins, data reduction, sampling, real-time information, materialized views, histograms, statistics, query plan optimizations) occur only after the DW physical implementation. They suggest that the performance issue must be considered together on the business requirements analysis step and before the logical design phase. To guarantee query performance, DW designers should be informed beforehand by business users about which operations (queries, joins, aggregations) are likely to appear frequently. Then, the DW physical model can be tuned based on these frequent operations to improve query performance. They compared the query time between the original TPC-H schema and their improved schema derived from their approach. The results have shown a significant performance improvement: up to 500 times in some queries.

Darmont et al. (2007) developed a new benchmark to test the index influence in different DW architectures and sizes since no benchmark until that time fitted their needs. They developed a Java application called DWEB (the *Data Warehouse Engineering Benchmark),* which is capable of generating star, snowflake, and constellation schemas with different amounts of facts, dimensions, and hierarchies tables. The application also has a workload routine. The results showed that the use of index improves response time, especially for queries with large results. But in bigger databases, the performance gain is softened because the fact table is sparser and indices grow as much as facts.

The most recent DW benchmark found was the research of Campbell et al. (2017). They posit that benchmarking data warehouses on a more technical level still not yet thoroughly explored. After addressing this gap, they developed two different approaches to explore the efficiency of data warehouses systematically: the Sorted Inverted Index Cube (SIIC) and the Dominant Answer Materialization (DAM). They used the TPC-H and the weather data set to test the developed algorithms and found out that DAM performs better on progressive data cube computation. Their contributions are more related to the computer science area since no clear definition of business requirements or business questions was taken into account. Nevertheless, it is still relevant to this research because it explores multidimensional benchmarking using the TPC-H data source with different sizes.

ETL literature is also related to this study. According to Demarest (1997), ETL tasks can consume up to 80% of the total time of a DW implementation. The time-consuming tasks

of ETL are partially justified by the volume of data being transferred. However, transformation tasks alone can be rather complicated, which mostly explains the ETL total time. Three approaches have been proposed to optimize ETL time by enhancing ETL conceptual modeling techniques: Vassiliadis et al. (2002), Trujillo and Luján-Mora (2003), and El Akkaoui and Zimanyi (2009).

Vassiliadis et al. (2002) introduced a new graphical notation specific for ETL tasks. Vassiliadis' model offers elements to model all ETL activities but does not allow ETL decomposition into sub-processes. Trujillo and Luján-Mora (2003) proposed an ETL conceptual model approach based on UML standard notation. Their model represents fundamental ETL activities (constraints, mapping, filtering) only as UML notes, where no restrictions can be applied. El Akkaoui and Zimanyi (2009) created a new set of ETL elements based on the Business Process Model and Notation (BPMN) and translated them using Business Process Execution Language (BPEL) to generate executable ETL code. According to Bala et al. (2016), each ETL modeling language has its pros and cons, and there is no de-facto standard to date.

Two main approaches were identified after analyzing literature about performance comparison on DW architectures: performance improvements before or after DW physical implementation. Before means that performance should be considered as a requirement in the DW design phase. In this approach, logical and physical data modeling have a significant impact on the DW solution. Variations on logical DW models like star, snowflake, and GDWH should be considered as optimizations before the physical implementation because each model has its DW implementation workflow.

Performance improvements after DW physical implementation mean the use of optimization techniques, for instance, indices, materialized views and partitioning. These techniques are applied after the DB physical model is defined, and data were already loaded. The two different approaches are not necessarily excluding each other because one can first optimize different DW data structures and, after on, optimize the stored data with optimization techniques.

It was found that the mainstream of current research is related to the Star Schema Benchmark (SSB) since Kimball's DW dimensional model using star has become popular. SSB is the most recurrent implemented DW benchmark, but the problem of SSB is that it was designed only to test different environments (hardware and software) without varying the DW architecture itself. DWEB, instead, considered three different DW architecture variations: star, snowflake, and constellation schemas. But the objective of DWEB was to test index techniques after the physical implementation.

No study considering the GDWH on DW performance comparison was found. This research extends the DW performance comparison debate adding the GDWH architecture to the discussion. A formal ETL specification for GDWH was also not found in the literature.

3 Research Approach

3.1 Overview

The starting point of this research was the literature review. The research was done on three databases: Scopus, Web of Science, and ScienceDirect. The objective was to find literature about Data Warehouse concepts and applications, and Data Warehouse architectures performance comparison. The literature review and the related research results are presented in chapter 2.

A data source is needed to put the DW architecture concepts into practice. That is why a careful process of data source evaluation was performed to assure that the selected data will fit the goal of this study. The final result critically depends on the input data. The output of queries after the DW design must be consistent with the initial data. TPC-H was the selected data source to be used in the case study. An overview of the applied methodology is shown in Figure 3.1.

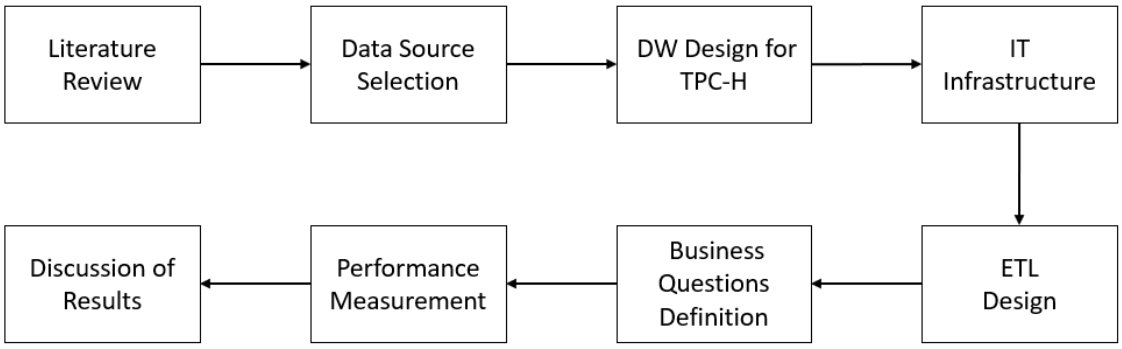


Figure 3.1 Research approach overview

In the DW Design for TPC-H, the data source will be analyzed to extract the dimensions, facts, and ratios that will be used in the DW architectures. Next, the IT infrastructure will present the software and tools used to perform the ETL processes and to run the queries. The ETL design section shows step by step how ETL was performed for each DW architecture.

With data transformed and loaded into the DWs, ten business questions were defined and translated into SQL queries in the performance measurement section. The discussion of results section analyses for each DW architecture: the ETL and query performance times, the DB size before and after ETL, the metadata management and data consistency, and redundancy.

3.2 Methodology

The first step of this research was the literature review. The terms used on the search were: “data warehouse literature review”; “data warehouse and/or BI and/or OLAP”; “benchmark data warehouse”; “benchmark multidimensional”; “performance data warehouse”; “star snowflake performance”. Two thousand two hundred nine papers were found with these search terms. A second filter was done to select only IS and Computer Science Journals for quality reasons. The final number of papers was sixty-two. The abstracts were read, and if the content was considered relevant to this study, the paper was completely read. After reading all selected papers, another ten papers were added based on the citations.

The literature review structure is based on the core concepts of DW architecture design. Kimball et al. (2013) propose a bottom-up approach to design a DW, while Inmon et al. (2008) criticizes this method and suggest a top-down approach to build an integrated DW solution. Malinowski and Zimányi (2009) show an integrated DW architecture view, which will be used in this study to build the DW solution using the top-down approach.

The goal of this study is to compare performance between DW architectures, but this research is not a complete benchmark specification because no workload was developed. However, the related research was framed in the DW benchmarking field because it partially shares the same goal of this research – performance comparison. Insights of published DW benchmarking were used as guidelines to develop this research. Attention was also given to ETL literature because 80% of the total time of a DW design is related to ETL tasks, according to Demarest (1997).

Data Source Selection

The evaluation criteria of data sources were related to data structure and consistency, automated workload solution to scale data, ETL, documentation, and public access. After Internet research, three relevant data sources were found based on the mentioned criteria: ABP-1 OLAP Benchmark[3], TPC-DS Benchmark[4] , and TPC-H Benchmark[5].

The APB-1 OLAP Benchmark comes with a pre-configured ETL with fixed dimensions and ratios. Data generated is already transformed into a multidimensional form, but adapting the ETL process to different DW architectures would be a complex task. The

[3] http://www.olapcouncil.org/research/resrchly.htm
[4] http://www.tpc.org/tpcds/
[5] http://www.tpc.org/tpch/

last update was in 1998, which probably means the life-cycle end. Due to the complexity of its structure and ETL, this data source was considered not suitable for this study.

The next two data sources analyzed are maintained by The Transaction Processing Performance Council[6] (TPC), a non-profit corporation that aims to develop and publish benchmarks to measure transaction performance to a variety of business and computer functions. TPC-DS is a decision support benchmark that comes with a set of tools to generate synthetic DW data from a retail company using the snowflake architecture. It comes with an ETL solution and it is scalable in size. The problem found in TPC-DS is the complexity of its ETL. The constellation schema generated is fixed and cannot be easily transformed into another DW architectures. (Darmont et al. 2007).

TPC-H is an ad hoc decision support benchmark that has a formal data structure definition and a workload to generate synthetic data with different sizes. TPC-H represents data from a generic retailer that buys and sells products around the world. The toolkit freely available for download comes with a specification document where all relevant information can be found. The workload solution generates consistent and structured data as a typical relational DB model.

Other important aspects observed in TPC-H were: mature state of development and documentation (since 1999); continuous updates (new benchmarks published from time to time and improvements on current benchmarks); a significant number of implementations and technical articles published.

Compared with the other two data sources analyzed, TPC-H was considered the most suitable for the study's goal because no ETL was previously executed, data is structured and documented. The data also represent a relevant business case scenario – retailer – to apply the DW architecture concepts. The complete analysis of TPC-H data will be carried out in the DW design section.

Data Warehouse Design for TPC-H

In this step, the DW will be designed based on the TPC-H data. First, data will be analyzed to extract dimensions and hierarchies. The entities and relations will be evaluated based on business relevance. A data-driven approach based on the business analysis of TPC-H data will be used to build the DW. The four-step dimensional design process proposed by Kimball et al. (2013) will be applied to the TPC-H data: business process identification, the grain declaration, dimensions, and facts identification.

[6] More info at http://tpc.org/information/about/abouttpc.asp

After the identification of dimensions, hierarchies, and facts, the multidimensional model will be created using ADAPT notation (Bulos and Forsman 2006). No logical model was implemented because only a single DB technology will be used to manage data. The physical DB models for each DW architecture will be derived from the multidimensional model.

IT Infrastructure

Before creating the required physical DBs and starting the data source generation, the system architecture was designed to meet the needs of this research. The IT Infrastructure section will describe the systems, software, and hardware used to build the DW architecture. This study will use MySQL and Linux to measure the performance of the three DW architectures. The ETL tasks were design and executed on Pentaho Data Integration (Kettle) running on Windows 10.

A Java web application called Data Warehouse Design Query (DWD-Q) was developed to offer an interface to query the different DW architectures and sizes. The development process and how queries are generated based on the user's request will be described. Only open-source software was used in this research to guarantee reproducibility. The development strategy and patterns adopted will also be described because it is relevant to the IS field of study.

ETL Design

After the data generation and the DW design phases, the ETL design will be executed. As no standard language exits to model ETL (Bala et al. 2016), an interactive and evolutive approach will be used to define the steps necessary to read the data source, transform, and load standardized data into the DW tier. A unique ETL will be designed for each DW architecture choice and executed for the scale factors one and ten.

Three ETL variations will be designed to GDWH:

- No combination (NCB): The formal GDWH specification will be used. The saved facts are related to only one dimension combination, which contains all atomic levels.

- Dynamic ETL (DYN): For each query request, a new dimension combination and the corresponding reference objects and facts are saved.

- All combinations (ACB): An adaptation of the formal GDWH schema will be implemented as a trial to improve performance. The idea is to store all possible

dimension combinations and facts to reduce joins and improve query performance. Tables reference object combination and reference object hierarchy will not be used on queries anymore.

Attention will be given to performance measurement on ETL tasks. The time spent for each ETL job on different DW architectures and scale factors will be collected to further analysis.

Business Questions

A DW is meaningful only if it can efficiently answer DW/BI user's questions related to the selected business process. Ten business questions were defined and answered with the three DW architectures. The business questions will be defined to represent typical sales performance analysis from a hypothetical retailer company.

The dimensions and ratios previously defined in the DW design phase will be used to build the questions. Only the sum of each ratio will be used to avoid variations on query comparison. Functional and selectivity coverage, introduced by O'Neil (1993), are technical aspects of query definition that will be respected as far as possible.

Performance Measurement

The first step in the performance measurement phase is the definition of query flights, which will be derived from the business questions. A query flight is the SQL query translation of a business question using a set of predicates. Year and Region, for instance, are two predicates that can be used in a query flight. One business question can have more than one query flight.

Sixteen query flights were derived from the business questions and will be executed three times for each DW architecture. For each executed query, a start and end time will be collected using the *SYSDATE()* function on MySQL. The *TIMEDIFF()* function will be used to calculate the difference between the end and start time, which corresponds to the query running time. The number of output rows will be useful to check if the queries were corrected translated for each DW architecture. The returned values will also be collected to check data consistency.

ETL performance will also be considered in the DW architectural performance comparison. The ETL running time will be collected for each DW architecture and for each scale factor. The different DB sizes before and after the ETL process will be collected to analyze the relationship between performance and DB size.

Discussion of Results

All information collected in the performance measurement phase will be discussed appropriately. The there DW architectures (plus two GDWH ETL variations) will be classified using a rank position from first to fifth for each evaluation criterion. The idea is to analyze how each DW architecture performs with scalable data from SF 1 to SF 10. The estimation of query and ETL times for scale factors 100 and 1,000 will also be analyzed.

The main focus of the performance discussion will be query running time because that is what matters for DW/BI users. As this study is also addressed to DW architects, the criteria related to ETL performance time, metadata management, and data consistency and redundancy will also be considered. The results of the discussion section can support the decision of which DW architecture is more suitable for each scenario.

4 Data Warehouse Design for TPC-H

4.1 TPC-H Data Analysis

A good starting point for DW designers is to analyze the logical data model from a data source, without forgetting the DW business goals. The TPC-H logical schema is reproduced in Figure 4.1. The model shows that TPC-H was designed as a typical relational database. Eight tables and nine relations were used to represent an OLTP system of a hypothetical retail company. An Entity-Relation model was created to support the TPC-H data analysis. (see Appendix A, Fig. A.1).

Customers and suppliers are split by geographical locations identified by a nation key. Each nation has a unique region, for instance: Germany – Europe. The phone number was generated based on the nation id, but without considering any other location attribute. The address was randomly generated. The lowest level of location information is the nation. Customers are classified into five different market segments: Automobile, Building, Furniture, Machinery, and Household. A customer has none or many orders.

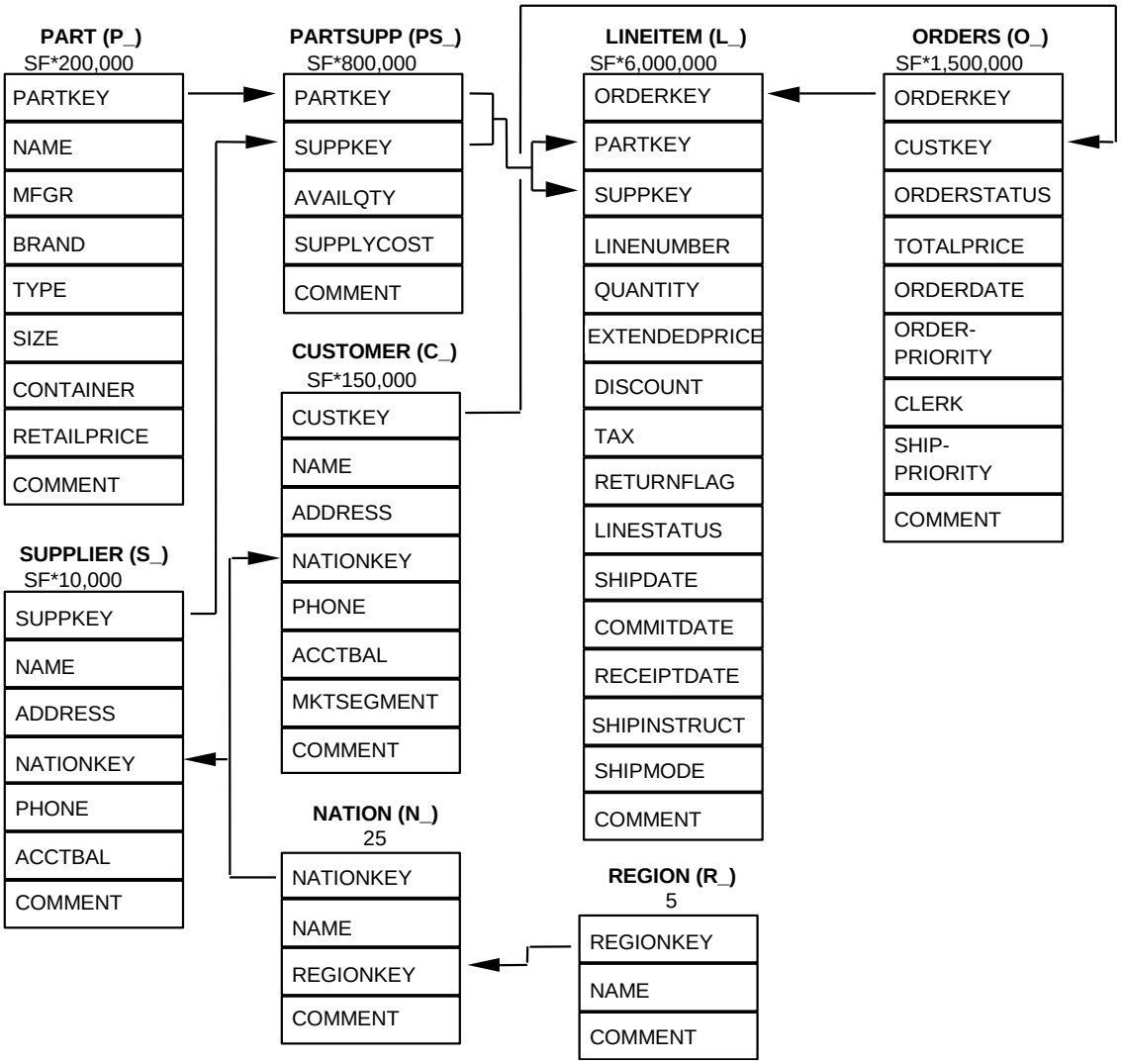
The table *part* stores the product information. Each product has a manufacturer group (MFGR field), a brand, and a type. A manufacturer group has many brands, and a brand has many product types. Product size and container are randomly generated and have no relation to the product brand or manufacturer group. The retail price (list price) of each product is saved in the *retailprice* field.

The relation between products and suppliers is stored into the *partsupp* table. The *supplycost* field is the source of the product cost information, and it is defined based on the supplier selection. A supplier can not be classified based on manufacturer group, product brand, or product type because there is a many to many relation. An issue was found in the *partsupp* table: the product cost and available quantity are immutable over time. This problem does not affect data consistency itself, but it is important to consider it in product cost historical analysis.

The *lineitem* table contains the product entity information represented as items. An item has quantity, price, discount, tax, and shipping information. The *linestatus* field stores the shipping status for each item. The *extendedprice* field holds the retail price information multiplied by the quantity ordered. The total order price will be defined based on the extended price minus discount, multiplied by the tax.

Each *lineitem* record is related to an order. Each order has only one customer, but a customer may not have any order (according to the TPC-H specification). The order status

is based on the line item status. If all items were already shipped, the order status is set to "F"; if no item was shipped set to "O"; otherwise is "P". The order priority can be set from 1-Urgent to 5-Low. The clerk field in the *orders* table represents the clerk's name, who created the order.



Source: Transaction Processing Performance Council

Figure 4.1 TPC-H logical schema

Four date fields were found in the TPC-H schema, three in the *lineitem* table, one in the *orders* table. In the *lineitem* table, date information was used to store the shipping status for each item. This can be used to analyze logistic data, for instance, the delivery time for each ship mode. The *orderdate* field represents the date in which a clerk created the order.

4.2 Four-Step Dimensional Design Process

The four steps of the dimensional design were applied based on the recommendations of Kimball et al. (2013), already introduced in the research background section. The first step defines the business process, which the DW cube will be built. Then, the grain declaration consists of the definition of the atomic level. Next, dimensions and hierarchies will be defined, followed by the facts and ratios identification. A multidimensional model will be created to represent the conceptual model of TPC-H DW.

Select Business Process

Different business processes can be selected to build a DW schema, but only a single process for each data cube should be chosen to limit the level of representation of a multidimensional model. The TPC-H schema was designed to store data about suppliers, customers, products, and items sold, which opens the possibility for different business processes analysis. It can be chosen more for the supplier relationship management side or more product and logistics analysis.

In this study case, only the sales process was selected because it represents a typical customer-centric analysis. Sales and customer behavior identification are relevant analysis for strategic business development. The clerk plays an essential role in the sales process because s/he is responsible for creating orders based on customer's needs. The product cost varies according to the supplier selected by a clerk on the ordering process. The criteria for choosing suppliers have a significant impact on profit generation. The customer is also closely related to the sales process because products can not be sold without being ordered.

Declare the Grain

With the successful definition of the business process, the grain declaration could be carried out. It is essential to identify, in business terms, the exact representation of a single row in the fact table. The grain identified in the sales process of TPC-H consists of a Clerk Name, Customer Name, Product Name, Supplier Name, and Month. That means a single fact row stores the ratios of these atomic levels. All other fact values will result from grouping or relations obtained from these grains.

Although the *orderdate* field contains the complete day information (year, month, and day), sales data were grouped by month. Grouping by month was applied in cases where the same clerk sold the same product from the same supplier to the same customer in the same month. Ratios were calculated and grouped using this logic.

Identify the Dimensions

Usually, when the grain declaration was correctly performed, the dimensions can be easily identifiable because the grain represents the lowest level of a dimension. Thereby, looking at the business process and the grain declaration, it was found that the sales data cube should contain the following dimensions and hierarchies:

- **Clerk**: the clerk dimension contains only the Clerk Name atomic level since no other hierarchies were identified on the business questions.

- **Customer**: customers were arranged by market locations and market segments. In the market locations hierarchy, customers can be grouped by regions and nations. Alternatively, customers can be group by market segments. That means, there are two possible paths to achieve the Customer Name atomic level: via market segment, or via market location.

- **Product:** products were also arranged by two different hierarchies: product type and manufacturer groups. The product type hierarchy contains only one grouping possibility – the product type itself. In the manufacturer group, products can be grouped by brand and manufacturer groups. The Product Name is the atomic level.

- **Supplier**: no aggregation was needed at the supplier level. It contains only the Supplier Name atomic level.

- **Time**: contains the years and months extracted from *orderdate*. The atomic level is the Month Name, which contains the corresponding years aggregation.

Identify the Facts

The last step of the dimensional design is the fact definition. If all precedent tasks were successfully carried out, the fact should be a consequence of the grain declaration itself. Ratios were extracted based on the relation with the sales process and calculated according to the TPC-H specification document. All the identified ratios can be summed across all dimensions. Discount and tax rates were used only to calculate the selling price and product cost. The following ratios *will* be present on the DW architectures:

- **Revenue**: corresponds to the revenue generated by a single product sold to a customer by a clerk and grouped by month. The tax was subtracted from the field l_extendedprice to obtain the revenue. The quantity of products sold is already included in the l_extendedprice field.

- **Product Cost**: represents the cost of a product according to the selected supplier.

- **Profit**: it is a derived fact which represents the difference between Revenue and Product Cost. It was computed in the ETL process and stored on the fact table.
- **Selling Price**: consists of subtracting the discount from the l_extendedprice and multiplying by the applied tax for each product.
- **Purchase Amount**: represents the field l_quantity from the lineitem table, no additional calculations were performed to extract this ratio.

The ADAPT notation was selected to represent the multidimensional schema because it is not related to any physical model. The complete conceptual multidimensional model containing all dimensions and their hierarchy levels can be visualized in Figure 4.2.

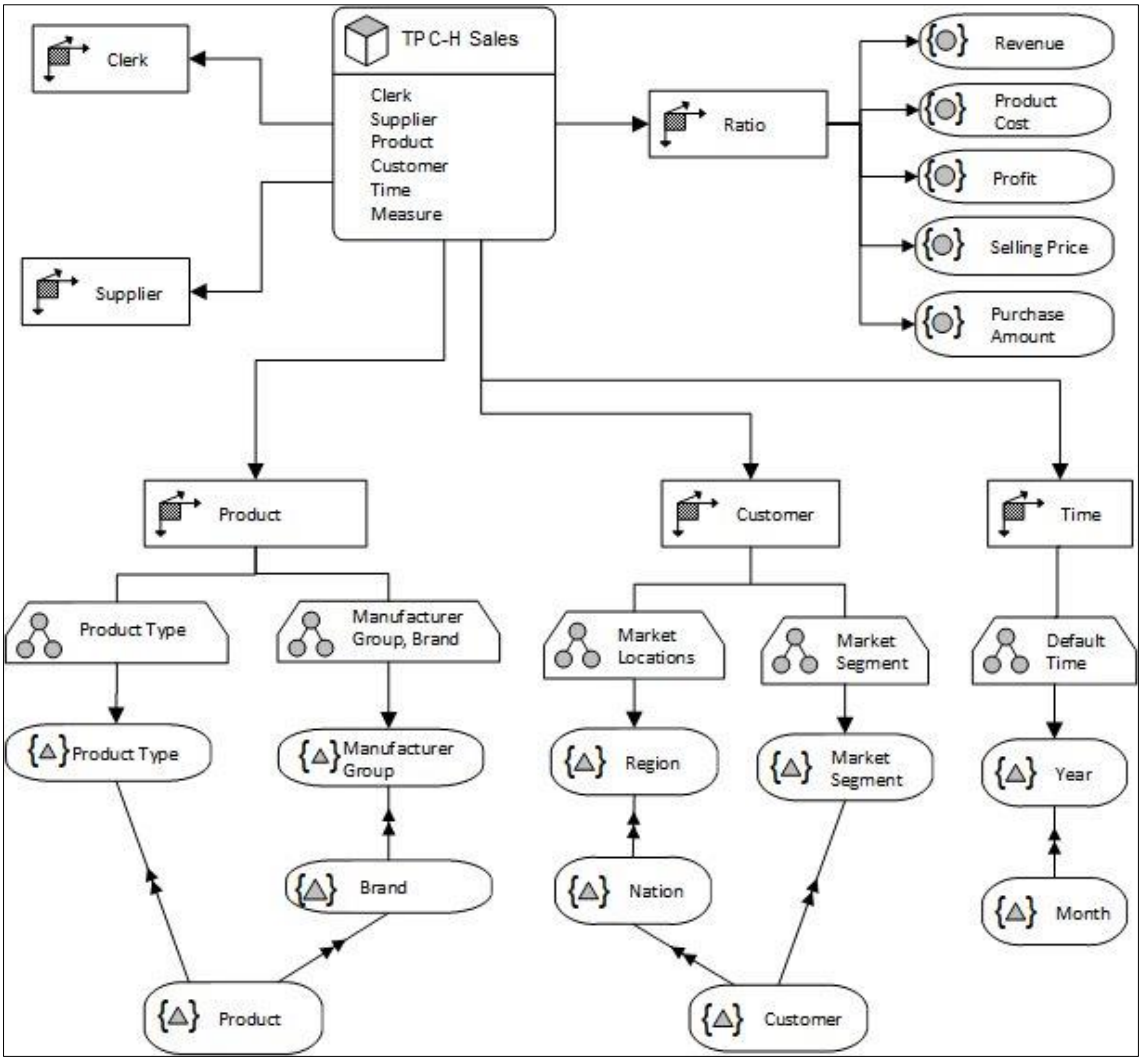


Figure 4.2 TPC-H sales conceptual multidimensional schema

After the Four-Step Dimensional Design Process has been completed, the physical DBs models had to be designed and implemented to receive output data from the ETL

processes. Malinowski and Zimányi (2009) propose that before the physical design, a logical design should be first done. As only one RDBMS technology will be used, the physical model was directly derived from the multidimensional model.

4.3 Data Warehouse Physical Schemas

On star schema, TPC-H original tables were first merged and then denormalized. An exception was the dim_clerk table, where unique clerk values had to be extracted from the TPC-H orders table to be treaded as a dimension table. The columns version, date_from, and date_to in the dim_supplier, dim_clerk, dim_product, and dim_customer are used to control updates on dimensional data. If a product is reclassified or a new market segment is inserted, these columns will store the historical changes on the DW tables. Figure 4.3 shows the physical implementation of the star schema.

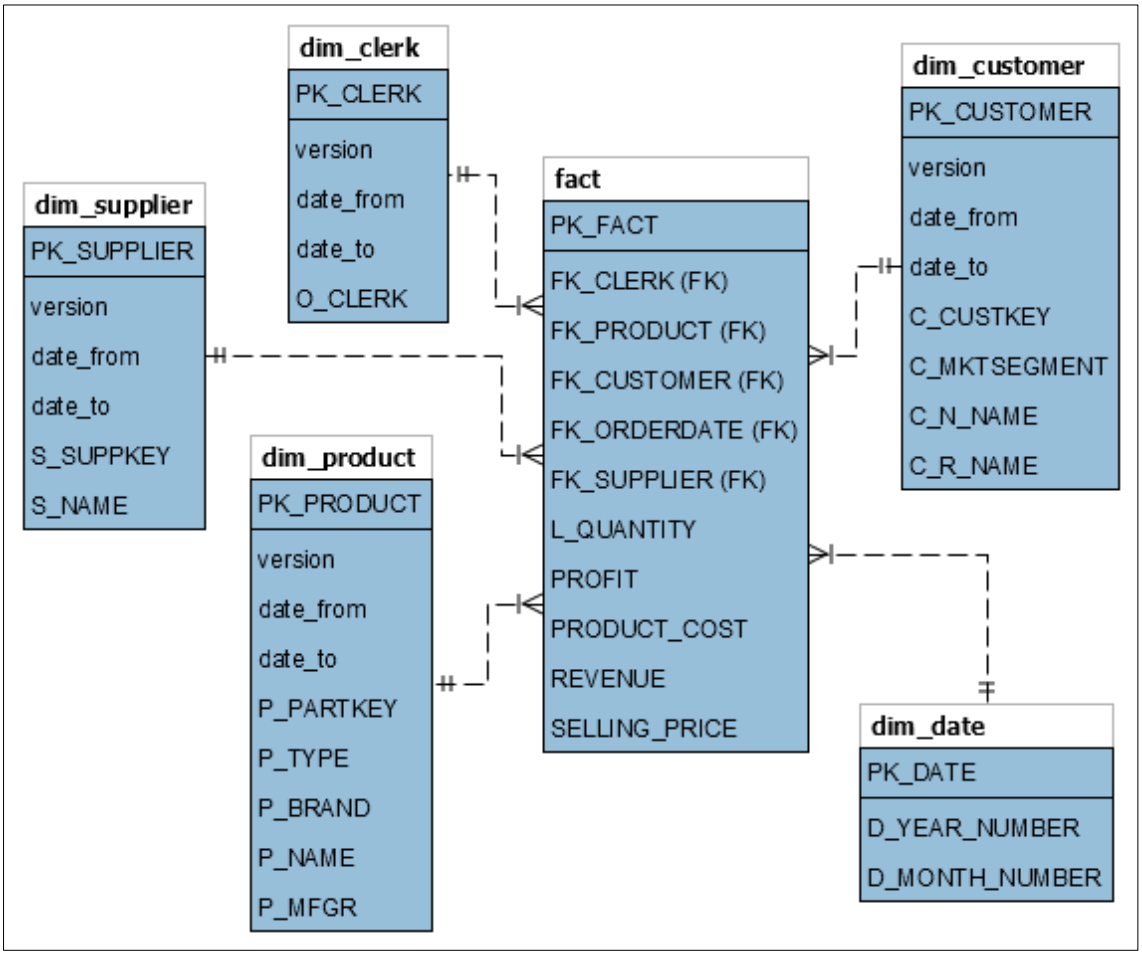


Figure 4.3 Star schema physical model applied to TPC-H

The dimensional customer data will be stored on the dim_customer table. The customer's nation and region are represented by the columns C_N_NAME and C_R_NAME, respectively. The column C_MKTSEGMENT will be used to store the market segment

related to each customer. The PK_CUSTOMER is the primary key of the dim_customer table. Column C_CUSTKEY stores the original customer key from the TPC-H database, and it is used to update customer information.

In the dim_clerk table, PK_CLERK represents the primary key, and O_CLERK stores the clerk's name. The dim_supplier table has the PK_SUPPLIER column that corresponds to the PK; the S_SUPPKEY column stores the PK from the TPC-H database; the S_NAME represents the supplier's name. The supplier's region and nation columns were not created because the multidimensional model did not consider those levels.

The PK on the dim_product table is represented by the column PK_PRODUCT. The column P_PARTKEY stores the original key value from the TPC-H database. P_TYPE column represents the product type; P_BRAND the product brand; P_NAME the product's name and P_MFGR the manufacturer group. The dim_date table stores the time dimension information, where the PK_DATE column is the table's PK; D_YEAR_NUMBER corresponds to the year; D_MONTH_NUMBER the month.

The fact table is positioned in the center of a star schema. Each FK represents one atomic level related to a dimension. The FK_CUSTOMER has its corresponding value on the dim_customer as PK_CUSTOMER. The same applies to the other FKs: FK_CLERK, FK_PRODUCT, FK_ORDERDATE, FK_SUPPLIER. Each ratio has its corresponding column: PROFIT, PRODUCT_COST, REVENUE, SELLING_PRICE, and L_QUANTITY represents the purchase amount ratio. The PK_FACT column was added to support ETL batch tasks.

According to the grain definition step, a single row in the fact table must represent one atomic level and its ratios. The designed fact table confirms this definition: a single row represents a combination of Clerk, Product, Customer, Month, and Supplier. All dimensions and hierarchies defined in the multidimensional modeling step are also present in the implemented star schema.

Different from a star schema, in the snowflake, all data related to the dimension levels had to be normalized. Tables nation and region were only copied from the TPC-H data source because they are already normalized. As tables dim_supplier and dim_clerk have only a single hierarchy level, they do not need to be normalized. The dim_date table has one hierarchy level (year-month), but for the sake of simplicity, the same structure of star was used. The fact table also shares the same structure of the star schema, that is why dim_supplier, dim_clerk, dim_date, and fact tables were copied from the star schema. The physical snowflake schema can be visualized in Figure 4.4.

The dim_product had to be normalized. The table product_type stores unique product types identified by the column PK_PRODUCT_TYPE. The column PRODUCT_TYPE represents the description of a product type. In the dim_product, the FK_TYPE was added as a reference to the product_type table. The same process was performed with product brands and manufacturer groups.

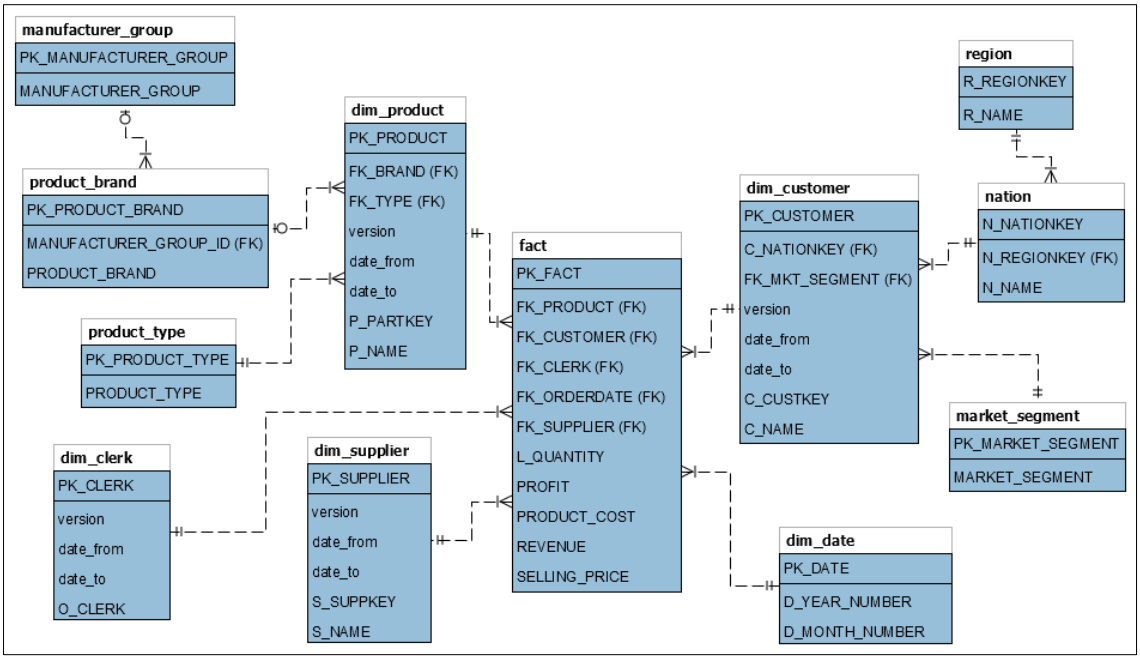


Figure 4.4 Snowflake schema physical model applied to TPC-H

The table dim_customer was also normalized. Unique market segments are stored in the market_segment table. The PK_MARKET_SEGMENT represents the table's PK; the MARKET_SEGMENT column is the description of a market segment. The FK_MKT_SEGMENT was added to the dim_customer to represent the relation. The tables region and nation table were only copied from TPC-H. The C_NATIONKEY is the FK, which corresponds to the N_NATIONKEY on the nation table. The N_REGIONKEY is the FK for R_REGIONKEY in the region table.

The GDWH physical model was implemented based on the GDWH ER model previously introduced. Different from star and snowflake, no direct relation exists between the multidimensional modeling and the physical implementation of the tables. This is explained by the high level of abstraction present on GDWH. Dimension levels are represented as rows in the dimension table. The PK of hierarchy and combination tables are composed of two FKs from their respective tables.

The fact table is the relation between reference_object and ratio tables. For each reference_object_id and ratio_id, there is a fact value. In GDWH, ratios are stored by rows. Different from star and snowflake, where they are stored by columns. That means

if a star or snowflake fact table has a thousand rows and five ratios, the same fact table on GDWH will have five thousand rows. The GDWH physical model used in this study is shown in Figure 4.5.

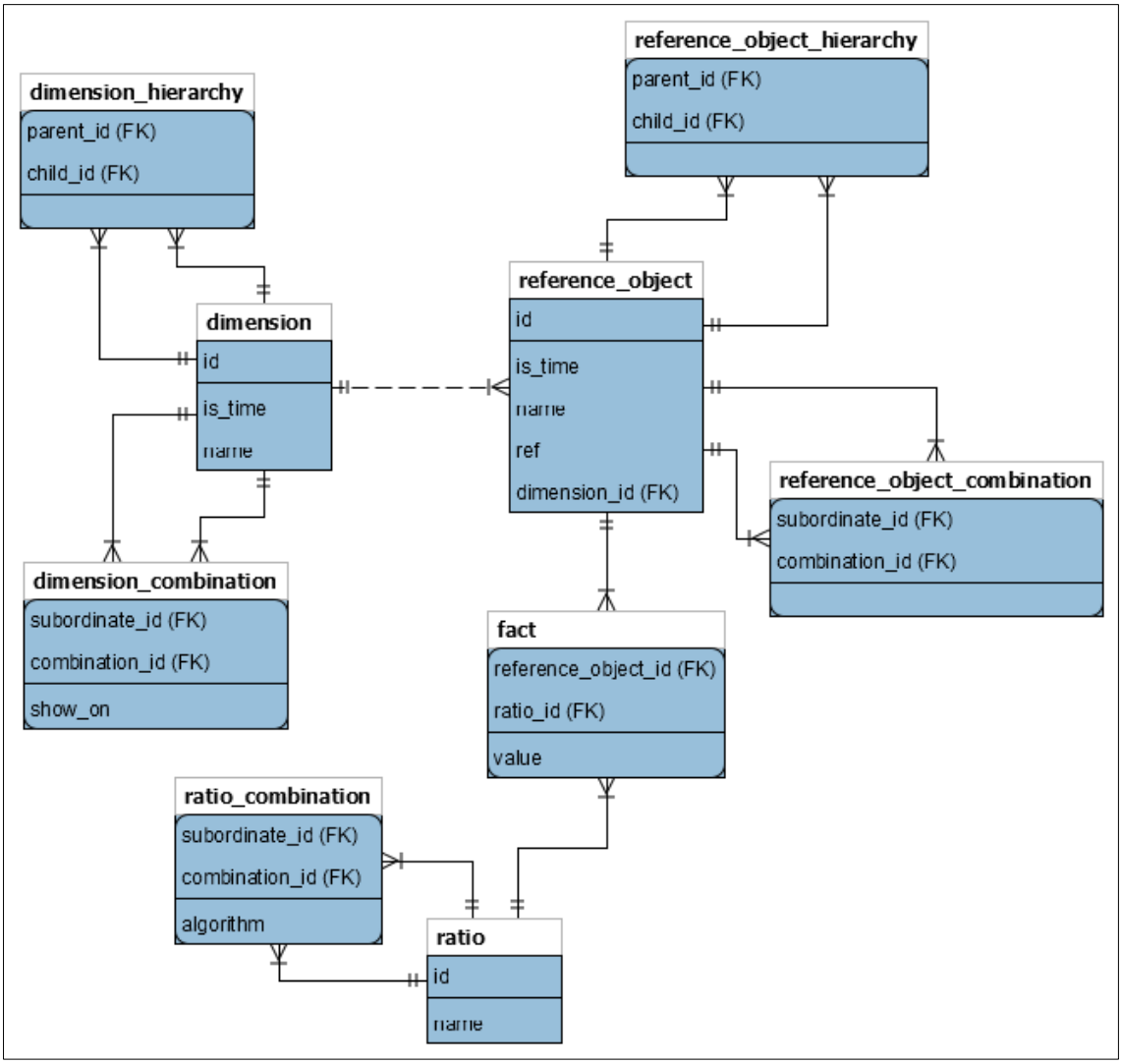


Figure 4.5 GDWH physical model

A DB physical model was created for each scale factor of 1 and 10. The same structure was used for both scale factors because the TPC-H data structure does not vary with different DB sizes. The same GDWH physical model was used to the three variations, although in the ACB, tables reference object hierarchy and reference object combination are not used on the queries. More details will be given in the discussion section. The next chapter presents the IT architecture used to generate and store data in the physical DBs.

5 IT Infrastructure

5.1 System Architecture and Tools

In this section, all software and tools used to generate, store, and query data in the DW architecture will be introduced. The starting point was the data source TPC-H generation using HammerDB. Two databases were created with SF1 and SF10. The process ran on Windows 10 and MySQL 8.0.16 was the chosen database technology to store data. After the data source creation, Pentaho Data Integration was used to run ETL tasks on star, snowflake, and GDWH. A Java web application was developed to query and store the querying running time of the three DW designs.

The set of software and tools used to achieve this project's goals is shown in Figure 5.1. The databases and the Java web application are hosted on a Linux server located on the Westfälische Wilhelms-Universität Münster (WWU) datacenter and can be accessed on http://dwh-architectures.uni-muenster.de/

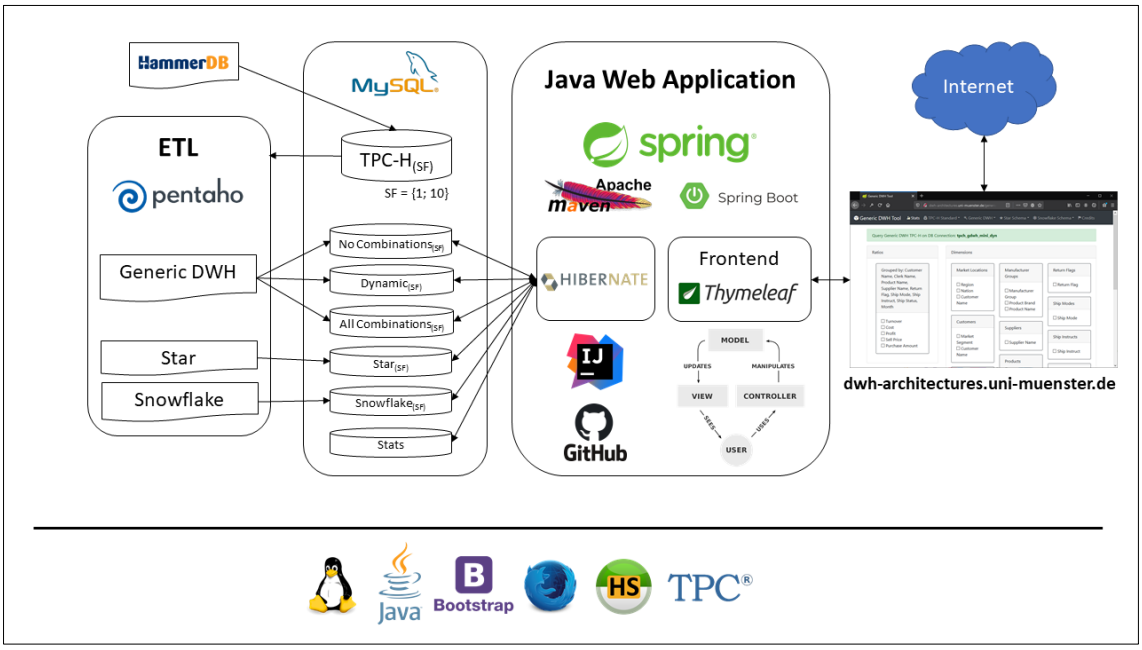


Figure 5.1 System architecture

The TPC-H toolkit comes with a script to generate data that offers the flexibility to set up a scale factor (SF). HammerDB[7] implements the TPC-H script in an automated way. That is why it was used to generate initial databases with different sizes. HammerDB has a user interface to set up the variables, configure database connection, and set the number

[7] http://hammerdb.com

of virtual users to load data. It is possible to choose between MyISAM or InnoDB as storage engines. In this study, only InnoDB was used. (see Appendix B).

The available scale factors on HammerDB to the TPC-H benchmark are 1, 10, 30, 100, and 1000. The approximated number of lines generated for each table is shown at the top of each table in Figure 4.1 as the result of the multiplication by the scale factor. The table part, for instance, has approximately 200,000 lines for SF 1; for SF 10, the table will have around 2,000,000 lines. Tables region and nation do not depend on scaling factors because they are size fixed for all of them. The complete field description and how they were generated can be found on the TPC-H specification document.

All the ETL processes were executed on the developer's machine because running Pentaho on the Linux server over the Internet would decrease the performance time considerably. After the execution of the ETL processes, the DW databases were sent to the university server through the Internet. The complete list of the software with versions used in this study can be visualized in Appendix C.

5.2 Data Warehouse Designs-Query Application

An application named Data Warehouse Designs-Query (DWD-Q) was developed as a query builder to the DW designs selected in this study. The application shows the available dimensions and ratios to be queried. After a combination of dimension and ratio is selected, it generates the corresponding SQL query. For each executed query, the query running time is stored. DWD-Q was developed with IntelliJ IDEA, and GitHub was used as the code repository.

Choosing a programming language to start a project is complex. As the main focus of this study was not testing programming language performances per se, Java was considered suitable to achieve the project's goal due to its robustness, stability, flexibility, scalability. Java is the most popular object-oriented language with a growing community of developers and it is in the first place from TIOBE Index Ranking December 2019. (Nanz and Furia 2015, Tiobe 2019).

The application requirements were set up according to the performance measurement goals. The most important aspect was to provide a friendly web interface where users can query different databases transparently. Then, for each executed query, the query time should be displayed and stored for further analysis. Another requirement was to have the queries being shown on the query result page. This is important for checking if the query string building for the selected ratios and dimensions was correctly done, and, second, to allow the comparison between the three DW designs. With the SQL queries showed on

the result page, it is possible to check how many joins were needed to read the requested information.

The monolithic application was designed as a client-server web architecture to keep portability and compatibility requirements, according to Gray (1993). The application runs over HTTP and can be accessed with a web browser without any need for authentication or authorization. Since the data source was randomly generated and did not represent real data from any real company, no security requirements or Access-Control List (ACL) were needed to be set. Direct database access is granted only for authenticated admin users.

Model-View-Controller (MVC) was the software design pattern used. The central aspect of this pattern refers to splitting the data manipulation layer from the user interface. In this logic, the user has access to a view where data are shown and manipulated. Once the user performs a request (e.g., insert new order), the view will pass this request to the controller layer. The controller processes the user request and calls the model layer to perform data manipulation itself (e.g., connect to DB and insert the new order). One advantage of this pattern is the independence of the view layer, which can be changed entirely without changing the entire model and view layers. (Leff and Rayfield 2001)

Since Spring Framework has full support to work with the MVC pattern, it was selected to build the application. Spring Framework is an open-source application framework to develop Java applications. Servlets are the core elements responsible for handling requests, exceptions, and view resolutions. The web-application was split into two modules: data and web. The Java classes corresponding to the model layer, repositories, and services were organized inside the data module directory. On the web module, the controllers and views were implemented (see Appendices D, E, F).

The feature branching methodology was used to develop the DWD-Q solution. For instance, star, snowflake, and GDWH query had their own feature branches. This strategy is important to keep track of the impacts caused by adding new features. If a merging operation with the develop branch goes wrong, it is possible to undo changes by tracking all commits since the beginning of the development.

Hibernate was used to manage data access to different databases. The flexibility of building queries was mandatory to use this technology. With Hibernate and Spring Data JPA, it is possible to define queries in many ways: from writing standard queries directly on repository methods to building customized strings and executing them as native SQL. Hibernate automatically does the abstraction and query translation. When complex queries had to be written, only native SQL was used to avoid translation errors or

performance issues. But for ordinary queries (e.g., find dimension by name), Spring Data JPA Repository was used.

Figure 5.2 shows the four ways (A, B, C, D) of accessing data with Hibernate. In the (A) variation, the query was defined directly on the method declaration in the DimensionRepository class. This a shortcut to handle data if the query does not get too complicated. In the (B) variation, the same "findByName" query was written using Java Persistence Query Language (JPQL). This option was not used very often in this project, because the combinations offered by Spring Data JPA are quite helpful and enough to define simple queries. In the next variation (C), a native query was defined above the method declaration. In this case, the query finds all dimension roots to build the dimensions hierarchies shown on the query page. Here, it would be hard to rewrite it using JPQL or Spring Data JPA. In the last and most complex variation (D), the query first built with a string concatenation algorithm, then passed to the NativeQuery repository to be executed on the database. This query returns a generic array without any reference to objects. This strategy was important because the dimensions of the array of the resulting values for each query are not fixed.

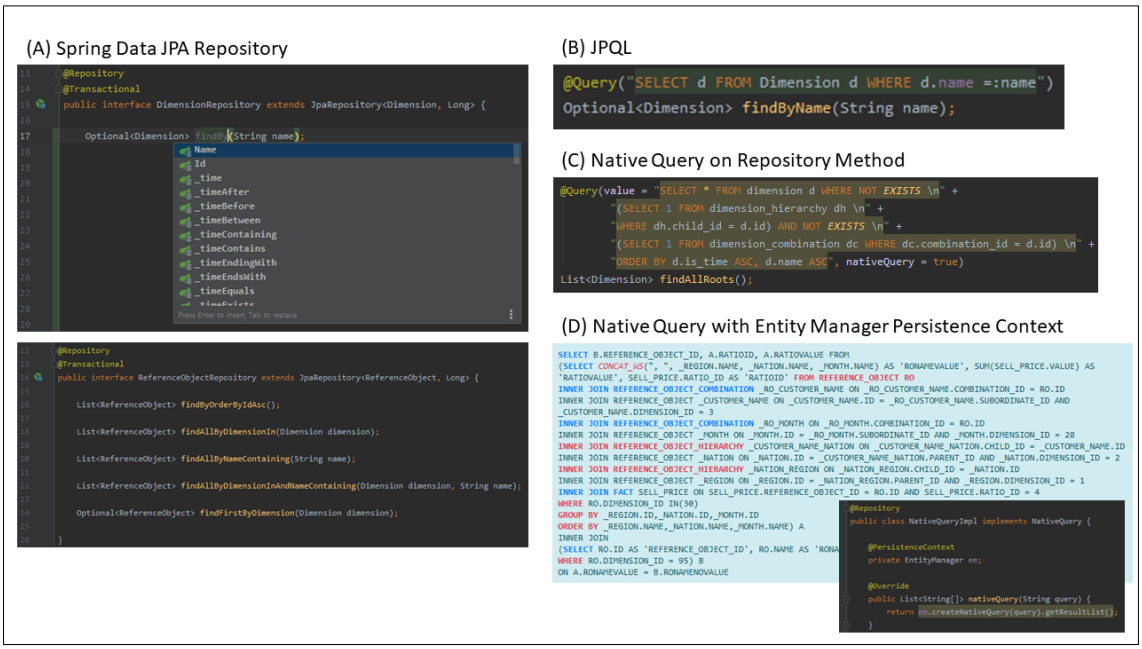


Figure 5.2 Query variations using Hibernate and Spring Data JPA

The flexibility of the SQL query definition was important on the query build for all models. For GDWH dynamic querying, the application first checks if facts are already stored for each requested dimension and ratio. If facts are found, nothing needs to be inserted, and the exiting facts are returned using a select query. However, if facts were not found, the algorithm first checks if the number of dimensions selected is one. If it is

one, a join of fact and reference object table is done where dimension id equals the requested single dimension. If the result set of this query is not empty, it means that the selected single dimension has already inserted facts, and it can be returned. If the result is empty, new facts are inserted.

If the dimension size is greater than one, that means that a new dimension combination needs to be inserted. First, the algorithm checks if the selected dimensions already have an entry on the dimension combination table. If it is the case, facts related to this combination are already stored. If not, it has to be first checked if the selected combinations and selected ratios are valid. The algorithm searches the subordinate dimensions for each combination id (check if the hierarchy is valid). If the selected dimensions and ratios are valid, a new entry on the dimension table is created containing the concatenation of the dimension combination names. With this new dimension id, new entries on the dimension_combination table are created. Subordinate ids are the selected dimensions coming from the user request, and combination id corresponds to the new dimension id. The last step is to insert reference objects and facts values.

To improve performance on the process of checking and inserting new facts, different reference objects can refer to the same value. For instance: the reference objects related to Nation have the same fact value of the dimension combination Region-Nation. The reference object Nation has a different dimension id comparing to Region-Nation. If the user selects only the Nation dimension, the system will query as a single dimension, but if both Region and Nation were selected, the query would be created as a dimension combination. If the dimension combination already exists, the query will return existing fact data; if not, it will create a new dimension combination Region-Nation, new reference objects, and finally, insert new facts. Saving an aggregate reference object with its child is theoretically wrong, according to the GDWH rules. But a workaround was implemented to improve performance. Hence, the algorithm considers those reference objects as different entities.

Figure 5.3 shows the DWD-Q application’s frontend. In this example, the GDWH NCB for SF 1 was selected to be queried. The ratios and dimensions are displayed according to the multidimensional model. The user needs to choose at least one ratio and one dimension to send a request. After clicking on the “Preview” button, the query logic for GDWH NCB will be loaded. After processing the dimensions and ratios combinations, the fact values and the SQL query will be returned to the user. A limit of 1000 was set for all queries to avoid errors in the result exhibition.

The same process applies to query data in the star and snowflake schemas. The system will read and check the selected combinations and build the corresponding query. For star

and snowflake, each dimension level has an initiative that identifies the corresponding dimension table. For instance: customer dimension levels start with “c”, supplier dimension levels start with “s”. For each selected dimension level, a new join is created.

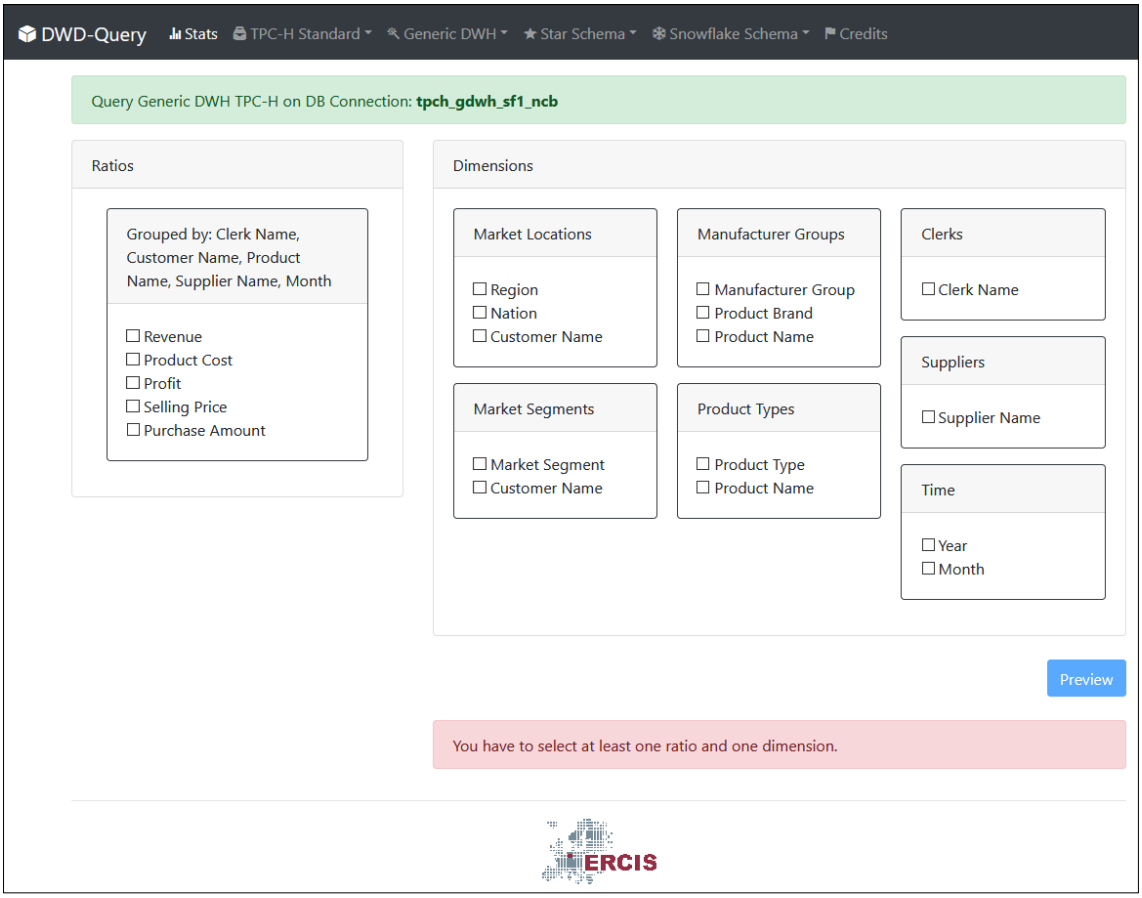


Figure 5.3 DWD-Query Java application

Using the Stream API from Java, ratios are read and mapped to a new string containing the FORMAT() function of MySQL. This procedure was necessary to show the ratio values as a decimal format on the fronted. “GROUP BY ROLLUP” and “ORDER BY” clauses were also added to the end of the query. If the user selects Region, Manufacturer Group, and Month as dimension levels and Revenue and Profit as ratios, the corresponding generated query will be the same as shown in Figure 5.4.

```
SELECT C_R_NAME,P_MFGR,D_MONTH_NUMBER,D_YEAR_NUMBER,
FORMAT(SUM(REVENUE),2),FORMAT(SUM(PROFIT),2) FROM FACT F
INNER JOIN DIM_CUSTOMER C ON C.PK_CUSTOMER = F.FK_CUSTOMER
INNER JOIN DIM_PRODUCT P ON P.PK_PART = F.FK_PART
INNER JOIN DIM_DATE D ON D.DATE_PK = F.FK_ORDERDATE
GROUP BY C_R_NAME,P_MFGR,D_MONTH_NUMBER,D_YEAR_NUMBER WITH ROLLUP
ORDER BY C_R_NAME,P_MFGR,D_MONTH_NUMBER,D_YEAR_NUMBER LIMIT 1000
```

Figure 5.4 Star query example

The same logic of star querying was applied to the snowflake, the difference is the required number of joins for each combination of dimension levels. For each selected dimension level, the system will add the corresponding joins to the query. If the user selects the same combinations of star query example shown in Figure 5.4, the corresponding SQL query is the same as shown in Figure 5.5. The SQL query for snowflake has seven joins, while on the star was executed with five joins.

```
SELECT R_NAME,MANUFACTURER_GROUP,D_MONTH_NUMBER,D_YEAR_NUMBER,
FORMAT(SUM(REVENUE),2),FORMAT(SUM(PROFIT),2) FROM FACT F
INNER JOIN DIM_CUSTOMER C ON C.PK_CUSTOMER = F.FK_CUSTOMER
INNER JOIN NATION N ON N.N_NATIONKEY = C.C_NATIONKEY
INNER JOIN REGION R ON R.R_REGIONKEY = N.N_REGIONKEY
INNER JOIN DIM_PRODUCT P ON P.PK_PART = F.FK_PART
INNER JOIN PRODUCT_BRAND PB ON PB.PK_PRODUCT_BRAND = P.PRODUCT_BRAND_ID
INNER JOIN MANUFACTURER_GROUP MG ON MG.PK_MANUFACTURER_GROUP =
PB.MANUFACTURER_GROUP_ID
INNER JOIN DIM_DATE D ON D.DATE_PK = F.FK_ORDERDATE
GROUP BY R_NAME,MANUFACTURER_GROUP,D_MONTH_NUMBER,D_YEAR_NUMBER WITH ROLLUP
ORDER BY R_NAME,MANUFACTURER_GROUP,D_MONTH_NUMBER,D_YEAR_NUMBER LIMIT 1000
```

Figure 5.5 Snowflake query example

The development of the DWD-Q application was performed in parallel to the ETL design for validation purposes. The next section goes through all ETL phases required to generate data that will be queried on DWD-Q.

6 ETL Design

6.1 ETL on Star Schema

In this phase, a particular ETL was developed for each DW architectural choice (star, snowflake, and GDWH). Pentaho Data Integration was the tool used to create and run the ETL tasks. To execute the jobs and transformations, the input and output DB has to be set. This was done to keep flexibility and run the same ETL to different DB sizes. Output tables can be created on the output task inside Pentaho, but as the physical DBs were already created on previous steps, DDL statements were not executed from Pentaho.

The data source time interval (from 1992 to 1998) was used for two purposes: (i) to split orders and lineitem tables to speed up ETL input data tasks and (ii) to simulate a real-world scenario when dimension lookup/updates[8] and fact loading must be executed when new data from a new time interval need to be inserted. The year was used as a time interval to reduce the ETL runs. Different strategies were used for each DW design to update the corresponding DBs.

The Star ETL main job, shown in Figure 6.1, performs all necessary tasks to denormalize data and load dimensional and fact data. The first step truncates all tables in star DB because this main job was designed to be used as a first ETL run. If there are already stored data, only a Lookup/Update process should be used instead. The job starts with dimensional data loading for Clerks, Customers, Products, and Suppliers. Then, a loop was created to generate staging values for each year, which was used as input to the Facts Loading transformation.

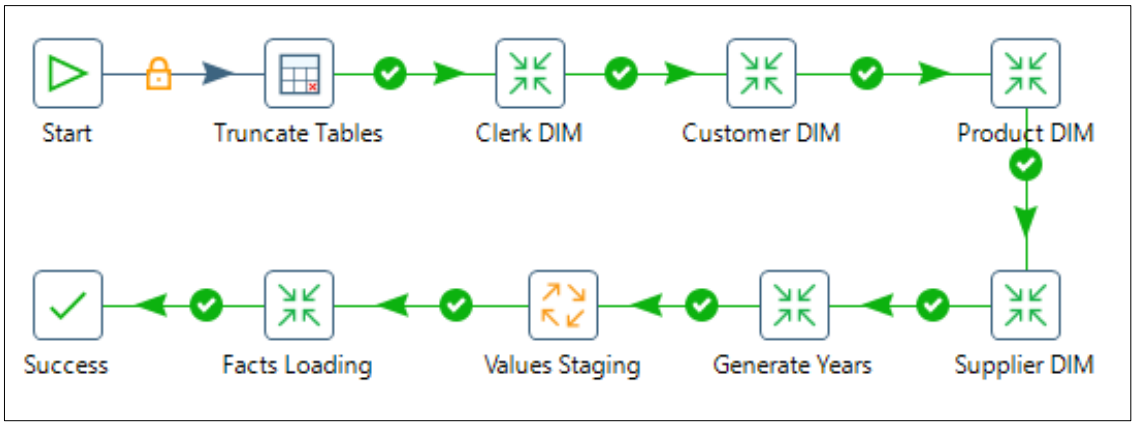


Figure 6.1 ETL on star schema

[8] A Dimension Lookup/Update or Database Lookup are used to compare input data with exiting data on a DW database. Different strategies can be used to do such comparison, but in this research, only the original entity ID was used to perform dimension lookups.

The dimensional data loading transformations were not split by year because the tables were not so big as lineitem and orders. The first transformation is responsible for extracting unique Clerks from the customer table and load data into the dim_clerk table. Group by and order by SQL clauses were used on the input table task to extract and sort unique clerks. After then, a dimension lookup/update was used to insert data into the dim_clerk table.

The next transformation related to dimensions was the Customer DIM. In this step, tables Nation and Region were merged, then merged both Nation name and Region name again on dim_customer table. From the customer table, only the customer name was extracted. Other fields were not used because it was not important to the sales data analysis. This transformation is responsible for executing Dimension Lookup/update using the customer table from TPC-H as input. The output data generated were stored on the dim_customer table.

The Product DIM transformation follows the same logic. First, only required fields from the part table were read. As the required dimension hierarchies fields were already denormalized (product type, manufacturer group, and product brand), the input data was directly forwarded to the dimension lookup/update step and inserted into the dim_product table.

On the Supplier DIM transformation, data were simply extracted from the original supplier table from TPC-H. Only the Supplier Name field was used. Supplier's Nation and Region data were not joined and stored in any DW design, because the current analysis is customer-centric, and Supplier's nationality was considered not important. As the supplier table was not big (SF *10,000), no splitting by year was necessary.

Due to computing power constraints, the Values Staging transformation was split by order year. The Generate Years transformation creates the years as strings to be used as input to Values Staging. Tables orders and lineitem were first split by year, then joined for extracting values for each order year from 1992 to 1998. This step was important to speed up the fact data loading as well since all measures were extracted from the lineitem table. In this process, data from lineitem and orders tables were read, sorted by o_orderkey, merged on this key, and finally inserted into a new table called stg_values.

In the Values Staging transformation, different from dimension lookup/update, only numeric data were extracted and stored to be used as input to the Fact Loading step later on. Here, the part and partsupp tables were merged, then merged again with the stg_values to make possible the supply cost extraction for each lineitem row. The supply cost is important to calculate the cost of each lineitem. The p_retailprice was removed because

the l_extendedprice already contains the p_retailprice value. To calculate the profit, the l_extendedprice field was divided by l_quantity and subtracted from supply cost.

The last step on the Star Schema ETL is the Facts Loading. This transformation can only be started after all dimensions and staging tables were created. The transformation starts reading data from the stg_value table. No splitting by year was necessary because data were partially read and cached, checked against each dimension table, and finally written. For each input row, a database lookup was executed. This task looks at primary keys on dimension tables and creates a corresponding Foreign Key (FK) on the fact table. Figure 6.2 shows the Facts Loading tasks for Star Schema.

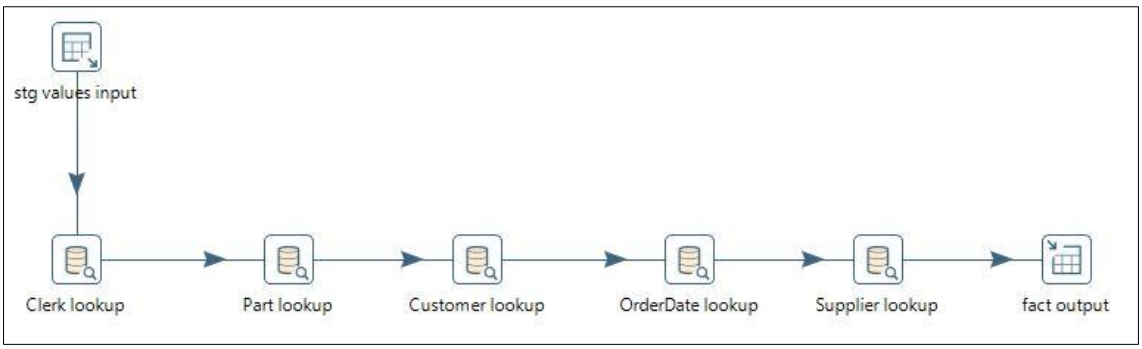


Figure 6.2 Facts loading on star schema

Date Dimension was create using a script including dates from 01-01-1992 to 12-31-1998. This interval was set based on the order date range found on the TPC-H specification. The time interval does not vary in different TPC-H database sizes. Hence, the date dimension table does not vary either and can be generated once and copied later on.

In the end, all numeric values in the fact table will have the corresponding foreign keys to their dimensions, which correctly follows the star schema definition. The entire ETL procedure was repeated for each database size, and the execution time was registered. These execution times will be presented and analyzed in the discussion section.

If new data need to be inserted into the star DB, current dimensions and fact tables should not be truncated. That is why a new job was created where the truncate table step is bypassed. To start this job, an order year must be provided with the input and output DBs. The dimensional data loading steps do not change because they were already configured to update dimensions fields based on the key fields. If this step is executed twice for the same year, no duplicates will be stored.

The Values Staging transformation always truncates the stg_value table by default because it is expected only new fact values. If, for some reason, data from the same year

is read twice, duplicates will occur on the fact table. There is no year split loop on this job because the year itself is required as input.

6.2 ETL on Snowflake Schema

It was found that for snowflake and GDWH, data have to be normalized before performing a dimension lookup/update. As both DW designs were used in this research, a single staging DB was created for each scale factor and used as input for both snowflake and GDWH. On the staging area, there is no dimension or fact data loading yet. The goal is to normalize data based on the defined hierarchy levels. An overview of this process is shown in Figure 6.3.

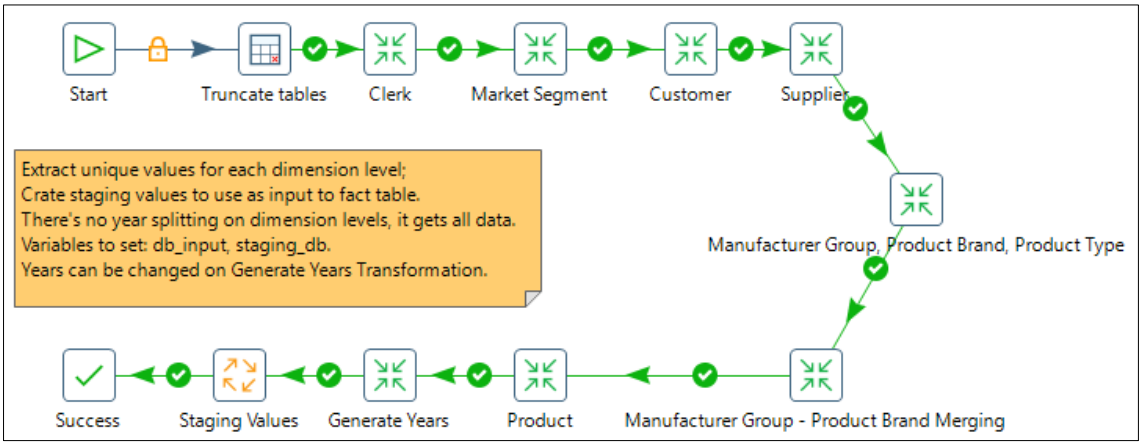


Figure 6.3 Staging ETL for snowflake and GDWH

The first transformation after truncating tables extracts unique clerks from the orders table and saves them into a new table called clerk with a newly generated PK. Next, unique market segments are extracted and saved with a new PK into the market_segment table. In the Customer transformation, data from the customer table from TPC-H is loaded and merged with the market_segment table to create a new 1:n relation between market segment and customer; tables region and nation are also copied from TPC-H to Staging DB in this step. The Supplier transformation extracts the supplier key and name from TPC-H and saves them into the supplier table in the Staging DB. As the supplier table has a scale factor of 10,000, considered small comparing to the scale factor of 6,000,000 on the lineitem table, no splitting was necessary.

The next transformation extracts unique manufacturer groups, product brands, and product types from the part table and stores each one in a new table. The 1:n relation between manufacturer group and product brand is created in the next step. In the Product transformation, data are read from the part table and merged twice with the normalized tables product brand and product type. FKs are added to the product table to identify the

product brand and type for each product. This step completes the normalization of dimensional data.

Next on, data from orders and lineitem table are split by year due to computing power limitations. The required fields to calculate fact values were merged. The stg_values table is the final output that will be used later on to populate the fact table on snowflake and GDWH. Date dimension was also created on the staging DB based on the order year time interval – from 1992 to 1998. A script was used to generate all dates.

To start the snowflake ETL, is mandatory that data were first preprocessed and saved on the Staging Area. The staging_db variable is used as input to the main job, and the snow_db will be the place where output data will be stored. It is also possible to set truncate_tables as true or false. If true is set, all tables on snowflake DB will be truncated; if false is set, dimensional and fact data will be updated. There was no need for splitting transformations by year because the Facts Loading transformation reads pieces of data and uses caching to lookup dimensions, which avoids memory overflow. An overview of all transformations can be visualized in Figure 6.4.

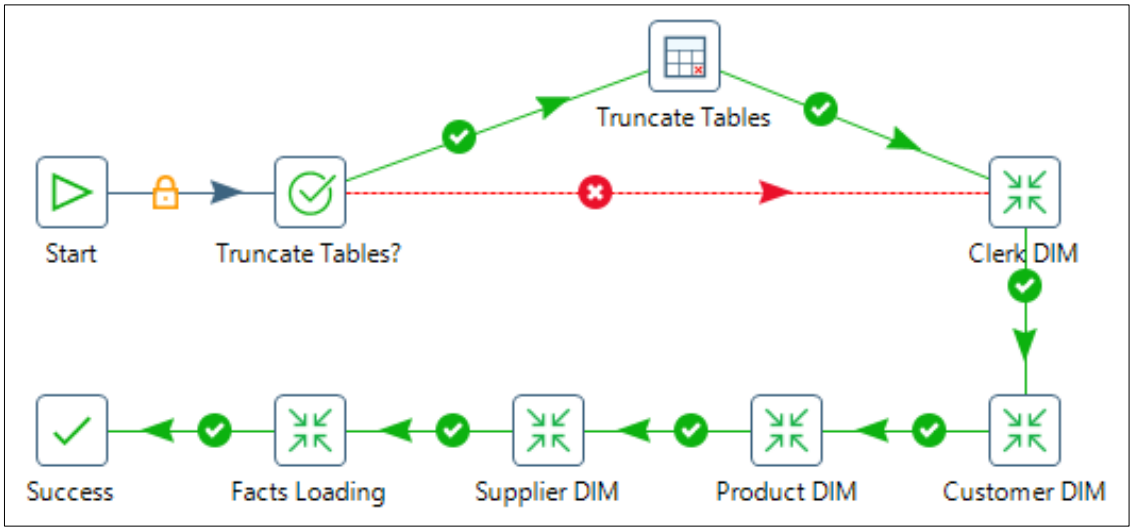


Figure 6.4 ETL on snowflake schema

After the truncate table evaluation, all dimensional data will be loaded. The same dimension lookup/update logic was applied to Clerk, Customer, Product, and Supplier dimensions. For each one of them, data from staging were read and compared with existing data on snowflake. If tables are empty, all data from staging are stored.

The last transformation loads fact data. This step has the same structure as the fact transformation on star schema. Data are loaded from stg_values table. Next, a database lookup was executed for each dimension, and the corresponding numeric values and foreign keys were stored into the fact table. These steps were shown in Figure 6.2.

To update the snowflake with new data, a job called “staging differential” needs to be executed with an order year as input. This job will read only the atomic levels needed for the selected year. That means, only customers, clerks, suppliers, products and respective fact values from a selected order year will be stored on the Staging Area to be later on loaded in the snowflake DB. The same snowflake ETL showed in Figure 6.4 can be used to update data. The only difference is that the truncate_table variable needs to be set as false.

6.3 ETL on GDWH

The GDWH ETL can be started only after the complete execution of the Staging ETL. To work properly, all dimensions levels and their hierarchies should be correctly defined on the corresponding metadata transformations. To extract and build the hierarchy Region – Nation, for example, respective parent and child tables must be joined based on keys and field names definition. The metadata was defined based on the Staging tables and field names. A valid atomic level must be present on each dimension. Customer is the atomic level in the Region – Nation hierarchy. This ETL describes the no combination (NCB) variation of GDWH ETLs. The other two variations, dynamic ETL and all combinations, will be introduced later on. The main GDWH job is shown in Figure 6.5.

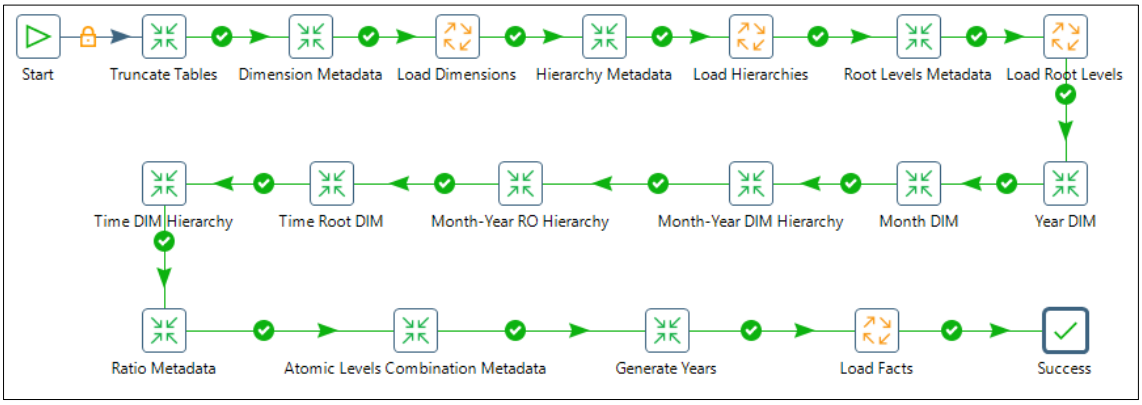


Figure 6.5 ETL on GDWH

The first transformation truncates tables on the selected input GDWH DB. Next, Dimension Metadata generates the dimension names and their respective tables and columns where data are read from Staging DB. The job Load Dimensions runs for each received input row a generic dimension transformation, where first a new dimension is saved on the dimension table and next, the respective reference objects inserted on the reference_object table. The order of saving dimensions matters because group by and order by fields will be sorted based on the dimension ID.

The next transformation called Hierarchy Metadata holds the information about hierarchy levels, e.g., Region (parent) – Nation (children), and their respective tables and fields on Staging DB. The Load Hierarchies job runs, for each input row coming from the previous step, a generic dimension and reference object hierarchy transformation. In these transformations, dimension and reference objects IDs are joined, and the parent-child relation stored on dimension_hierarchy and reference_object_hierachy tables, respectively.

The Root Levels Metadata generates data about the root levels. For Region – Nation – Customer, the root level is Market Locations. The Load Root Levels insert parent and children Ids into the dimension_hierarchy and referece_object_hierarchy tables. For Market Locations parent, all reference object Ids related to Region are stored as child, while the parent ID is the Market Locations reference object ID.

The next two transformations create the years and months dimensions and insert the corresponding reference objects. Dates were read from the dim_date table also created on the Staging ETL. The next two steps create the Month-Year dimension and reference object hierarchies. The “Time Root DIM” saves the “All Time” entry into the reference object table, while the Tim DIM Hierarchy saves the Time entry as parent and Year as child into the dimension_hierarchy table.

Ratio Metadata transformation inserts into the ratio table the ratio names, the ID is automatically generated. The Atomic Level Combination Metadata first saves a new dimension record containing all atomic level’s name separated by a comma, then, creates a new entry on the dimension_combination table with the atomic level IDs as subordinates and the just created entry on dimension table, as combination ID.

The last steps on this ETL refer to the fact data loading. The Load Facts transformation was split by year due to computing power limitations. All reference objects which correspond to an atomic level are loaded and merged by their primary keys, and the names concatenated to create a new reference object, for instance: “Customer#000000719, Clerk#000000154, goldenrod lace spring peru powder, Supplier#000000077, 1992-01”. This reference object has only atomic levels: Customer Name, Clerk Name, Product Name, Supplier Name, and Month. This ETL creates only one dimension combination with all atomic levels, which will be used to load facts. To query the total of profit by Region, all the corresponding atomic levels Customer need to be joined with the reference_object_hierarchies Nation and Region and profits summed by Region.

As the hierarchy levels above atomic levels do not change even with different scale factors, only atomic levels were considered on the lookup/update of GDWH DBs. In this

process, a year must be set as input to load only new facts that need to be stored. A lookup runs on the current DB to identify those reference objects that need to be created, if not found, or need to be updated if something changes. The ref column on the reference_object_table was used to store the original PKs from the atomic levels on the data source. With the PK, the lookup matches the current data with the new data and performs the necessary updates.

ETL variations on GDWH: no combinations (NCB), dynamic ETL (DYN) and all combinations (ACB)

The "no combination" (NCB) variation is the starting point for the other two variations. The NCB ETL consists of saving one single dimension combination entry containing all atomic levels. This dimension combination PK is represented as FK on the reference_object table. Only the reference objects related to the atomic level will be inserted into the fact table.

Figure 6.6 shows an example of GDWH after ETL on TPC-H data. The relation dimension contains the unique Ids and dimension names. Only two atomic levels, Clerk Name, and Customer Name, were loaded. The relation dimension combination stores the combinations of the relation dimension. The relation dimension hierarchy represents the dimension hierarchies present on the relation dimension. In this example, the parent dimension region has children dimensions nations. The ratios revenue and purchase amount are stored in the relation ratio. No ratio combination was needed for this example.

A reference object has only one dimension. Europe (R2) represents the dimension Region (D2), while Germany (R3) represents a Nation (D3). The relation reference object combination R6 represents the combination of atomic levels. All aggregations can be calculated using only the atomic level combination. The relation reference object hierarchy has one line that represents the parent Europe (R2) and child Germany (R3). This relation is used in NCB queries to aggregate data.

The "dynamic ETL" (DYN) variation starts from the same point of the "no combination". However, new combinations are created on the fly based on the selected dimensions and ratios done by the user. Figure 6.6 shows what happens if a user selects the total revenue and total purchase amount for Europe for market segment automobile. First, the system will check if this combination exists; if not, a new combination Region, Market Segment will be inserted into the relation dimension table (D7), and the new corresponding relations D7-D2 and D7-D4 stored on the relation dimension combination table. Then, a reference object containing the instances of Europe and automobile will be queried and

inserted into the relation reference object table (R7). Lastly, the relation fact is updated with the sum of revenue (R1-R7) and the total purchase amount (R2-R7).

Relation Reference Object		
RO-ID	RO-Name	#D-ID
R1	Clerk 01	D1
R2	EUROPE	D2
R3	GERMANY	D3
R4	AUTOMOBILE	D4
R5	Customer 01	D5
R6	Clerk 01, Customer 01	D6
R7	EUROPE, AUTOMOBILE	D7

Relation Dimension	
D-ID	D-Name
D1	Clerk Name
D2	Region
D3	Nation
D4	Market Segment
D5	Customer Name
D6	Clerk Name, Customer Name
D7	Region, Market Segment

Relation Fact		
#R-ID	#RO-ID	Fact Value
R1	R2	371,199,645.72
R1	R3	74,598,484.11
R1	R7	73,967,026.52
R2	R7	55,398
R1	R4	406,079,267.93
R2	R4	305,943

Relation Ref. Obj. comb.	
#CO-RO-ID	#S-RO-ID
R6	R1
R6	R5
R7	R2
R7	R4

Relation Dimension Combination	
#CO-D-ID	#S-D-ID
D6	D1
D6	D5
D7	D2
D7	D4

Relation Ratio	
R-ID	Ratio-Name
R1	Revenue
R2	Purchase Amount

Relation Ref. Obj. Hierarchy	
#P-RO-ID	#C-RO-ID
R2	R3

Relation Dimension Hierarchy	
#P-D-ID	#C-D-ID
D2	D3
D4	D5

Relation Ratio combination		
#CO-R-ID	#S-R-ID	Algorithm

Figure 6.6 Example of GDWH physical implementation

The third variation, called “all combinations” (ACB), stores the result of all possible combinations of dimension levels (excluding duplicates, i.e., region-year is the same as year-region). This variation can be understood as the evolution of the “dynamic ETL” because after querying all combinations, in the end, the DB will contain all combinations stored. There is no need to store reference object combinations or reference objects hierarchies anymore because there will be a fact-value for each new reference object saved. Only the relation dimension combination grows accordingly.

The idea behind the implementation of three different ETLs is a workaround to improve performance. Because each GDWH ETL variation requires a different SQL query definition to read data, the query running time will be different. The ETL times and DB sizes after each GDWH ETL will be presented in the performance measurement section and analyzed in the discussion section.

7 Business Questions

Business questions are derived from the business process sales (already specified on the dimensional design phase) and defined based on a real-world case for a wholesale supplier. A sale process starts with a customer ordering items. Clerks are crucial to the sale process because they are responsible for adding products to orders. Every item added to an order generates quantitative data (purchase amount, product cost, selling price, tax, discount).

TPC-H represents activities of a generic wholesale supplier that needs to manage the selling and distribution processes of products worldwide. Customers from different geographic regions and nations can order products. A clerk is responsible for creating an order with one or more products that will be delivered to a customer. Suppliers are situated in different geographic locations and deliver products that are stored in the retail facilities.

Identify customer behavior is an objective of sales performance analysis. Business questions were defined to analyze customer purchases related to different aspects. For instance, geographical locations were used to group customers and product purchases. With customer and product relation analysis, it is possible to set goals to improve business revenue. Logistics can also be improved if geographical locations influence sales performance.

Clerk sales performance supports managers in defining strategies to improve business profits in two ways. The first way is to analyze the relation between clerks and suppliers. If there is a pattern between a clerk and a supplier, it can be considered in future orders to reduce product costs by buying more products. The second way is the relation between clerks and customers. In this case, there is the possibility to offer better promotions or a customized service to a group of customers.

Time is a pivot dimension in TPC-H sales performance analysis. A strategic decision can be based on revenue variation by year or by month. Identifying which products were sold in which month is helpful to manage stock. A market segment analysis by year can be used to support customer behavior analysis and set marketing strategies. Every dimension can be analyzed from a historical perspective if combined with a time hierarchy level.

Based on these characteristics of the sales process from TPC-H, ten Business Questions (BQ) were defined and are shown in Table 7.1. As the aim of a DW is to provide aggregated data to answer strategic business questions, the ratios used in the business questions always represent the sum of the corresponding fact values. Atomic levels are

not the main focus of business questions because the result of queries with atomic levels can easily overflow the computing capacity in big databases. Hence, more attention was given to grouping levels, e.g., Region, Manufacturer Group, Product Type.

BQ	Business Questions
BQ1	What is the total revenue, product cost, profit, and selling price in EUROPE in 1996?
BQ2	What is the total profit and purchase amount for the market segment AUTOMOBILE in December 1995?
BQ3	What is the total product cost, profit, and selling price in BRAZIL to product type SMALL PLATED TIN in 1997?
BQ4	How much revenue and profit the Clerk#000000015 generated to the manufacturer groups Manufacturer#2 and Manufacturer#5 in 1993.
BQ5	What is the total revenue generated by each region for the manufacturer group Manufacturer#4 in November 1992?
BQ6	What is the total product cost, selling price, and purchase amount generated from the market segment MACHINERY with all product types for Supplier#000000093 in 1995, 1996 and 1997?
BQ7	Which customers have purchased more than 199 units grouped by product brand, month, and year?
BQ8	What is the total revenue, product cost, profit, selling price, and purchase amount generated from Clerk#000000535 in all regions for the market segment FURNITURE?
BQ9	What is the total revenue, profit, and selling price generated in the region AMERICA and nations CANADA and UNITED STATES for Supplier#000000024?
BQ10	What is the total profit and purchase amount of all products with the Brand#41 and negative profit grouped by clerk?

Table 7.1 Business questions

In the definition of business questions, attention was also given to technical issues related to performance measurement: functional coverage (different types of queries), and selectivity coverage (variations in the portion of data being accessed). Table 7.2 confirms that a variety of dimension levels and ratios combination is present. The idea was to cover both business expectations and a wide variety of joins and database filtering operations. (O'Neil et al. 2006).

BQ	Dimension Levels	Ratios (sum)				
		Revenue	Product Cost	Profit	Selling Price	Purchase Amount
BQ1	Region – Year	X	X	X	X	
BQ2	Market Segment – Month			X		X
BQ3	Nation – Product Type – Year		X	X	X	
BQ4	Clerk Name – Manufacturer Group – Year	X		X		
BQ5	Region – Manufacturer Group – Month	X				
BQ6	Market Segment – Product Type – Supplier Name – Year		X		X	X
BQ7	Customer Name – Product Brand – Month – Year					X
BQ8	Clerk Name – Region – Market Segment	X	X	X	X	X
BQ9	Region – Nation – Market Segment – Supplier Name	X		X	X	
BQ10	Clerk – Product Brand – Product Name			X		X

Table 7.2 Dimension levels and ratios used in business questions

8 Performance Measurement

8.1 Query Definition

The corresponding SQL queries were derived from the business questions. The concept of *Query Flight*, proposed by O'Neil et al. (2006), was used in this section. A Query Flight has one or more query variations related to different filtering conditions. The first query variation of a query flight uses the predicates defined in the business question. For instance, BQ1 has two predicates: region = EUROPE and year = 1996. A query variation can be defined as region = AMERICA and year = 1994.

The SQL queries used in the performance measurement are not the same as the queries generated by the DWD-Q application. Functions or clauses like FORMAT, SUBSTRING, and ROLLUP were removed to improve performance, but the query structure and the values returned are the same.

Figure 8.1 shows the corresponding SQL query translation for the BQ1 used for star schema. The first line selects the fields region name, year, and the total revenue from the fact table. Two joins were necessary to retrieve the region from the customer dimension table and the year from the date dimension table. The fourth line filters the results for Europe and 1998.

```
1  SELECT C_R_NAME, D_YEAR_NUMBER, SUM(REVENUE) FROM FACT F
2  INNER JOIN DIM_CUSTOMER C ON C.PK_CUSTOMER = F.FK_CUSTOMER
3  INNER JOIN DIM_DATE D ON D.DATE_PK = F.FK_ORDERDATE
4  WHERE C_R_NAME = "EUROPE" AND D.D_YEAR_NUMBER = 1998;
```

Figure 8.1 Query 1.1 for star schema

The Q1.1 definition for snowflake is shown in Figure 8.2. The first line is the same as the star schema. Four joins were necessary to return the same result because the region information is not in the customer dimension table. The last line filters the result for Europe and 1998.

```
1  SELECT R.R_NAME, D_YEAR_NUMBER, SUM(REVENUE) FROM FACT F
2  INNER JOIN DIM_CUSTOMER C ON C.PK_CUSTOMER = F.FK_CUSTOMER
3  INNER JOIN NATION N ON N.N_NATIONKEY = C.C_NATIONKEY
4  INNER JOIN REGION R ON R.R_REGIONKEY = N.N_REGIONKEY
5  INNER JOIN DIM_DATE D ON D.DATE_PK = F.FK_ORDERDATE
6  WHERE R.R_NAME = "EUROPE" AND D.D_YEAR_NUMBER = 1998;
```

Figure 8.2 Query 1.1 for snowflake

Figure 8.3 shows the Q1.1 executed on GDWH NCB. The atomic level for each selected dimension level needs to be included to read the fact values. Six joins must be added to select the reference object region: two for the atomic level customer (lines 5 and 6), two for the hierarchy level nation (lines 8 and 9), and two for the hierarchy level region (lines 11 and 12). The dimension Id for each hierarchy level is sent from the query request page. The filters Europe and 1998 are added to the corresponding reference objects joins, lines 14 and 18, respectively.

```
1  SELECT _REGION.NAME, _YEAR.NAME, SUM(REVENUE.VALUE) FROM REFERENCE_OBJECT RO
2  INNER JOIN REFERENCE_OBJECT_COMBINATION _RO_MONTH ON _RO_MONTH.COMBINATION_ID = RO.ID
3  INNER JOIN REFERENCE_OBJECT _MONTH ON _MONTH.ID = _RO_MONTH.SUBORDINATE_ID
4      AND _MONTH.DIMENSION_ID = 18
5  INNER JOIN REFERENCE_OBJECT_COMBINATION _RO_CUSTOMER_NAME ON _RO_CUSTOMER_NAME.COMBINATION_ID = RO.ID
6  INNER JOIN REFERENCE_OBJECT _CUSTOMER_NAME ON _CUSTOMER_NAME.ID = _RO_CUSTOMER_NAME.SUBORDINATE_ID
7      AND _CUSTOMER_NAME.DIMENSION_ID = 5
8  INNER JOIN REFERENCE_OBJECT_HIERARCHY _CUSTOMER_NAME_NATION ON _CUSTOMER_NAME_NATION.CHILD_ID = _CUSTOMER_NAME.ID
9  INNER JOIN REFERENCE_OBJECT _NATION ON _NATION.ID = _CUSTOMER_NAME_NATION.PARENT_ID
10     AND _NATION.DIMENSION_ID = 3
11 INNER JOIN REFERENCE_OBJECT_HIERARCHY _NATION_REGION ON _NATION_REGION.CHILD_ID = _NATION.ID
12 INNER JOIN REFERENCE_OBJECT _REGION ON _REGION.ID = _NATION_REGION.PARENT_ID
13     AND _REGION.DIMENSION_ID = 2
14     AND _REGION.NAME = "EUROPE"
15 INNER JOIN REFERENCE_OBJECT_HIERARCHY _MONTH_YEAR ON _MONTH_YEAR.CHILD_ID = _MONTH.ID
16 INNER JOIN REFERENCE_OBJECT _YEAR ON _YEAR.ID = _MONTH_YEAR.PARENT_ID
17     AND _YEAR.DIMENSION_ID = 17
18     AND _YEAR.NAME = "1998"
19 INNER JOIN FACT REVENUE ON REVENUE.REFERENCE_OBJECT_ID = RO.ID AND REVENUE.RATIO_ID = 1
20 WHERE RO.DIMENSION_ID IN(20);
```

Figure 8.3 Query 1.1 for GDWH NCB

Figure 8.4 shows the corresponding SQL queries used on Q1.1 for GDWH DYN. To return the requested values for Q1.1 with GDWH DYN, five queries must be executed in sequence: the first query inserts a new dimension combination “Region, Year” on dimension table; the second query inserts two lines into dimension_combination table where the combination_id is the new dimension created and the subordinate_id is the Region and Year dimension ids; the third query returns the reference objects names and ratio values to be saved, which corresponds to Figure 8.3; a fourth query is executed to insert the reference objects and facts; finally, the fifth query return the result filtered.

The GDWH ACB architecture has all the dimension combinations stored. That means no insert operation needs to be performed to query data because all data were already inserted in the ETL process. No joins with reference object combination and reference object hierarchy tables must be performed anymore. Only one join for each ratio must be added. The Q1.1 definition for GDWH ACB will not be represented because it is the same as the fifth query used on GDWH DYN. It can be visualized in Figure 8.4 from lines 19 to 23.

```
# First query
INSERT INTO DIMENSION (NAME, IS_TIME) VALUES ("REGION, YEAR", FALSE);

# Second query
INSERT INTO DIMENSION_COMBINATION (COMBINATION_ID, SUBORDINATE_ID, SHOW_ON)
    VALUES ( (SELECT ID FROM DIMENSION WHERE NAME = "REGION, YEAR"), (SELECT ID FROM DIMENSION WHERE NAME = "REGION"), FALSE);
INSERT INTO DIMENSION_COMBINATION (COMBINATION_ID, SUBORDINATE_ID, SHOW_ON)
    VALUES ( (SELECT ID FROM DIMENSION WHERE NAME = "REGION, YEAR"), (SELECT ID FROM DIMENSION WHERE NAME = "YEAR"), FALSE);

# Third query is the same used on GDWH NCB

# Fourth query
INSERT INTO REFERENCE_OBJECT (NAME, DIMENSION_ID, IS_TIME)
    VALUES ("EUROPE, 1998", (SELECT ID FROM DIMENSION WHERE NAME = "REGION, YEAR") , FALSE);
INSERT INTO FACT (REFERENCE_OBJECT_ID, RATIO_ID, VALUE)
    VALUES (LAST_INSERT_ID(), 1, 3927873513.900003);

# Fifth query - return saved reference objects and facts
SELECT RO.NAME, REVENUE.VALUE AS 'REVENUE' FROM REFERENCE_OBJECT RO
INNER JOIN FACT REVENUE ON REVENUE.REFERENCE_OBJECT_ID = RO.ID AND REVENUE.RATIO_ID = 1
WHERE RO.DIMENSION_ID = (SELECT ID FROM DIMENSION WHERE NAME = "REGION, YEAR")
    AND RO.NAME LIKE '%EUROPE%'
    AND RO.NAME LIKE '%1998%';
```

Figure 8.4 Query 1.1 on GDWH DYN

The complete list of the sixteen executed queries and their corresponding translations to each DW architecture are available in the DWD-Q application repository[9]. The results of the query running times will be presented in the next section.

8.2 Query Running Times

The queries were executed from a SQL file inside the university's server and the results saved in the same server. Each query was executed three times in different moments to avoid MySQL caching improvements. The MySQL service was restarted every time before setting the DB to run the queries. The Linux system used has the following hardware configuration: Intel® Xeon® Gold 6146 3.20GHz CPU; 16 GB RAM; SCSI HDD virtual disk.

For star, snowflake, and GDWH NCB, no data were inserted in the querying processes, only reading operations were used. The query running time for the GDWH DYN is the sum of all the five queries executed sequentially (read and write operations), according to the query definition. The ACB query corresponds to the last query on DYN variation (only reading operations) because in ACB all reference objects and facts were already written in the ETL process.

The average of each query running time is presented in Table 8.1 (for SF 1) and Table 8.2 (for SF10). The complete result of each query running time for SF 1 is shown in Appendix G, and SF 10 is shown in Appendix H.

[9] https://github.com/pedro911/generic-dwh-app/tree/master/generic-dwh-web/src/main/resources/static/resources/queries

Query	Star	Snowflake	GDWH NCB	GDWH DYN	GDWH ACB	Rows Returned
Q1.1	9.22	43.59	43.77	43.85	0.03	1
Q2.1	8.81	7.08	42.46	42.61	0.01	1
Q2.2	9.40	13.66	52.38	52.83	0.01	9
Q3.1	11.30	7.91	53.38	53.52	0.07	1
Q3.2	16.34	27.34	189.99	196.43	0.05	100
Q3.3	10.35	8.47	16.35	16.93	0.03	9
Q4.1	2.52	14.00	30.68	30.83	0.06	2
Q5.1	9.42	1002.37	44.34	44.59	0.01	5
Q5.2	16.01	2022.33	116.96	127.29	0.01	320
Q6.1	11.74	11.19	60.06	62.69	6.02	42
Q7.1	14.56	253.62	140.16	140.98	344.94	24
Q7.2	14.56	228.83	111.82	113.95	9.07	67
Q8.1	2.83	34.87	25.82	26.36	0.03	5
Q9.1	5.79	7.57	7.77	8.48	1.73	10
Q10.1	8.44	8.18	16.56	53.37	2.04	761
Q10.2	23.94	38.48	49.19	71.47	1.43	451
Sum	**203.69**	**3729.51**	**1001.67**	**1086.18**	**365.54**	**1806**
Avg.	**12.73**	**233.09**	**62.60**	**67.89**	**22.85**	**112.88**

Table 8.1 Query running times (seconds) for scale factor 1

Query	Star	Snowflake	GDWH NCB	GDWH DYN	GDWH ACB	Rows Returned
Q1.1	78.65	447.04	710.37	710.52	0.04	1
Q2.1	75.88	59.39	766.66	766.78	0.02	1
Q2.2	83.45	117.20	783.02	783.46	0.01	9
Q3.1	98.27	96.65	849.03	849.21	0.05	1
Q3.2	139.54	239.89	3,396.00	3,402.54	0.03	100
Q3.3	90.32	72.46	342.32	342.90	0.02	9
Q4.1	79.38	667.49	246.70	246.86	0.22	1
Q5.1	82.35	8,548.14	795.65	795.89	0.01	5
Q5.2	137.23	17,344.92	3,665.61	3,676.05	0.01	320
Q6.1	107.92	95.16	1,881.13	1,883.68	28.82	38
Q7.1	3,067.24	6,521.63	61,257.00	61,257.79	6,164.37	22
Q7.2	171.22	1,961.30	5,857.39	5,858.77	250.67	42
Q8.1	24.57	297.36	56,395.47	56,396.02	1.39	5
Q9.1	57.28	76.06	21,860.81	21,861.52	163.42	10
Q10.1	77.29	75.47	2,594.14	2,893.50	338.08	6145
Q10.2	300.33	358.18	35,612.23	35,886.12	58.06	4749
Sum	**4,670.93**	**36,978.34**	**197,013.53**	**197,611.61**	**7,005.22**	**11,459**
Avg.	**291.93**	**2,311.15**	**12,313.35**	**12,350.73**	**437.83**	**716.19**

Table 8.2 Query running times (seconds) for scale factor 10

8.3 ETL Running Times

The ETL running times for scale factors 1 and 10 were collected from the logging panel on Pentaho and presented in Table 8.3. Pentaho was executed on Windows 10 and the machine has the following configuration: Intel® Core™ i7-7700K CPU @ 4.20GHz; 32GB RAM; SSD local storage. An estimation was calculated to scale factors 100 and 1,000 based on the variation from SF 1 to 10. The staging ETL execution time was added to the total time on GDWH and Snowflake ETL processes. The fact loading for all ETL processes was split by order year (1992 to 1998) to avoid memory overflow.

The GDWH DYN execution time represents the NCB time and the elapsed time to load the dimension combinations derived from the business questions for each order year. The GDWH DYN started with NCB data for the year 1992. After the sixteen queries were executed, the new dimension combinations were collected and the corresponding reference objects and facts were loaded into DYN DB. The process was repeated for each order year (1993 to 1998). The objective was to measure the time to synchronize the new dimension combinations for each new batch of data.

For the GDWH ACB ETL, only the dimension combinations, its reference objects, and facts derived from the business questions were loaded. The total time consists of the NCB query time and insert new reference objects and facts times. There is no difference between dimension and reference objects/fact loading time on GDWH DYN and ACB because there is no dimension data loading.

SF	Star		Snowflake		GDWH NCB		GDWH DYN	GDWH ACB
	Dim.	Facts	Dim.	Facts	Dim.	RO/Facts	RO/Facts	RO/Facts
1	0.17	1.95	0.96	2.11	0.08	1.55	1.82	4.90
10	0.55	5.62	3.03	6.52	0.42	24.60	32.55	87.35
100*	1.78	16.20	9.56	20.15	2.08	390.43	582.14	1,557.15
1,000*	5.76	46.68	30.18	62.26	10.42	6,196.44	10,411.43	27,758.54

* not implemented, only estimated. “Dim.” means dimensions. “RO” means reference objects.

Table 8.3 ETL execution times (hours)

The size of each DB used in the ETL processes is shown in Table 8.4. TPC-H DB corresponds to the synthetic data source generated by HammerDB for SF 1 and 10. The DB sizes for scale factors 100 and 1,000 were estimated based on the variation from SF 1 to SF 10. The GDWH DYN DB sizes were collected after running the sixteen queries (reading and writing operations). The GDWH ACB size is greater than the DYN because on DYN, only filtered reference objects were stored. In ACB, all reference objects and facts derived from dimension combinations were loaded.

SF	TPC-H	Star	Snowflake	GDWH NCB	GDWH DYN	GDWH ACB
1	3.4	0.6	0.6	5.5	5.6	12.9
10	20.4	5.3	5.2	52.8	54.4	96.7
100*	123.1	48.4	46.1	501.7	535.1	726.4
1,000*	744.9	429.6	408.8	4,769.7	5,261.0	5,454.5

* not implemented, only estimated.

Table 8.4 DB sizes in GB

A broad discussion about the data collected in the performance measurement will be carried out in the next section.

9 Discussion

Five evaluation criteria were defined to compare the selected DW architectures from different perspectives. Table 9.1 shows the five evaluation criteria and the corresponding rank position achieved by each DW architecture. Except for metadata management, all other rank positions were derived from the performance measurement results.

Evaluation Criteria	Star	Snowflake	GDWH NCB	GDWH DYN	GDWH ACB
Query Performance	1st	3rd	4th	5th	2nd
DB Size	2nd	1st	3rd	4th	5th
ETL Performance	1st	2nd	3rd	4th	5th
Data Consistency and Redundancy	5th	2nd	1st	3rd	4th
Metadata Management	5th	4th	1st	2nd	3th

Table 9.1 Data Warehouse architectures rank positions for each evaluation criterion

Query performance is the essential criterion to evaluate the performance of each DW architecture. More attention will be given to the number of joins in queries and the DB size because they are highly correlated to the query time. ETL performance, data consistency and redundancy, and metadata management are not directly related to query performance, but also relevant aspects to evaluate a DW architecture.

Query Performance

Star was the best on query performance because it has the smallest total query running time, according to Table 8.1 and Table 8.2. Snowflake was the worst in query performance for SF 1, but the third for SF 10. Snowflake was ranked as third because the classification of each DW architecture is related to the impact of DB size increments. This also justifies the fourth and fifth places for GDWH NCB and DYN, respectively.

The smallest number of joins explains the best performance of GDWH ACB in 15 of 16 queries for SF 1, and in 12 of 16 queries for SF 10. The number of joins in the GDWH ACB query does not depend on the number of hierarchy levels selected by the user. There is always a reference object for each hierarchy level combination. Hence, the distance between a reference object and facts is always one join.

On star, the number of joins corresponds to the number of dimensions requested on each query. On snowflake, the number of joins depends on the hierarchy levels distance from atomic levels. The worst case is the number of joins for GDWH NCB queries. Because the GDWH ER model has a high level of abstraction, each GDWH query needs to

reconstruct the multidimensional relations with joins. The SQL Query 1.1 showed this particularity: one join on GDWH ACB, two joins on the star, four on the snowflake, and eleven on GDWH NCB.

Figure 9.1 shows the average query running time for each DW architecture using SF 1. The same was done for SF 10 in Figure 9.2. Star was the fastest architecture on query running time for both scale factors, but if query 7.1 is not considered, the GDWH ACB is the fastest one. Actually, 67% of the total time on the star was spent on query 7.1, while 88% on ACB. This is a consequence of the use of SQL clauses GROUP BY, HAVING, and ORDER BY in a huge result set, as no filtering on dimensions was applied to this query. GDWH ACB spends the double of the star time to execute the query 7.1 for SF 10. For queries 7.2, 9.1, and 10.1, GDWH ACB was also slightly slower.

The query 8.1 for SF 10 was executed in 24.39 seconds on star, while on GDWH NCB was 16 hours. This represents more than two thousand times slower. For SF 1, GDWH NCB was ten times slower than star. Query 8.1 has a filter on the atomic level, which means it still is possible to query atomic levels with filters on star. But on GDWH, querying and filtering atomic levels is rather slow. The abstraction level of GDWH leads to more joins on queries, and as a consequence, the query running time is higher than star or snow. Filtering atomic levels on GDWH is time-consuming because reference objects combinations and hierarchies have to be first joined, then filtered.

DB Size

Figure 9.1 and Figure 9.2 show in the background a black area that represents the DB size (with index) evolution for each DW architecture and scale factors. Snowflake is the smallest DB for both scale factors. For SF 1, snowflake DB size is 5% smaller than star; for SF 10, snowflake is 2% smaller. For both scale factors, the GDWH NCB DB size grows ten times if compared with snowflake. The reference object combination table on GDWH NCB for SF 10 represents 52% of the total DB space, while fact table represents 40%. If only filtered data is saved on GDWH DYN, the DB grows slowly. GDWH ACB is the most extreme case because it grows twenty times if compared with snowflake.

The DB size variation between DW architectures was quite different. However, each architecture kept a linear DB size variation. The reason is that TPC-H is a synthetic DB with fixed variations on generated data. DB size constant variation was essential to compare the performance of queries in a stable scenario, but in a real business case, this may not be the right representation. It is clear to observe that GDWH is the most sensible DW architecture in terms of query running time in scenarios where DB size rapidly increases.

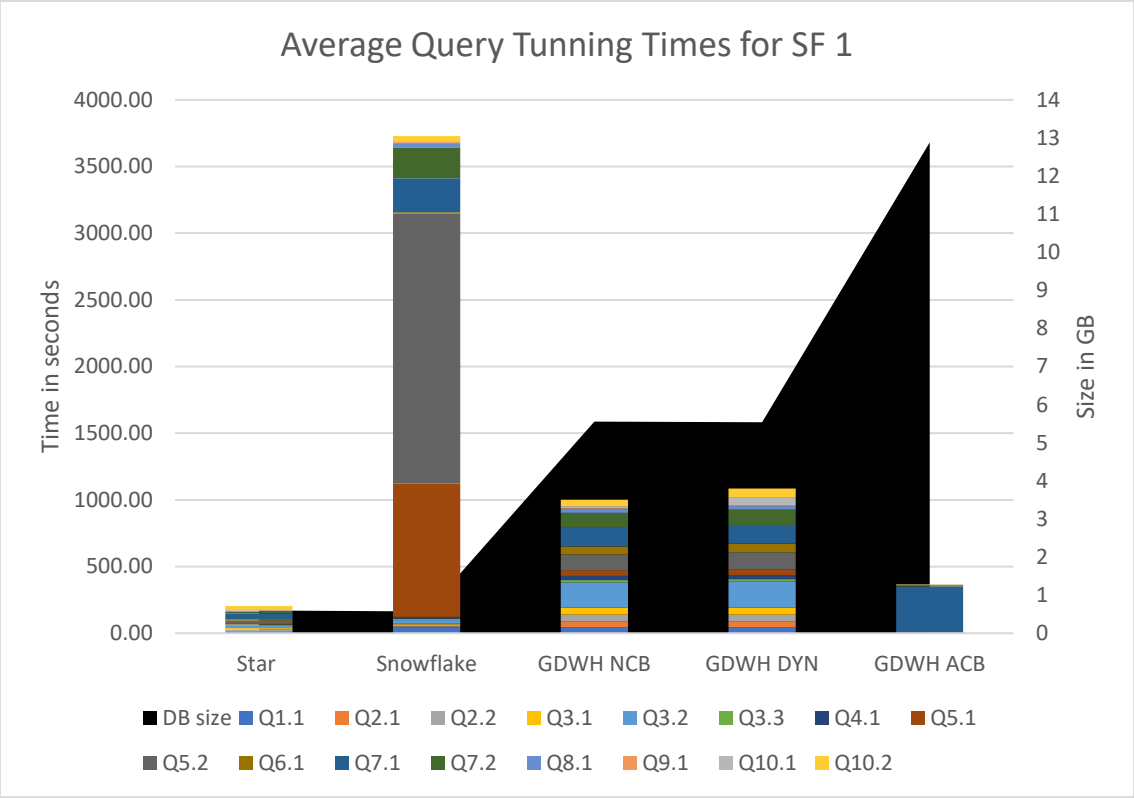


Figure 9.1 Average query running time (seconds) for scale factor 1

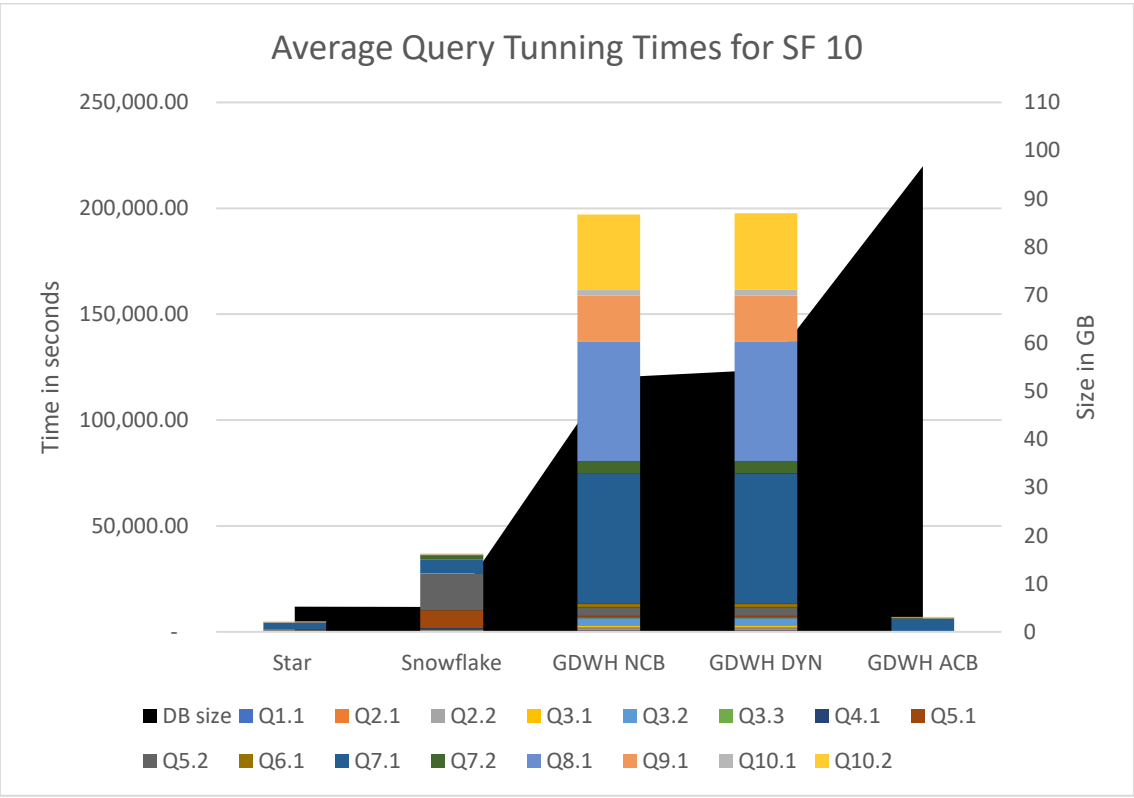


Figure 9.2 Average query running time (seconds) for scale factor 10

The conclusion of Darmont et al. (2007) that index size grows as much as data on fact tables was also observed in this study. For SF 10 in GDWH NCB, 46% of the fact table size corresponds to the index size. In the reference object combination table, the index length represents 32% of the table size, although the reference object combination table is bigger than the fact table. The fact table on star and snowflake is four times smaller than GDWH. On star and snow, ratio values are stored in columns. On GDWH, each ratio value is a new line that represents the relation between a reference object and a ratio. If a DW has five ratios, the fact table on GDWH will have five times more rows than the star or snowflake.

If DB size grows linearly, the query running time can be estimated based on the SF variation observed. The average query time for star, snowflake, and GDWH ACB increase together on the scale of 10^8. If SF is multiplied by ten, snowflake average query time increases 9.8 times; GDWH ACB 19.2 times; and star 24 times. Based on this estimation, the GDWH ACB is faster than star but slower than snowflake. The estimated average query time on snowflake for SF 1,000 is 60.3 hours. The worst case is GDWH NCB because it increases on the scale of 10^{14}. If SF is multiplied by 10, the average query time increases 189 times. The estimated average query time on GDWH NCB for SF 1,000 is fourteen years.

ETL Performance

Improving ETL performance is related to avoiding bottlenecks. In the ETL design, it has to be decided if input data should be first grouped, or first filtered, or first sorted, then, passed to the next step. These tasks usually slow down the entire ETL process, configuring a bottleneck. A shortcut is manipulating data directly with SQL because it is faster than reading all data and processing it later on inside Pentaho. Hence, SQL clauses ORDER BY and GROUP BY, and filters were used directly on the input table steps. Data type transformations were also done with SQL. Using SQL on input data tasks reduces the portability of ETL tasks to other database technologies. That is why efforts on the ETL designing were done to try to keep the tasks as much generic as possible. However, the optimal balance between generic tasks and performance is hard to achieve.

Pentaho offers the possibility of reading only a part of the input data and process it in memory. For both scale factors 1 and 10, Pentaho standard configuration was used to handle input data. The strategies utilized to reduce the amount of incoming data was to split input data by order year and to create temporary tables to store intermediate (staging) values. Then, staging data were used as input data for the next tasks. For output data, commit values between 10 and 100,000 were tested. The commit value of 1000 was the best value found; thus, this value was used on all output table tasks. It was challenging to

keep a balance between data reading, data in memory, joins, calculations, and data writing on ETL tasks. The optimal configuration of Pentaho and MySQL is a major challenge.

The ETL on the star was the fastest, and DB sizes increased proportionately to the scale factors. That is why the star was ranked first on the ETL performance criterion. Star reflects the proportional increase in table size on TPC-H. The time spent to load dimension data was 10%, and 90% of the total time was spent to load fact data. The fact table size corresponds to 91% of the total star DB size. The time of dimension loading is more related to joins and denormalization tasks, while on fact loading, is more related to calculations and insert operations. The update process time of the star is also proportional to the insertion process, i.e., if a new block of data must be inserted, the update time is close to the insertion time for the same block size. The star ETL execution time increases three-fold if the SF is multiplied by 10.

The time spent on fact data loading was 2.5 longer than dimensional data loading for the snowflake ETL. Dimension tables are normalized on the snowflake; thus, they are smaller than fact tales and faster to be fulfilled. Although normalizations for the product, customer, and order tables had to be done, the dimensional data loading was faster than the same process on the star ETL. On the other hand, the fact loading was slower than the star.

The DYN ETL time increases eighteen times if SF is multiplied by ten. The problem is the lookup process for reading reference objects. Tables reference object hierarchy and reference object combination must be merged for each atomic level and filtered if needed. After data were read, inserting new reference objects and facts is a fast process. A batch update was done, splitting data by year as a trial to improve performance, but no significant improvement was achieved.

The total query time of ACB for SF 1 is 76% faster than the NCB. For SF 10, ACB is 96% faster. However, the ACB ETL was the most time consuming, and it generated the biggest DB among all ETLs. The execution time is related to the total number of dimension combinations and the corresponding reference objects to be loaded. The total number of dimension combinations can be calculated using the powerset[10] definition: 2^n, where *n* is the number of dimension levels. As the first subset is an empty set, it must be removed because there is no empty dimension level. Thus, the formula used to calculate all dimension combinations is $2^n - 1$.

[10] A powerset consists of all subsets of a set. If the set *S* = {a, b, c}, the powerset of *S* is:{{}, {a}, {b}, {c}, {a,b}, {a,c}, {b,c}, {a,b,c}}.

The multidimensional model used in this research has twelve dimension levels (including atomic levels), that means 4,095 (2^{12} - 1) possible combinations exist. Loading the combinations is not a problem because they are read from a text file and directly inserted into the corresponding tables, but reading and saving the reference objects is a bottleneck. The query to read the reference objects is the same used on the GDWH NCB, which slows down the process considerably. For SF 10, the number of reference objects to be loaded reaches a billion lines.

The *small* TPC-H DB, with a hundred orders used only for tests, took 48 hours to load 2,000 dimension combinations, and DB size increased from 19,6 MB to 60 GB. Even after using workarounds to improve performance, like caching, commit size and batch update for inserts, it was not possible to load all combinations for ACB ETL. That is why only the eleven dimension combinations derived from the business questions were loaded.

The ETL time estimation for scale factors 100 and 1,000 shows that for star and snowflake, there is a three-fold time increase if SF is multiplied by ten. For GDWH NCB, there is a fifteen-fold increase. The estimated ETL time on star for SF 1,000 is 52 hours, while on GDWH NCB is 8.5 months. Of course, an ETL running for months makes no sense at all. Hardware improvements and splitting data are necessary to implement DW architectures on SF greater than ten.

Data Consistency and Redundancy

In the matter of data consistency and redundancy, the GDWH NCB obtained the best grade because of its high level of abstraction. If all premises of the classic GDWH are respected, the generic representation of multidimensional data as reference objects and dimensions ensures a high level of data consistency and avoids data redundancy. In the DYN and ACB variations, these rules were partially ignored as trials to improve performance. Thus, the DB is more vulnerable to data redundancy and data inconsistency.

Although the dynamic ETL improves query performance by saving DW user's questions, foresee all possible questions is not possible. The querying running time for facts already stored (GDWH ACB) is much faster, but the process of saving new reference objects and facts may lead to data inconsistency. If a filter is applied to a ratio on the GDWH DYN, e.g., purchase amount greater than 100, only a portion of reference objects and facts related to a dimension combination will be inserted. In this case, it is not suggested to store only the values returned by the query because a reference object can have more facts with the purchase amount smaller than 100. A workaround to keep data consistency in GDWH DYN would be to store only the new dimension combinations and ratios metadata in a temporary table, and update table values in another moment.

Metadata management

DW metadata management consists of changing dimensions, hierarchies, and ratios definitions. If a new dimension must be inserted on star or snowflake schema, it means that a new physical table must be created. DW designers need to pay attention to the existing tables and relations to avoid inconsistency. After inserting a new dimension table, the fact table must be updated with the new fact values. Null values may occur because old facts may not have reference to new dimensional data. As the DW grows, star and snowflake schemas become incomprehensible and can lead to data inconsistency. That is why star was in the 5th and snowflake in the 4th position of this criterion.

From all DW architectures compared in this study, the GDWH NCB is the best model in the metadata management matter. No physical DB changes need to be implemented if metadata changes. If a new dimension must be inserted, it represents a new entry on the dimension table and its reference objects instantiation. To insert new facts, there is no need to create a new column in the fact table and null values are avoided. The GDWH NCB DW architecture assures a high level of data consistency and avoids data redundancy, which justifies the first position.

The GDWH DYN and ACB variations are not as good as the NCB in terms of metadata management. If a new dimension and atomic levels must be inserted, it may be the case that current reference objects or facts are affected. Especially if atomic level combination changes, the new dimension combinations of DYN and ACB may overlap the old data and create data inconsistency. DYN was classified as second and ACB as third place.

The DB physical model is also related to metadata management because it helps DW designers to read and understand the DW implementation. In the star, it is hard to identify hierarchy levels because they are denormalized. In the dimension product, for instance, it is not possible to identify if a manufacturer group has brands or vice versa. This can lead to wrong cube and query specifications.

In the snowflake, hierarchies are identifiable by the (1:n) relations. Looking at the snowflake implementation, it is clear that a manufacturer group has many brands, and a brand has many products. The business relations are kept on the physical model, which reduces the risk of errors in query specification. Physical model validation with business users is also possible with a snowflake schema.

Due to its high level of abstraction, the GDWH has the most complex physical representation. If the multidimensional conceptual model is not well designed, errors are not easily identifiable before running the queries. It is not possible to validate the physical

model with business users without auxiliary tables with real examples. DW designers can also misunderstand the notion of atomic levels since all queries on NCB depends on the correct specification and relation between atomic levels.

Figure 9.3 represents the ranking position for each DW architecture, considering each evaluation criterion. The three criteria in the top, DB Size, Query Performance, and ETL Performance, are related to DW architecture performance. The two criteria in the bottom are related to data management and data quality.

The star architecture is well ranked in the performance area, while GDWH NCB and DYN are well ranked in the data management and data quality area. GDWH NCB was the architecture without a single fifth position, but on query performance, it was ranked as fourth. The GDWH ACB is the second-fastest in query performance, but it is the worst in all other evaluation criteria. The snowflake architecture is the intermediate solution because its query performance is the third better, DB size the smallest, and it preserves data consistency and avoids data redundancy.

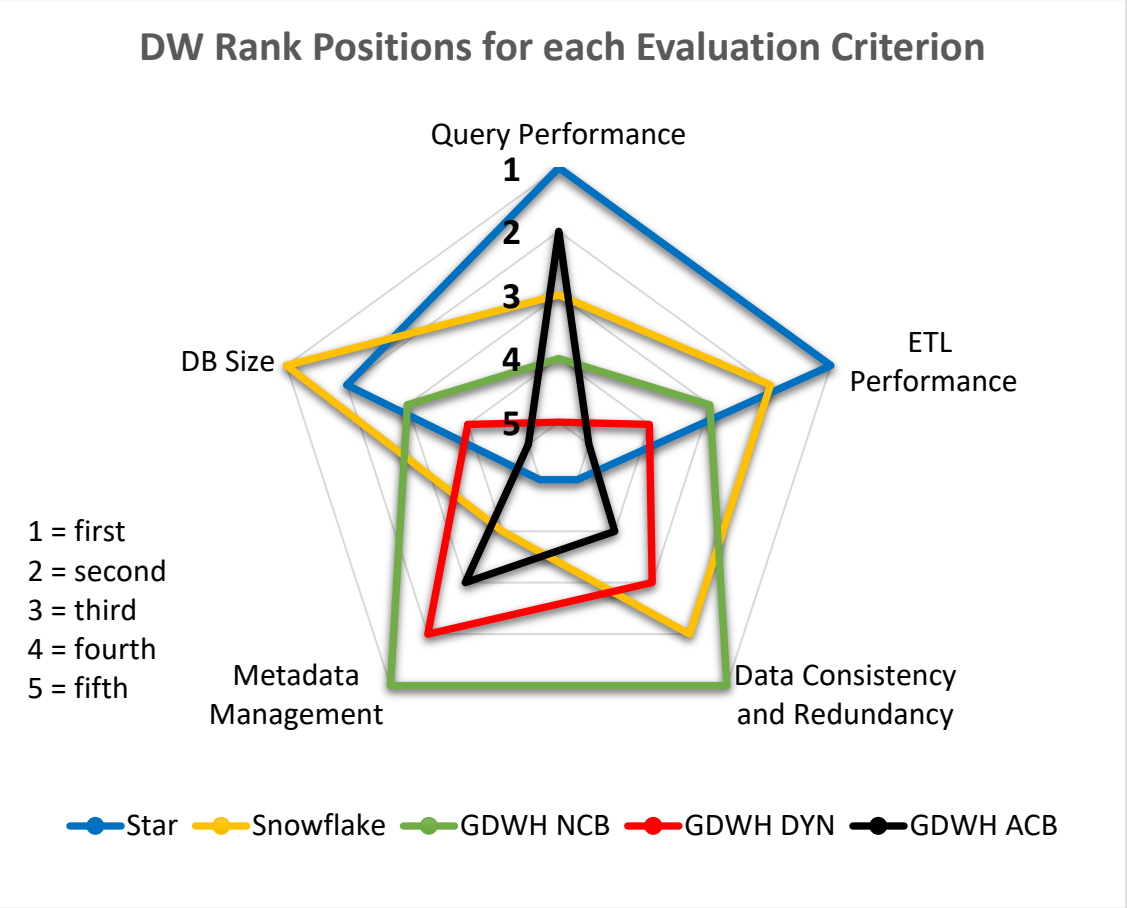


Figure 9.3 Data Warehouse architectures rank positions for each evaluation criterion

There is a clear trade-off between the DW architectures analyzed in this study. It is not possible to establish the best approach, which optimizes both query/ETL performance and metadata management/data consistency and redundancy. For small DBs, the GDWH NCB and DYN can be used for rapid implementation and straightforward metadata management, keeping data consistency. If DB size is huge, one should choose star (if the metadata does not change frequently), or snow (if metadata changes often).

10 Conclusion

The performance analysis of three DW architectures showed that the star schema is the best performing architecture if only query running time and ETL time is taken into account. The problem of star is the data redundancy and metadata management for complex DW architectures. In scenarios where metadata is stable, and DB size grows linearly, the estimation of the query running time and ETL time is accurate.

Snowflake query performance was the worst for SF 1. Nevertheless, snowflake kept a proportional increase of query time if DB size grows. According to the estimation, it still possible to query TPC-H data for an SF of 1,000. Improvements in hardware and ETL processes would undoubtedly improve the overall performance of snowflake. The physical data representation is also an advantage if compared with the two other DW architectures compared.

Even after ETL variations (NCB, DYN, and ACB), the GDWH query and ETL performance still much slower than star or snowflake. The ACB variation was the fastest in query running time for 84% of all queries, but the ETL time is extremely high for scale factors greater than one. For big databases, a star or snowflake architecture is more suitable. The GDWH DYN variation has advantages over NCB if the dimension combinations queried by the user are first stored, and the DYN ETL is scheduled to run in another moment with all reference objects and facts. Saving only filtered facts can lead to data inconsistency and increase lookup time.

A workaround to improve performance for all DW architectures would be to ask DW/BI users the most common questions that a DW should answer. Based on those questions, ETL and queries can be improved. However, limiting user flexibility on DW systems is not the goal of designing a BI application. With the constant increase of computation power, DW architectures should offer more self-service reports to users, independent of the dimension combinations or atomic levels queried.

The scope of this study was to compare performance between DW architectures per se, without DB improvements after ETL processes. The use of index techniques, for instance, could improve performance on GDWH NCB to a level that the ACB ETL effort is not worth anymore. The problem is to find out the best index techniques for huge tables on GDWH architecture. The search space and combinations of such techniques may be endless.

Split a DW into small Data Marts is a common strategy adopted to improve performance. Future studies could analyze the impact of using Data Marts with GDWH architecture

and its ETL variations. The classical GDWH definition contains nine tables to represent any DW. The problem is if the data source is huge, fact and reference object tables grow exponentially, causing loss of query performance. In a Data Mart, those tables could be split by time, or geographical location, or department. That means a GDWH architecture would have more than nine tables.

Other possible improvements to DW performance that this study did not explore are related to technical aspects of data storage and DB technologies for GDWH. Future research could test the DW performance with table partitioning or clustering techniques offered by DB vendors. The GDWH architectures could also be tested with In-memory DBs, NoSQL, or Big Data environments like Hadoop.

References

Adamson, C. 2010. *The complete reference star schema*, New York: McGraw-Hill.

Bala, M., Boussaid, O., and Alimazighi, Z. 2016. "Extracting-Transforming-Loading Modeling Approach for Big Data Analytics," *International Journal of Decision Support System Technology* (8:4), pp. 50–69.

Becker, J., Janiesch, C., Pfeiffer, D., and Seidel, S. 2006. "Evolutionary method engineering: towards a method for the analysis and conception of management information systems," *AMCIS 2006 Proceedings* , p. 470.

Becker, J., and Schütte, R. 2004. *Handelsinformationssysteme*, Frankfurt am Main: Redline Wirtschaft bei Verl. Moderne Industrie.

Becker, J., and Winkelmann, A. 2019. *Handelscontrolling*, Berlin, Heidelberg: Springer Berlin Heidelberg.

Bizarro, P., and Madeira, H. 2002. "Adding a Performance-Oriented Perspective to Data Warehouse Design," in *Data warehousing and knowledge discovery*: *4th international conference ; proceedings*, Y. Kambayashi (eds.), Berlin: Springer, pp. 232–244.

Bulos, D., and Forsman, S. 2006. "Getting started with ADAPT-OLAP Database Design," *White Paper* .

Campbell, A., Mao, X., Pei, J., and Al-Barakati, A. 2017. "Multidimensional benchmarking in data warehouses," *Intelligent Data Analysis* (21:4), pp. 781–801.

Chaudhuri, S., and Dayal, U. 1997. "An overview of data warehousing and OLAP technology," *ACM SIGMOD Record* (26:1), pp. 65–74.

Darmont, J. 2009. "Data Warehouse Benchmarking with DWEB," in *Progressive methods in data warehousing and business intelligence*: *Concepts and competitive analytics*, D. Taniar (eds.), Hershey, Pa.: IGI Global (701 E. Chocolate Avenue, Hershey, Pennsylvania, 17033, USA), pp. 302–323.

Darmont, J., Bentayeb, F., and Boussaïd, O. 2007. "Benchmarking data warehouses," *International Journal of Business Intelligence and Data Mining* (2:1), p. 79.

Dehdouh, K., Boussaid, O., and Bentayeb, F. 2014. "Columnar NoSQL Star Schema Benchmark," in *Model and Data Engineering: 4th International Conference*, Y. Ait Ameur, L. Bellatreche and G. A. Papadopoulos (eds.), Cham: Springer, pp. 281–288.

Demarest, M. 1997. *The Politics of Data Warehousing*.

El Akkaoui, Z., and Zimanyi, E. 2009. "Defining ETL worfklows using BPMN and BPEL," in *Proceedings of the Acm Twelfth International Workshop on Data Warehousing and Olap*, I.-Y. Song (eds.), Hong Kong, China. 11/6/2009 - 11/6/2009, [Place of publication not identified]: Association for Computing Machinery, p. 41.

Gang, T., Kai, C., and Bei, S. 2008. "The research & application of Business Intelligence system in retail industry," in *IEEE International Conference on Automation and Logistics, 2008*: *ICAL 2008 ; 1-3 Sept. 2008, Qingdao, China*, Qingdao, China. 9/1/2008 - 9/3/2008, Piscataway, NJ: IEEE, pp. 87–91.

Garani, G., and Helmer, S. 2012. "Integrating Star and Snowflake Schemas in Data Warehouses," *International Journal of Data Warehousing and Mining* (8:4), pp. 22–40.

J. Gray (ed.) 1993. *The Benchmark handbook*: *For database and transaction processing systems*, San Francisco, Cal.: Kaufmann.

Gupta, V., and Singh, J. 2014. "A Review of Data Warehousing and Business Intelligence in different perspective," .

Harrison, R., Parker, A., Brosas, G., Chiong, R., and Tian, X. 2015. "The role of technology in the management and exploitation of internal business intelligence," *Journal of Systems and Information Technology* (17:3), pp. 247–262.

Huppler, K. 2009. "The Art of Building a Good Benchmark," in *Performance evaluation and benchmarking*: *First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009 ; revised selected papers*, R. Nambiar and M. Poess (eds.), Berlin: Springer, pp. 18–30.

Inmon, W. H. 2005. *Building the data warehouse*, Indianapolis, IN: Wiley Pub.

Inmon, W. H., Strauss, D., and Neushloss, G. 2008. *DW 2.0*: *The architecture for the next generation of data warehousing*, Amsterdam, Boston: Morgan Kaufmann/Elsevier.

J. Gray, A. Bosworth, A. Lyaman, and H. Pirahesh 1996. "Data cube: a relational aggregation operator generalizing GROUP-BY, CROSS-TAB, and SUB-TOTALS," in *Proceedings of the Twelfth International Conference on Data Engineering*, pp. 152–159.

Kimball, R., and Ross, M. 2002. *The data warehouse toolkit*: *The complete guide to dimensional modeling*, New York, NY: Wiley.

Kimball, R., Ross, M., and Kimball, R. D. w. t. : t. c. g. t. d. m. 2013. *The data warehouse toolkit*: *The definitive guide to dimensional modeling*, Indianapolis, Ind.: John Wiley & Sons.

Leff, A., and Rayfield, J. T. 2001. "Web-application development using the Model/View/Controller design pattern," in *Fifth IEEE International Enterprise Distributed Object Computing Conference*: *September 4-7, 2001, Seattle, Washington USA*, F. M. Titsworth (eds.), Seattle, WA, USA. 4-7 Sept. 2001, Los Alamitos: I E E E [Imprint]; IEEE Computer Society Press, pp. 118–127.

Malinowski, E., and Zimányi, E. 2009. *Advanced data warehouse design*: *From conventional to spatial and temporal applications*, Berlin: Springer.

Manuel Serrano, Coral Calero, Juan Trujillo, and Sergio Luján and Mario Piattini 2004. "Towards a Metrics Suite for Conceptual Models of Datawarehouses," in *Proceedings of the 1st International Workshop on Software Audits and Metrics,* Porto, Portugal. 4/14/2004 - 4/17/2004, SciTePress - Science and and Technology Publications, pp. 105–117.

Martyn, T. 2004. "Reconsidering Multi-Dimensional schemas," *ACM SIGMOD Record* (33:1), p. 83.

Michael Stonebraker, Chuck Bear, Uur Çetintemel, Mitch Cherniack, Tingjian Ge, Nabil Hachem, Stavros Harizopoulos, John Lifter, Jennie Rogers, and Stan Zdonik 2007. "One size fits all? - Part 2: Benchmarking results," *CIDR 2007 - 3rd Biennial Conference on Innovative Data Systems Research* , pp. 173–184.

Nanz, S., and Furia, C. A. 2015. "A Comparative Study of Programming Languages in Rosetta Code," in *BIGDSE 2015*: *First International Workshop on BIG Data Software Engineering : proceedings : May 23, 2015, Florence, Italy,* Florence, Italy. 5/16/2015 - 5/24/2015, Los Alamitos, California, Washington, Tokyo: Conference Publishing Services, IEEE Computer Society, pp. 778–788.

O'Neil, P. 1993. "The Set Query Benchmark," in *The Benchmark handbook*: *For database and transaction processing systems*, J. Gray (eds.), San Francisco, Cal.: Kaufmann, pp. 209–245.

O'Neil, P. E., O'Neil, E. J., and Chen, X. 2006. *Star Schema Benchmark*.

Sanchez, J. 2016. "A Review of Star Schema Benchmark,"

Sapia, C., Blaschka, M., Höfling, G., and Dinter, B. 1998. “Extending the E/R model for the multidimensional paradigm,” in *International Conference on Conceptual Modeling*, pp. 105–116.

Saroop, S., and Kumar, M. 2011. “Comparison of Data Warehouse Design Approaches from User Requirement to Conceptual Model: A Survey,” in *2011 International Conference on Communication Systems and Network Technologies*, Katra, Jammu, India. 6/3/2011 - 6/5/2011, [Place of publication not identified]: IEEE, pp. 308–312.

Tiobe 2019. *Programming Community Index*. http://www.tiobe.com. Accessed October 2019.

Transaction Processing Performance Council (TPC) 2018. *TPC Benchmark H*: *Decision Support* (Standard Specification Revision 2.18.0). http://www.tpc.org. Accessed May 2019.

Trujillo, J., and Luján-Mora, S. 2003. “A UML Based Approach for Modeling ETL Processes in Data Warehouses,” in *Conceptual modeling-ER 2003*: *22nd International Conference on Conceptual Modeling, Chicago, IL, USA, October 2003 : proceedings*, I.-Y. Song (eds.), Berlin, New York: Springer-Verlag, pp. 307–320.

Vassiliadis, P., Simitsis, A., and Skiadopoulos, S. 2002. “Conceptual modeling for ETL processes,” in *DOLAP 2002*: *ACM fifth International Workshop on Data Warehousing and OLAP, in conjunction with the eleventh International Conference on Information and Knowledge Management (CIKM 2002), November 8, 2002, McLean, Virginia, USA. Sponsored by the Association for Computing Machinery, Special Interest Group on Information Retrieval (SIGIR), and Special Interest Group on Management Information Systems(SIGMIS)*, D. Theodoratos (eds.), McLean, Virginia, USA. 11/8/2002 - 11/8/2002, New York: ACM, pp. 14–21.

Appendix

A ER Model from TPC-H

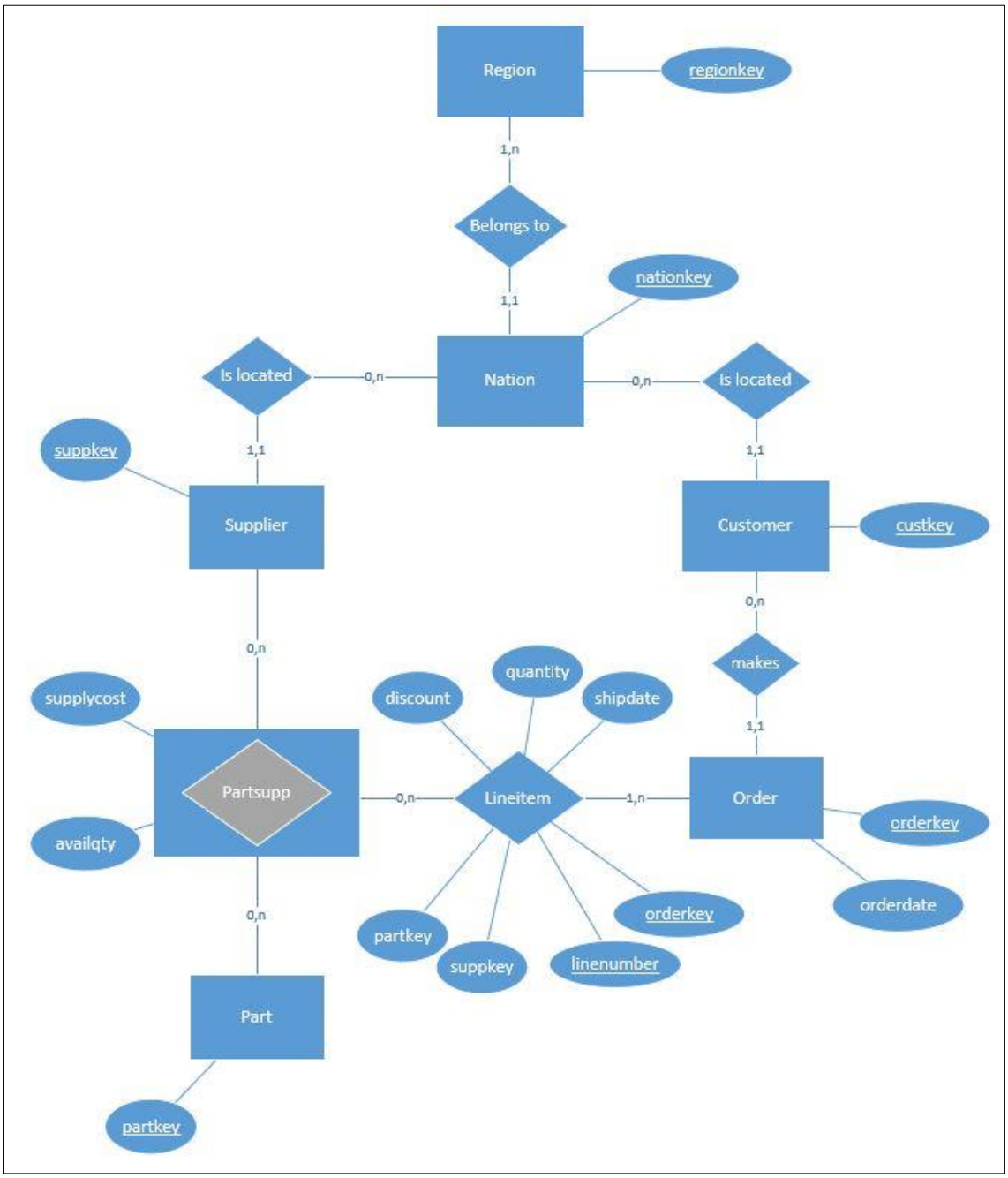


Figure A.1 TPC-H ER model

B HammerDB TPC-H MySQL Build

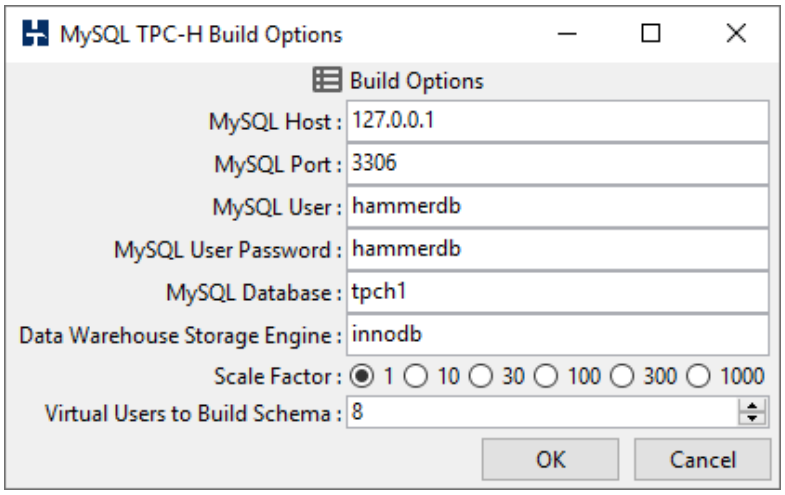


Figure B.1 HammerDB TPC-H Build Options

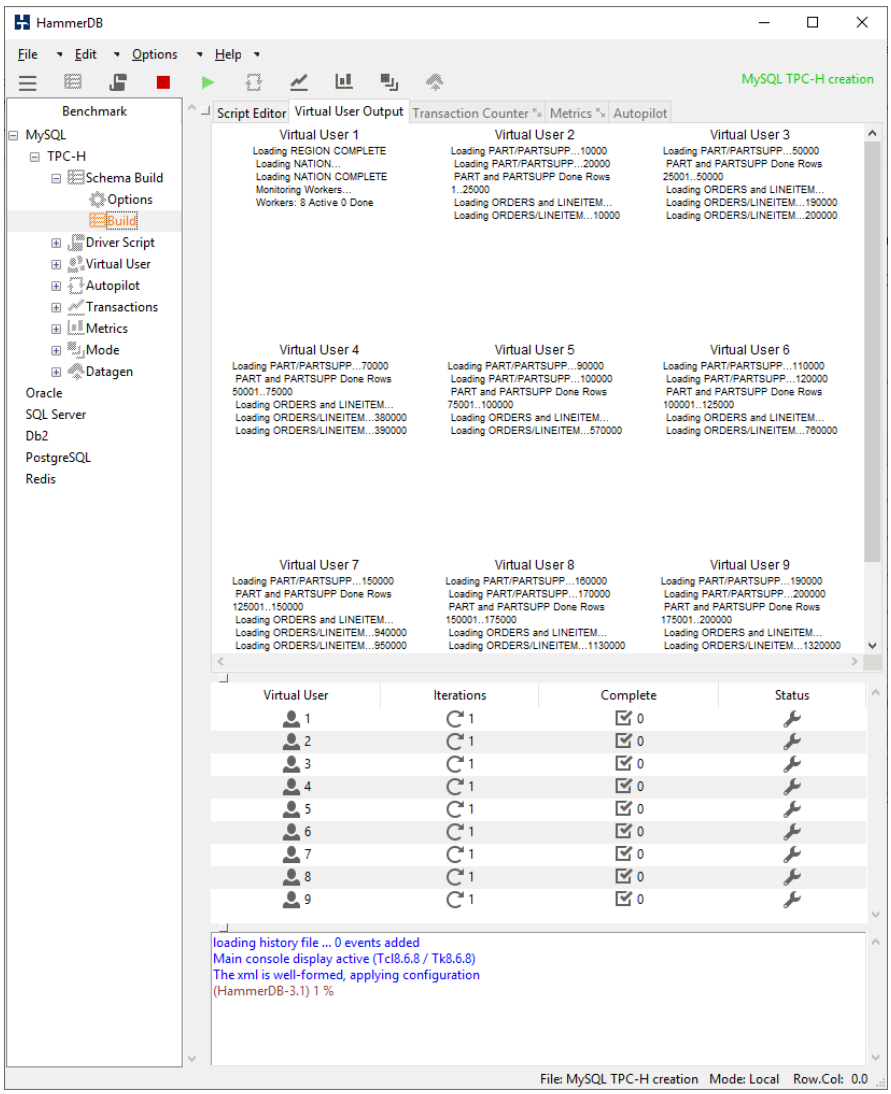


Figure B.2 HammerDB TPC-H data generation

C Software used on the System Architecture

Software with version	Description
Linux Ubuntu 18.04.2 LTS	OS where the Java web-application is running.
MySQL 8.0.16	Relational Database used to store data.
MySQL Workbench 8.0.16	Relational Database Management System (RDBMS). The graphical interface tool to manage schemas.
Java 11	Object-Oriented language selected to develop the application to query different DW architectures.
Bootstrap 4	Framework used on the fronted.
Firefox Browser Developer Edition 71	Used to test and debug the web-application.
HeidiSQL (HS) 10.2	Used to manage database schemas (test queries, copy tables, etc.).
Model-View-Controller (MVC)	Software development pattern adopted on Java web application.
Spring Framework 5.2.2 RELEASE	Open source framework which fully supports Java web applications and comes with many modules that offers a set of services: data access, authorization and authentication, MVC, remote management, etc.
Spring Boot 2.1.5 RELEASE	It creates and runs Spring applications, configure necessary dependencies and comes with embed HTTP web server (Tomcat, Jetty or Undertow).
Hibernate 5.3.10 Final	Object-Relation Mapping (ORM) framework which offers flexibility, scalability and extensibility to manage data access on Java web applications.
Apache Maven 3.6.2	Project Object Model (POM). The Java project contains an XML file where the all dependencies and modules are configured.
Thymeleaf 3.0.11	View layer where objects were displayed after processed by the controllers. This is an alternative to JavaServer Pages (JSP).
GitHub	Repository versioning used to store the Java web-application
IntelliJ (IJ) from JetBrains	Integrated Development Environment (IDE) used to develop the Java web application.
Microsoft Vision	It was used to create the multidimensional model with ADAPT notation.
Pentaho Data Integration (PDI) 8.2 (also know as Kettle)	Used to design and run all ETL processes.
HammerDB 3.1	Tool for TPC-H data generation. Different scale factors can be set.

Table C.1 Software used in the system architecture

D Model Layer on Java Project

Figure D.1 Model layer on Java project

E Controller Layer on Java Project

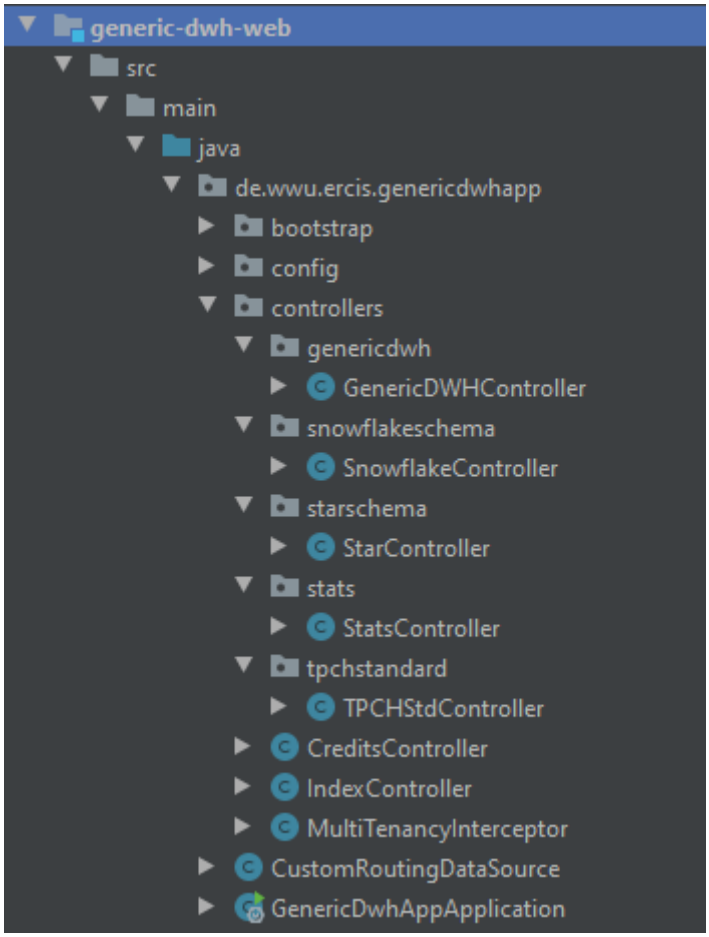


Figure E.1 Controller layer on Java project

F View Layer on Java Project

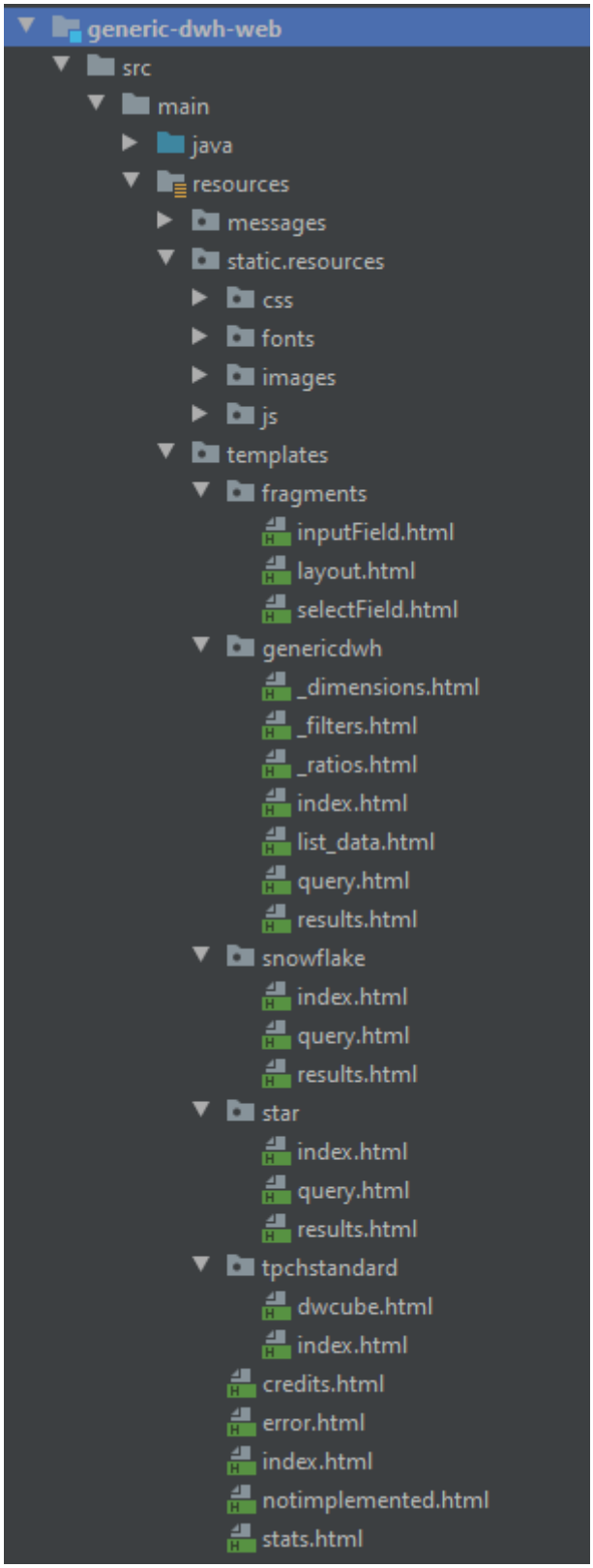


Figure F.1 View layer on Java project

G Query Running Times for Scale Factor 1

	Star				Snowflake				GDWH NCB				GDWH DYN				GDWH ACB				Rows
	1	**2**	**3**	**mean**	**1**	**2**	**3**	**mean**	**1**	**2**	**3**	**mean**	**1**	**2**	**3**	**mean**	**1**	**2**	**3**	**mean**	
Q1.1	9.02	9.82	8.82	**9.22**	50.34	40.11	40.32	**43.59**	45.88	41.95	43.47	**43.77**	43.86	43.86	43.85	**43.85**	0.03	0.02	0.03	**0.03**	1
Q2.1	8.93	8.79	8.72	**8.81**	7.09	7.12	7.02	**7.08**	41.88	42.69	42.81	**42.46**	42.57	42.66	42.59	**42.61**	0.02	0.01	0.01	**0.01**	1
Q2.2	9.45	9.43	9.33	**9.40**	13.73	13.66	13.60	**13.66**	68.68	53.66	34.79	**52.38**	52.80	52.87	52.82	**52.83**	0.01	0.01	0.01	**0.01**	9
Q3.1	14.42	9.80	9.68	**11.30**	11.22	6.30	6.21	**7.91**	56.66	60.96	42.52	**53.38**	53.52	53.52	53.51	**53.52**	0.08	0.06	0.07	**0.07**	1
Q3.2	16.32	16.33	16.38	**16.34**	27.07	27.51	27.44	**27.34**	198.95	199.48	171.53	**189.99**	196.48	196.45	196.36	**196.43**	0.05	0.05	0.04	**0.05**	100
Q3.3	10.24	10.34	10.47	**10.35**	8.45	8.45	8.51	**8.47**	16.21	16.64	16.19	**16.35**	16.95	16.92	16.94	**16.93**	0.03	0.03	0.03	**0.03**	9
Q4.1	2.52	2.49	2.54	**2.52**	13.90	13.85	14.25	**14.00**	30.57	31.15	30.32	**30.68**	30.83	30.83	30.83	**30.83**	0.06	0.05	0.06	**0.06**	2
Q5.1	9.26	9.48	9.53	**9.42**	1010.44	998.31	998.37	**1002.37**	46.35	44.61	42.07	**44.34**	44.60	44.59	44.58	**44.59**	0.01	0.01	0.01	**0.01**	5
Q5.2	15.76	16.06	16.22	**16.01**	2031.43	2007.95	2027.62	**2022.33**	117.33	116.56	116.99	**116.96**	127.24	127.36	127.27	**127.29**	0.01	0.01	0.01	**0.01**	320
Q6.1	11.68	11.88	11.67	**11.74**	12.42	10.53	10.63	**11.19**	59.57	60.47	60.13	**60.06**	62.40	62.88	62.80	**62.69**	6.26	5.93	5.88	**6.02**	42
Q7.1	40.30	42.66	39.88	**40.95**	271.07	237.51	252.29	**253.62**	144.58	137.85	138.04	**140.16**	141.00	140.97	140.99	**140.98**	347.70	343.05	344.08	**344.94**	23
Q7.2	16.07	15.84	17.95	**16.62**	235.47	224.27	226.76	**228.83**	110.89	112.10	112.48	**111.82**	113.96	113.93	113.95	**113.95**	9.02	9.12	9.06	**9.07**	66
Q8.1	2.83	2.82	2.83	**2.83**	34.95	34.43	35.22	**34.87**	26.52	25.19	25.76	**25.82**	26.35	26.36	26.37	**26.36**	0.03	0.04	0.03	**0.03**	5
Q9.1	5.75	5.76	5.85	**5.79**	7.64	7.52	7.56	**7.57**	8.20	7.60	7.50	**7.77**	8.48	8.46	8.50	**8.48**	1.75	1.70	1.74	**1.73**	10
Q10.1	8.43	8.39	8.49	**8.44**	8.23	8.20	8.12	**8.18**	18.54	15.73	15.40	**16.56**	53.23	53.53	53.35	**53.37**	2.24	1.89	1.99	**2.04**	761
Q10.2	23.16	23.99	24.68	**23.94**	44.29	35.70	35.44	**38.48**	52.64	47.20	47.73	**49.19**	71.58	71.69	71.15	**71.47**	1.45	1.43	1.42	**1.43**	451
Sum	**204.14**	**203.88**	**203.04**	**203.69**	**3787.74**	**3681.42**	**3719.36**	**3729.51**	**1043.45**	**1013.84**	**947.73**	**1001.67**	**1085.83**	**1086.86**	**1085.84**	**1086.18**	**368.75**	**363.41**	**364.47**	**365.54**	**1806.00**
Mean	**12.76**	**12.74**	**12.69**	**12.73**	**236.73**	**230.09**	**232.46**	**233.09**	**65.22**	**63.37**	**59.23**	**62.60**	**67.86**	**67.93**	**67.87**	**67.89**	**23.05**	**22.71**	**22.78**	**22.85**	**112.88**

Table G.1 Query running times for scale factor 1

H Query Running Times for Scale Factor 10

	Star				Snowflake				GDWH NCB				GDWH DYN				GDWH ACB				Rows
	1	2	3	mean	1	2	3	mean	1	2d	3	mean	1	2	3	mean	1	2	3	mean	
Q1.1	77.35	82.08	76.53	**78.65**	378.98	431.67	530.48	**447.04**	719.59	637.60	773.92	**710.37**	710.50	710.53	710.52	**710.52**	0.06	0.05	0.02	**0.04**	1
Q2.1	76.14	76.66	74.84	**75.88**	59.28	60.69	58.19	**59.39**	865.62	772.31	662.04	**766.66**	766.77	766.79	766.78	**766.78**	0.03	0.02	0.02	**0.02**	1
Q2.2	88.14	81.44	80.78	**83.45**	114.22	121.58	115.81	**117.20**	876.73	784.60	687.73	**783.02**	783.48	783.47	783.44	**783.46**	0.01	0.01	0.01	**0.01**	9
Q3.1	126.86	85.26	82.70	**98.27**	92.14	112.96	84.86	**96.65**	871.57	893.34	782.19	**849.03**	849.21	849.18	849.23	**849.21**	0.05	0.06	0.04	**0.05**	1
Q3.2	137.98	143.09	137.56	**139.54**	235.57	250.67	233.43	**239.89**	3,400.14	3,643.57	3,144.28	**3,396.00**	3,402.63	3,402.48	3,402.51	**3,402.54**	0.03	0.03	0.03	**0.03**	100
Q3.3	90.39	91.78	88.79	**90.32**	71.55	74.58	71.25	**72.46**	342.69	364.82	319.46	**342.32**	342.89	342.89	342.90	**342.90**	0.02	0.02	0.01	**0.02**	9
Q4.1	79.83	80.64	77.66	**79.38**	623.28	694.34	684.84	**667.49**	239.19	257.68	243.23	**246.70**	246.86	246.87	246.85	**246.86**	0.24	0.21	0.20	**0.22**	2
Q5.1	82.36	84.48	80.20	**82.35**	8,440.75	8,557.82	8,645.84	**8,548.14**	790.71	853.55	742.69	**795.65**	795.89	795.89	795.89	**795.89**	0.01	0.01	0.01	**0.01**	5
Q5.2	136.86	139.09	135.73	**137.23**	17,147.50	17,450.25	17,437.02	**17,344.92**	3,753.20	3,636.79	3,606.83	**3,665.61**	3,676.20	3,675.85	3,676.11	**3,676.05**	0.01	0.01	0.01	**0.01**	320
Q6.1	104.74	112.20	106.83	**107.92**	90.75	101.43	93.29	**95.16**	1,859.66	1,941.89	1,841.85	**1,881.13**	1,883.66	1,883.64	1,883.74	**1,883.68**	29.22	28.68	28.56	**28.82**	38
Q7.1	3,298.55	3,740.65	2,872.91	**3,304.04**	6,449.68	6,449.68	6,665.53	**6,521.63**	62,357.47	61,724.47	59,689.06	**61,257.00**	61,257.81	61,257.77	61,257.78	**61,257.79**	6,159.42	6,145.06	6,188.62	**6,164.37**	22
Q7.2	159.11	183.83	170.73	**171.22**	1,940.27	1,951.04	1,992.59	**1,961.30**	5,548.38	5,939.64	6,084.16	**5,857.39**	5,858.76	5,858.74	5,858.81	**5,858.77**	248.22	252.09	251.70	**250.67**	42
Q8.1	24.39	24.68	24.64	**24.57**	291.32	294.66	306.10	**297.36**	59,932.25	57,806.31	51,447.85	**56,395.47**	56,396.02	56,396.03	56,396.02	**56,396.02**	1.33	1.41	1.44	**1.39**	5
Q9.1	56.83	56.46	58.56	**57.28**	82.24	71.83	74.11	**76.06**	22,788.14	21,870.23	20,924.07	**21,860.81**	21,861.54	21,861.50	21,861.52	**21,861.52**	159.75	165.34	165.18	**163.42**	10
Q10.1	73.69	82.61	75.56	**77.29**	70.04	84.05	72.31	**75.47**	2,694.28	2,519.71	2,568.43	**2,594.14**	2,892.02	2,893.50	2,894.99	**2,893.50**	309.10	325.47	379.67	**338.08**	6,145
Q10.2	322.87	296.72	281.41	**300.33**	332.59	416.60	325.35	**358.18**	37,148.60	36,193.90	33,494.18	**35,612.23**	35,886.27	35,885.91	35,886.18	**35,886.12**	56.88	59.22	58.08	**58.06**	4,749
Sum	**4,936.09**	**5,361.67**	**4,425.43**	**4,907.73**	**36,420.16**	**37,123.85**	**37,391.00**	**36,978.34**	**204,188.22**	**199,840.41**	**187,011.97**	**197,013.53**	**197,610.51**	**197,611.04**	**197,613.27**	**197,611.61**	**6,964.38**	**6,977.69**	**7,073.60**	**7,005.22**	**11,459.00**
Mean	**308.51**	**335.10**	**276.59**	**306.73**	**2,276.26**	**2,320.24**	**2,336.94**	**2,311.15**	**12,761.76**	**12,490.03**	**11,688.25**	**12,313.35**	**12,350.66**	**12,350.69**	**12,350.83**	**12,350.73**	**435.27**	**436.11**	**442.10**	**437.83**	**716.19**

Table H.1 Query running times for scale factor 10

Declaration of Authorship

We hereby declare that, to the best of my knowledge and belief, this master thesis titled "**Comparing Performance of Data Warehouse Designs: Implementing Generic Data Warehouse, Star, and Snowflake Schemas using TPC-H as Data Source**" is our own work. We confirm that each significant contribution to and quotation in this thesis that originates from the work or works of others is indicated by proper use of citation and references.

Münster, 20 April 2020

Pedro Henrique Marangoni (446278)

Consent Form

for the use of plagiarism detection software to check our master thesis

Pedro Henrique Marangoni

Course of Study: Master Thesis WS 2019/2020

Title of the thesis: Comparing Performance of Data Warehouse Designs: Implementing Generic Data Warehouse, Star, and Snowflake Schemas using TPC-H as Data Source

What is plagiarism? Plagiarism is defined as submitting someone else's work or ideas as your own without a complete indication of the source. It is hereby irrelevant whether the work of others is copied word by word without acknowledgment of the source, text structures (e.g. line of argumentation or outline) are borrowed or texts are translated from a foreign language.

Use of plagiarism detection software. The examination office uses plagiarism software to check each submitted bachelor and master thesis for plagiarism. For that purpose the thesis is electronically forwarded to a software service provider where the software checks for potential matches between the submitted work and work from other sources. For future comparisons with other theses, your thesis will be permanently stored in a database. Only the School of Business and Economics of the University of Münster is allowed to access your stored thesis. The student agrees that his or her thesis may be stored and reproduced only for the purpose of plagiarism assessment. The first examiner of the thesis will be advised on the outcome of the plagiarism assessment.

Sanctions. Each case of plagiarism constitutes an attempt to deceive in terms of the examination regulations and will lead to the thesis being graded as "failed". This will be communicated to the examination office where your case will be documented. In the event of a serious case of deception the examinee can be generally excluded from any further examination. This can lead to the exmatriculation of the student. Even after completion of the examination procedure and graduation from university, plagiarism can result in a withdrawal of the awarded academic degree.

We confirm that we have read and understood the information in this document. We agree to the outlined procedure for plagiarism assessment and potential sanctioning.

Münster, 20 April 2020

Pedro Henrique Marangoni