INVESTIGATING THE STAR SCHEMA BENCHMARK AS A REPLACEMENT FOR THE TPC-H DECISION SUPPORT SYSTEM

by

Jimi Carmen Sanchez

November, 2016

Director of Thesis: Nasseh Tabrizi, PhD

Major Department: Computer Science

Decision Support System (DSS) are at the core of business intelligence systems. Implementation costs for enterprise level Database Management System (DBMS) and DSS average $10,461 for installation costs. This does not include costs associated with database migrations or testing, which can double the cost, nor does this quoted price include the cost of yearly licensing or support agreements. Depending on the software vendor, there may be additional costs associated with using an application cluster, logical and virtual partitioning, data guards, and even costs per processor core. It is easy to see how the cost of operating a database server can grow expensive rapidly. Information Technology (IT) decision makers and software architects need the ability to choose a DBMS to suit their application's needs. To choose the correct DBMS solution a comprehensive and adaptive benchmark is needed. This benchmark must be capable of predicting how the performance of a given system will scale, as well as offer an estimation of cost. A problematic benchmark that is unable to accurately predict these values is worthless and leads to costly software decision mistakes. To continue to be successful and remain competitive in a given industry it is important for organizations to know their customers, target and acquire new markets, and look to future trends. This is where database business intelligence and decision support systems become useful. DSS allow users to data mine critical information about their work-flows, sales history and trends and have the data readily available so that they may make informed decisions and plan future growth. Business intelligence tools and decision support systems provide executive officers and members

of management, the tools needed to create complex ad-hoc queries and mine important data. Presently, IT decision makers and software engineers use the TPC-H decision support system benchmark as a guide to determining the optimal hardware and database vendor configurations to utilize for their decision support system. The TPC-H benchmark is a popular decision support system benchmark. In recent years, however, TPC-H has become heavily criticized for its many problems. The issues outlined within this thesis can lead IT decision makers to purchase and implement improper hardware and software solutions. This thesis examines the criticisms and issues of the TPC-H benchmark. Utilizing Amazon Web Services cloud computing power, we evaluate the Star Schema Benchmark (SSB), as an alternative to TPC-H. We successfully identify and demonstrate several previously undefined problems in the TPC-H benchmark. Our results conclude that the SSB not only resolves the issues inherent in TPC-H, and should serve as a replacement for TPC-H.

INVESTIGATING THE STAR SCHEMA BENCHMARK AS A REPLACEMENT FOR THE TPC-H DECISION SUPPORT SYSTEM

A Thesis

Presented to The Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Software Engineering

by

Jimi Carmen Sanchez

November, 2016

INVESTIGATING THE STAR SCHEMA BENCHMARK AS A REPLACEMENT FOR THE TPC-H DECISION SUPPORT SYSTEM

by

Jimi Carmen Sanchez

APPROVED BY:

DIRECTOR OF THESIS: _____

Nasseh Tabrizi, PhD

COMMITTEE MEMBER: _____

Mark Hills, PhD

COMMITTEE MEMBER: _____

Junhua Ding, PhD

CHAIR OF THE DEPARTMENT OF COMPUTER SCIENCE: _____

Venkat Gudivada, PhD

DEAN OF THE GRADUATE SCHOOL: _____

Paul J. Gemperline, PhD

ACKNOWLEDGMENT

I wish to express my most sincere gratitude and thanks to my thesis supervisor Dr. Nasseh Tabrizi, as well as my co-advisor Dr. Mark Hills, for helping guide me through the process, providing feedback, encouraging me, and keeping me motivated and in good spirits despite several personal issues that have arisen during my time at ECU. Their patience, motivation, enthusiasm, and immense knowledge have proven to be invaluable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS & TERMS

1NF First Normal Form.

2NF Second Normal Form.

3NF Third Normal Form.

AMI Amazon Machine Image.

AWS Amazon Web Services.

CDO Chief Data Officer.

CEO Chief Executive Officer.

DBMS Database Management System.

DSS Decision Support System.

DW Data Warehouse.

EC2 Elastic Computer Cloud.

I/O Input Output.

IT Information Technology.

NIST National Institute of Standards and Technology.

OLAP Online Analytical Processing.

OLTP Online Transaction Processing.

RDBMS Relational Database Management System.

RTI Research Triangle Institute.

SDL Schema Definition Language.

SF Scale Factor.

SPEC Standard Performance Evaluation Corporation.

SQL Structured Query Language.

SSB Star Schema Benchmark.

TPC Transaction Performance Processing Council.

PREFACE

In this preface I will explain my personal background that led me to take an interest in business intelligence and decision support system software and where my passion for wishing to uncover a perfect DSS benchmark comes from.

In January 2006, we founded HomeInsurance.com (at the time called MyStateInsurance.com), an online insurance call center based in Wilmington, NC where we sold property and casualty (home and automobile) insurance both on the phone and online. In 2007 we purchased the domain name HomeInsurance.com through a private deal with Brad Larson, a GoDaddy broker, and Frank Schill a domainer (domain investor) for $1,000,000 and began operating under the name HomeInsurance.com LLC.

When we started, the company had fifteen employees consisting of nine insurance agents, two sales managers, a call center manager, a web-designer / SEO analyst, Chief Executive Officer (CEO), and a junior developer. We rented a hosted 3rd-party lead management system for insurance agencies. It was a well known solution, but lacked key functionality. Leads had to either be manually entered, or parsed from an email which required a finicky format to be parsed properly. It offered little in the means for integrations with other systems. The sales and marketing reporting provided were inaccurate, non-customizable, and did not provide us with the analytics needed to forecast sales, purchase leads, nor the ability to do skills based routing (routing leads to the agents licensed in those states, or with higher close ratios in certain states, etc.). The system did not offer the ability to integrate with any big-name telephony systems or automated dialers. Although this rented system was sub-par, it served its purpose while we grew our business and designed the ultimate system.

Off the shelf software was not cutting it. We sat through countless demonstrations from software and hardware vendors promising that they could deliver everything we needed and much more. After the millionth failed vendor demonstration, we noticed that each

vendor had their own tricks and wizardry to make their hardware and software solution for business intelligence and decision support systems look amazing. Some utilized hardware with large amounts of ram and disk space, far beyond what would be found in a typical environment. Some used clever indexing techniques. We could not get an honest answer on which solution we should go with, or even how to choose one ourselves. Without a custom software solution tailored to our needs, we would be unable to accomplish the growth we strived for, nor stay competitive in the market.

After researching for a while, I came across the TPC-H benchmark suite which seemed to do exactly what we wanted. It was supposed to replicate a decision support system, be scalable, and provide metrics such as the average cost per hour to run the system as well as the average queries per minute. We enlisted the help of a database administrator to help us analyze the results, and what we found was surprising to say the least. The results of the TPC-H DSS benchmark showed that SQLite would be the fastest, cheapest, and easiest to scale solution. After consulting with several developers of SQLite (which is a public domain license, not open source) we were told that not only would SQLite not be a suitable solution for our scenario, it would not be a good solution for any organization expecting to have a large data set and many transactions being executed against it. If we had taken the results of the many TPC-H DSS we would have mistakenly made an investment in hardware and software to the tune of $50,000 which for a startup company is not pocket change.

Building on my own SQL knowledge (mostly of MySQL and PostgreSQL) and data warehousing concepts, I decided to do further research and arrived at a solution that I felt would allow us to store our archived historical data at an affordable rate, run real time ad-hoc sales reports, and give us the ability to forecast future sales trends, as well as lead trends. My own investigations showed that utilizing MySQL, an open-source enterprise database management system, would allow us to build the DSS of our dreams. Being a start-up, we opted out of the cost of yearly support licensing. The only upfront cost that

we were faced with was hardware (servers, load balancers, database redundancy, etc.), which we rented from RackSpace.com as Amazon Web Services was not in existence at the time.

In 2011, INC. 5000 ranked HomeInsurance.com as one of the top 100 fastest growing countries in America. We also made a transfer-trade with NetQuote (now BankRate) to acquire AutoInsuranceQuotes.com for $750,000.

By March of 2012, we had expanded to two call center locations, fifteen sales managers, and over one hundred and fifty licensed sales ages, and thirty telephone processors. In April of 2012, we sold HomeInsurance.com to Red Ventures, a Charlotte, North Carolina based customer acquisition firm, for a significant amount of money, after being in business for barely six years.

I truly believe that the TPC-H benchmark contains many problems that cause companies to make costly mistakes by trusting the results of the TPC-H performance reports. I narrowly escaped making a mistake that could have caused us to go out of business in short order instead of becoming the success story that we are. My motivation is to expose TPC-H's problems, and recommend the Star Schema Benchmark as an alternative.

Chapter 1: Introduction

Since their inception in the 1970s, database management systems have become increasingly more prevalent in software applications. The cost of storing data has been continuously decreasing [1]. This significant decrease in storage costs has allowed companies to accumulate and aggregate large volumes of data continuously and store it indefinitely. With the continuing growth in the amount of data being collected, application performance has become a growing concern among organizations. As technology evolves and customer demands increase, software requirements are becoming increasingly more demanding. Customer's performance expectations of a software application can be greater than the software's capability. Database benchmarks are a technique to collect performance metrics and identify bottlenecks.

Identifying bottlenecks and problem areas are only one part of the equation. After identification of the bottleneck, action must be taken to resolve the issue. There are many ways to tune a DBMS for optimal performance: Structured Query Language (SQL) query tuning, indexing, caching, data partitioning, and table locking. Query tuning and index optimization are the two most common ways to increase database performance. However, despite these best practices, as will be discussed in Chapter 4, when evaluating hardware and database vendor solutions this is not always an option that is allowed.

As previously mentioned, organizations generate a lot of data. However, these data are not always stored in a single easy to manage location. In fact, it is often stored in different formats across many disparate systems, which can make it difficult to analyze the organization's data and generate meaningful reports. A solution to this problem is to make use of data warehouses and decision support systems which aggregate disparate data into usable formats [2].

Business analytics and business intelligence are important to management as well as marketing professionals. Business intelligence and decision support systems allow users

to answer questions such as: Who are the company's best sales performers? What are some areas we need to improve upon? DSS help turn disorganized data into actual information, improve efficiency, gain sales and market intelligence, which yields competitive intelligence. Studies show that there exists a positive correlation between the usage of decision support systems and corporate performance [3]. But which DSS system should be chosen? Which database vendor should the DSS run on? There are an ample amount of database vendors and software solutions that provide business intelligence and decision support systems. This abundance of software options presents a challenge for IT decision makers. With this abundance of vendors to choose from it can easily become overwhelming for a user to choose the appropriate set of tools for their organization. Database benchmarks such as TPC-H and the Star Schema Benchmark attempt to ease the process of choosing a database system capable of running a decision support system.

The goal of a benchmark is to ease and assist the process of making informed decisions and comparisons between DBMS solutions. Database management systems are at the core business intelligence solutions. The task of evaluating a database system's performance is not a trivial task and can become cumbersome, leading to some IT decision makers blindly choosing a solution and forcing it to fit. Factors such as architecture differences, disk caching, compression choices, and indexing choices all play into a system's overall performance.

As of today, there is only one industry-wide accepted and widely-adopted organization tasked with defining DBMS benchmark standards: Transaction Performance Processing Council (TPC). The TPC has three core benchmarks: TPC-C for measuring Online Transaction Processing (OLTP) performance, TPC-H for ad-hoc decision supporting, and TPC-E to simulate OLTP workloads of a brokerage firm [4]. Another alternative to TPC is Standard Performance Evaluation Corporation (SPEC).

Although the standards developed by TPC have served their purpose over the years, as technology has evolved, the TPC standards have their lost relevance. As a result of

this antiquation, many attributes of the TPC standards are outdated and no longer reflect industry best practices, or real-world scenarios, leaving the IT industry in need of an overhaul and replacement for the TPC-H decision support system benchmark.

Why is database benchmarking important or relevant? There are several reasons for organizations to perform database benchmarking. For a business to become successful and implement their goals, it is imperative that they have a complete and concise understanding of their data. Sales trends and customer demographics are a few metrics that organizations use to predict growth and forecast future sales. Benchmarks are useful for discovering bottlenecks, constraints on concurrent users, and the maximum size of a database. The metrics weigh heavily on the cost needed to implement a scalable system, which organizations must consider carefully when choosing a solution.

Cost savings and operational efficiency are important to businesses implementing database management solutions. Organizational workloads and large volumes of data play an important part in database evaluation decisions. Organizations must determine if a relational database should be implemented, or if a NoSQL key-store could be a better solution. There are many database vendors to choose from. Microsoft SQL Server, MySQL, Oracle, and DB2 make up most database systems used in production enterprise level applications. If an organization is not equipped with the benchmarking tools needed to make an informed decision, costly IT mistakes can occur as a result. In a study conducted by the National Institute of Standards and Technology (NIST) and Research Triangle Institute (RTI), it was shown that software errors and IT mistakes cost United States businesses $59.5 billion dollars every year. While many of these mistakes were unrecoverable, nearly 1/3rd ($22.2 billion) of these software mistakes could have been avoided or caught earlier with improved testing infrastructure and benchmarking tools [5].

Companies that employ decision support systems are five times more likely to make quicker decisions than those organizations who do not utilize DSS [6]. By 2017 it is projected that over 50% of organizations will use business intelligence and decision support

systems to guide the company's directions [6]. By 2019, businesses will spend over $20 Billion dollars per years on DSS systems. It is easy to see how costly a mistake can be by choosing the wrong DSS vendor. It is projected that 90% of organizations hire a Chief Data Officer (CDO) [6].

What makes a database benchmark both worthy and successful? How well a database will perform is directly correlated to the performance of the underlying software algorithms, instead of by raw hardware speed alone as previously thought [7]. Fortunately, four laws were created by Jim Gray to serve as guidelines for creating successful database benchmarks [7]. A benchmark that adheres to these criteria will explicitly define rules pertaining to the execution of these queries and how the rules weigh heavily on the fairness of the tests.

Table 1 depicts Gray's four laws of good characteristics for good benchmarks. These four laws defined by Jim Gray will become useful in Chapter 6 and Chapter 7 as we present our findings against TPC-H and for the Star Schema Benchmark.

Table 1: Gray's four laws of good benchmarks

Relevance	It must measure the peak performance and price/performance of systems when performing typical operations within that problem domain.
Portable	It should be easy to implement the benchmark on different systems and architectures.
Scalability	The benchmark should apply to small and large computer systems. It should be possible to scale the benchmark up to larger systems, and to parallel computer systems as computer performance and architecture evolve.
Simple	It benchmark must be understandable, otherwise it will lack credibility.

Research Contribution: Through empirical techniques this thesis quantitatively shows that the TPC-H decision support system is antiquated and no longer relevant, biased against certain vendors, and has inherent problems that can lead to making costly IT purchasing mistakes. We have also evaluated the Star Schema Benchmark as a replacement

to TPC-H and have shown that it not only addresses and rectifies all the issues defined within this paper, it also simplifies the benchmarking process and provides more freedom to vendors.

This thesis will unfold as follows -

- Chapter 2 will provide the background of DBMSs, and normalization/denormalization.
- Chapter 3 explores relevant related work in detail.
- Chapter 4 will introduce the TPC-H benchmark in detail.
- Chapter 5 will introduce and discuss the Star Schema Benchmark.
- Chapter 6 will discuss the methodology conducted to evaluate the Star Schema Benchmark.
- Chapter 7 will present the results of the experiments and evaluate their significance.
- Chapter 8 will summarize our work and conclude.

Chapter 2: Background

This thesis assumes that the reader has a certain level of familiarity with database concepts and a basic understanding of SQL queries. This chapter also introduces terms, concepts, and theories that the reader may not be familiar with, such as database indexing, set operations, and relational algebra.

2.1 Data Warehouses

What is a data warehouse? First, let's say what a data warehouse is not. A data warehouse is not a product like Oracle or MySQL. It is not a technology, as there are many ways to implement a data warehouse. It is however a database, and much more. Data warehouses provide us with a single version of the truth, performance, simplicity, and data persistence. Data do not originate in a data warehouse. Data are pulled in (integrated) from an organization's source systems such as a CRM, financial system, operation systems, etc. By integrating data from these various sources, the data warehouse becomes a central repository. Not only do internal data sources need to be integrated, but occasionally externally available sources such as fiscal reports from stock charts, or weather reports will require integration. By integrating data from these various sources, the data warehouse becomes a central repository with all the organization's relevant information.

Why use a data warehouse? As mentioned above, we have all the data we need about our customers, human resources, and operational systems. Why add one more piece of technology to the mix? Why would the user or application not go directly to each source of data, collect them and aggregate the data themselves? How are decisions made? An organization's success depends on the cumulative ability to make successful outcomes. Decisions are best made when decision makers best understand the environment that influence that decision's outcome. Relevant and accurate information is the key to providing

them with this information. This is why a data warehouse / decision support system can be one of the most powerful tools at an organization's fingertips.

An organization may have many business systems that track the same information. An organization may have sales information in its CRM system, operation systems, etc. If a user or application asks for information on a list of sales on a specified day, it may be difficult to give an accurate answer if the user is attempting to collect data from manually from many different sources. However, when a user interacts with a data warehouse, all the data have been collected and integrated, and all the business logic is applied when the user attempts to retrieve that information.

Performance can be broken down into two areas. If we go to a system of record, and attempt to run the type of queries that would need to be generated for analytical decision making, it would be trivial to bog this system down, causing issues in production. The reason for this is because system of records is typically built and tuned for transactional operations. A decision maker may wish to see this year's sales by quarter, quarter over quarter, compared to last year, which can include millions of rows and comparisons of data, something a system of record structure would not be well suited for. The business user needs answers to their questions, with the best performance possible. One role of a data warehouse is to deliver a well performing repository where questions are answered in a timely manner.

Applications have back-end databases that are highly normalized for transactional processing, small reads and writes, which is perfect for these transactional operations, but not so for business answers. A data warehouse provides users a simple method to navigate data. Data are organized in such a way that it is intuitive and easy to obtain data answers. In a system of record, a user may be allowed to update information, such as a phone number or an address. The application may or may not store a full history. It may only store the current version. A data warehouse contains a full audit trail for its data. If a user wanted to know information about sales by an area, and only the most recent

address is stored, it is impossible to gain insight because we have lost our historical data. Data warehouses maintains the data persistence at a level the organization needs.

Inmon defined four crucial properties of data warehouses: subject-orientation, integration, time variation, and non-volatile data. These properties of data warehouse can be seen in Table 2. More information on Inmon will be discussed in Section 3.6.

Table 2: Characteristics and properties of data warehouses

Subject-Oriented	Used to analyze a subject area.
Integrated	Integrates data from multiple data sources.
Time-Variant	Historical data are kept in a data warehouse.
Non-volatile	Once data are in the data warehouse, the data will not be changed or altered in any way.

2.1.1 Schema Design

Data modeling and schema design consist of three main stages: conceptual, logical and physical. The complexity of the data model or schema increases significantly as stages are progressed. Since the data model can become complex rapidly, it is important to always start with the conceptual stage of data modeling. By starting with the conceptual stage this helps ensure that the user understands the different entities, relationships, and how they relate to one another at a higher level. In data warehousing projects the conceptual and logical models are occasionally combined into a single model [8].

The conceptual stage models information gathered from organizational business requirements and identifies the highest-level relationships that exists between different entities. Models at the conceptual stage contain only the entities that describe the data and the relationships, meaning that features such as attributes and primary keys are excluded.

The logical design stage looks at the logical relationships among data sets. The data

are arranged into a series of logical relationships by identifying relationships that exist between entities in the data model. This stage is only concerned with high level design [9].

The primary concern of the physical design stage is determining the most efficient method for storing and retrieving data sets. Database systems make use of indices and materialized views to efficiently process complex queries. Determining the best fitting indices and views is a complex modeling problem. Index look-ups and table scans can be effective for data-selective queries, whereas data-intensive queries on the other hand can require costly sequential scans.

2.2 Query Optimization

Accurate estimation is one of the most difficult problems in query optimization. Query optimization attempts to identify the optimal path (measured as minimum execution time) for a given query. Optimization begins by parsing the query and considering possible query plan paths. The results of parsing are passed to the query optimizer. Processing times vary greatly among each possible query path. The purpose of query optimization is to find the path to process a given query in minimum time.

The optimized query path produced by the query optimizer is only an approximation of the query's optimum path. Query optimization has the potential to be a time-consuming task which results in a trade-off between the time needed to calculate the optimum query plan, and the quality of the chosen query path. The optimizer may not always choose the optimum query plan on its own. Each database system has its own method for weighing the optimization paths and determining the ideal solution.

These optimization methods work by associating a cost (weight) with each plan and choosing the plan with the lowest cost. Examples of factors that are weighed when estimating cost are Input Output (I/O) operations, disk buffer space, disk storage service

time, and the cardinality of the data. Possible access paths through primary and secondary indices are examined. Relational table join techniques are analyzed to determine the set of query plans. The order tables are joined weighs heavily on the query plan's performance. The unique paths for each relation are computed and the optimizer logs the optimal method to scan the relation, and the optimal method to produce record sets in a given sort order. The optimizer considers combining each pair of relations for when a join condition exists. The optimizer will consider the available join algorithms for each pair. The cheapest path to join each pair of relations, and sort order is preserved. Since optimization involves estimations, produced query plans may not be the optimal solution. Such plans require manual examination to be tuned for performance.

2.3 Indices

Indexing is a performance enhancement method for sorting records on multiple fields. Indices create an additional data structure which holds the column's value. A pointer to the record is stored [10]. The data structure is sorted which allows algorithms such as binary search to be performed on it. An index reduces or eliminates the need to access the base tables when all projection columns are present in index scans. Searching on a field that is not sorted requires a linear search to be performed. On average, a linear search requires O(n) block accesses, where represents n is the number of blocks a table utilizes. However, if a field is sorted, a binary search may be performed. Binary search has an average case of $O(\log n)$ and a worse-case of $O(n)$ block accesses. Thus, the performance increase is substantial.

There are different types of database indexing architectures and methods that can be performed when creating indices on a database table. Examples of indexing methods include B+-trees and B*-trees, and hashing variants such as linear and spiral hashing. Indices can be clustered, or non-clustered [11]. Indexing is the single most important tool

for performance optimization. Proper indexing requires a knowledge of the business data and the implications of a specific index. The more index records that can fit into a single block of memory, the faster the queries will be. Understanding index layouts is crucial in index and storage engine decisions.

A clustered index alters the way that the data are stored. When a clustered index is created on a column or set of columns, the table data are sorted by that column selection. Clustered indices store rows physically on the disk in the same order as the index [12]. This physical storage of the clustered data means only one clustered index on a table may exist. A clustered index stores close values physically close to one another on the disk. The benefit of storing similar data close to each other is rapid scan and retrieval of records that fall into a range of clustered index values [13].

With non-clustered indices, a second list contains pointers to the physical rows. Unlike clustered indices, it can have many non-clustered indices on a single table [13]. Each additional index will increase the time it takes to write new records. The lowest level of the index contains information that allows the database server to navigate to the data pages it needs. Under certain conditions, the overhead associated with non-clustered indices may be deemed too great by the query optimizer and the database server will resort to a table scan to resolve the query [13].

A covering index incorporates at least all the columns needed for the query execution, without having to perform additional clustered index lookups. Covering indexes have a performance penalty for INSERT and UPDATE operations [13].

2.4 Materialized Views

Views are named SQL queries which provide simple data models as well as implement security constraints. A normal view is a query that defines a virtual table. Virtual tables mean that the data are not sitting in a physical table, instead the data are created ad-hoc at

run-time. A materialized view is a view where the query is executed and the result saved in a physical table. Data in materialized views are precomputed, meaning that the query is not run with each access to the view. The data in a materialized view remains the same until it is manually refreshed [14].

2.5 Table Scans

A full table scan means that the database system must iterate over all the rows of a database table. A table scan will occur when an index does not exist on a column specified in the query's WHERE clause. Every row in the table will be evaluated to see if the WHERE clause's conditions match. Regardless if an index exists on the column(s) matched in the WHERE clause, the database optimizer may choose to execute a full table scan if the overhead for using an index is too high [15].

An example when using an index may take longer than a full table scan is when the number of rows in a table is relatively small. Some comparison operators prevent the use of an index, such as not, not equals, or when a wild-card operator is used on both sides of the operand.

2.6 Operations

Projection

A database projection is an operation of relational algebra. Projection discards those values that are not part of the subset of columns in a relation. One downside to indexing is that these indices require additional space on the disk. Relational algebra allows querying, updating, inserting, and deleting of data, as well as creation and updating of views. Projection eliminates columns [16].

Selection

The select operation selects rows from a table that are based on a predicate. Selection refers to which rows are to be returned. In contrast to PROJECTION, SELECT eliminates rows. Set operations allow the results of multiple queries to be combined into a single result set. The set operators include UNION, INTERSECT, and EXCEPT [16, 17].

JOIN

A JOIN is a binary operator in relational algebra that produces a set of all combinations of tuples from two sets of data based on a defined relationship. There are five types of joins: inner, left, right, full and cross. There are three primary join algorithms: nested loop join, sort-merge join, and hash join [18]. Figure 1 illustrates SQL JOINs as Venn diagrams.

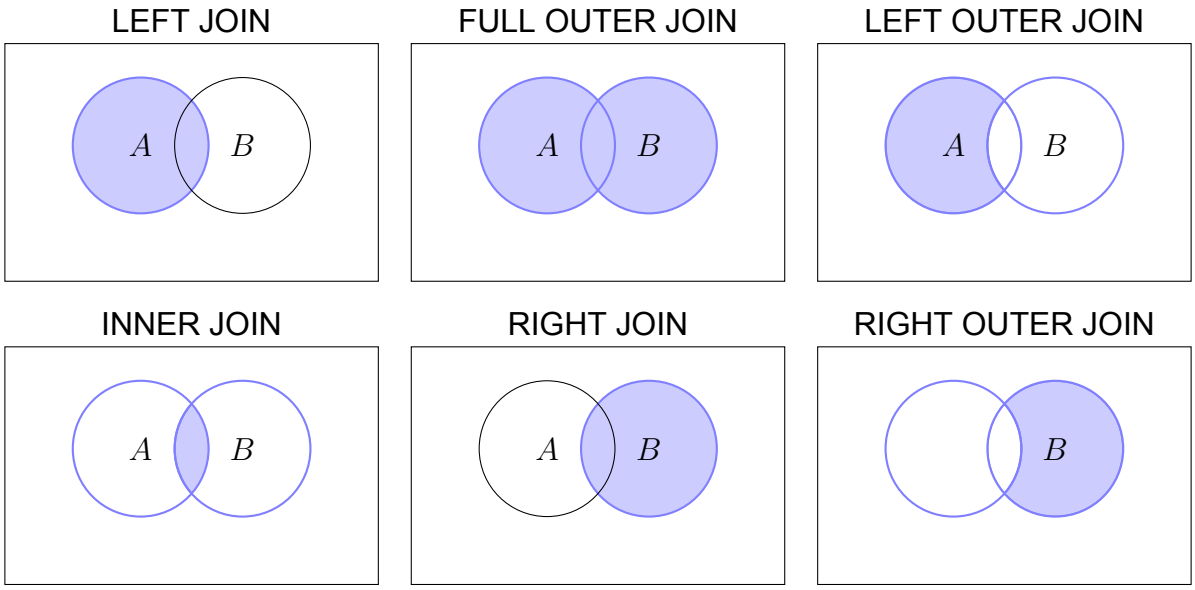

Figure 1: A visual representation of SQL JOINS as Venn diagrams

An inner join is the set of records from set A and set B where the join condition is true. A left join is a set that contains all the records from set A, along with the records from set B which the join condition meets, if any at all. Right joins are sets that contain all the records from set B, along with the records from set A which the join condition meets, if any at all.

A full join is the set of all records from sets A and B regardless of whether the join condition is met or not. A cross join yields a Cartesian product formed from the rows from each table specified in the join criteria. Cartesian products combine each row from the first set with each row from the second set [18]. Nested loop joins use the results from the outer query. More will be discussed in Section 2.10.

2.7 Normalization

Normalization is the process of organizing data in a database, by creating tables and establishing relationships. This eliminates redundancy and inconsistent dependencies within the data. Redundant data wastes disk space and creates maintenance problems. Data that exists in many locations must be changed in the same way in all locations it. There are three main forms of normalization: first, second, and third [19].

Data must be defined which requires looking at the data and data types to be stored. Next the data must be organized into columns, putting related columns into their own database tables. Next, repeating groups of data must be eliminated [20]. Finally, for a schema to be in First Normal Form (1NF), a primary key must be created for every table defined.

For a schema to be in Second Normal Form (2NF), all rules of 1NF must be met. There must not exist any partial dependencies on any of the columns defined as primary keys [20]. 2NF allows data to be narrowed into an easier to access and single purpose source. All non-key columns are dependent on the table's primary key [20].

Tables are in Third Normal Form (3NF) as long as they are also in 2NF, and the columns contained within are non-transitively dependent on the primary key [20]. Transitive simply means that a relationship is equivalent in the middle of a relationship as well as across the whole relationship. So, to be considered non-transitive, all the columns are dependent on the primary a key and no other columns on the table, which comes inherently from 2NF.

2.8 Compression

Abadi et al., have shown the importance of column data compression on performance. They showed that by using column-oriented storage algorithms, in their they reduced the query execution time by 98.18% [21]. The reason that their result showed this reduced query time is compression algorithms perform better on data with low information entropy [21]. In data compression, entropy is the randomness in the data being passed into a compression algorithm. The higher the entropy, the lower the compression ratio. Data becomes harder to compress as the entropy level increases [22]. Disk space is cheap and has continued to become cheaper with each passing year. Compression improves performance as well as reducing used disk space. If data are compressed well, then it will require less time reading data from the disk to memory [21]. Whenever the query optimizer and executor can operate on compressed data directly, then decompression is avoided entirely, which further improves performance, such as when the query executor must perform the same value on multiple columns at once [21]. Abadi et al. have shown that compression usually has a higher impact on performance when the percentage of columns accessed have some order [21].

2.9 Cardinality and Selectivity

Cardinality is the uniqueness of data values contained throughout a table's columns. A higher cardinality means a higher degree of uniqueness in the column's values. A low cardinality means that there are a lot of duplicate values in the data set. Selectivity is calculated using cardinality and can be used to determine how effective an index will be. Query optimizers use the selectivity to determine the best query plan, if it is useful to use a specific index or not. Functional coverage typically is a term used in software testing. However, for the purposes of this thesis functional coverage refers to verifying the

completeness of the typical business work-flows defined within TPC-H and Star Schema Benchmark.

2.10 Subqueries

Subqueries are queries that are nested inside of SQL statements. Subqueries can be nested within the SELECT, WHERE, or HAVING clauses to return data to the outer statement, which can then be used in the outer statement's WHERE clause to restrict the result-set. Subqueries in themselves are not necessarily bad and are often useful. Subqueries become problems and a source of performance suffering when they use data from the outer statement before it can be evaluated. These types of queries are called dependent / correlated subqueries [23]. Correlated subqueries are subqueries that use values from the outer query for its parameters. The subquery is in a nested loop and executed once for each row selected by the outer query. This is an inefficient and expensive method. All correlated subqueries can be optimized by being re-written using JOINs [23, 24, 25, 26, 27, 28].

2.11 Scale Factor

The TPC-H Scale Factor (SF), is a ratio of total storage to database size. SF is used to scale the database workload to mock an application's growth. TPC-H SF will be discussed further in Section 4.5.

Chapter 3: Review of the Literature

3.1 Relational Database Model

The relational database model [29] and the concept of data normalization was developed by Edgar F. Codd. Two methods of data processing emerged from his work; OLTP and Online Analytical Processing (OLAP). OLTP focuses on inserting data, but not necessarily querying that data, whereas OLAP focuses on easily querying and retrieving data. Codd's work went on to form the basis of relational databases and paved the way for all the database vendors we use now.

3.2 The Wisconsin Benchmark

The Wisconsin Benchmark was created based on two main ideas. Benchmark queries should test the performance of the major components. It should be easy to adopt and add new queries as the system evolves [7]. The Wisconsin benchmark gained popularity as it was the first benchmark which contained evaluations of impartial measurements of real products. This sparked competition and wars among the other vendors as customers began recognizing the importance of this benchmark and demanding performance results based upon it. The Wisconsin Benchmark pointed out performance weaknesses of each system. By pointing out these weaknesses, vendors were forced to significantly improve their systems to remain competitive [7].

The Wisconsin Benchmark was designed to allow a range of update and retrieval queries to be executed. The designers used synthetically generated relations in lieu of empirical data [7]. Empirical databases often difficult to scale to scale [7]. Empirical databases contain values that are not flexible enough to permit systematic benchmarking [7]. When joins are utilized it becomes even more difficult to build models that produce

results or relations of a certain size. Empirical data must be analyzed with large amounts of data before it can be determined if data values are randomly distributed. Synthetic databases utilize pseudo-random number generators and data-distribution algorithms to obtain uniformly distributed attribute values while maintaining the relation sizes and scale factor. The Wisconsin Benchmark was designed to have an easy to understand schema and intuitive relations. The results of the benchmark queries are easy to interpret and extending the benchmark query set is simple. The relational attributes were designed to ease the task of controlling selectivity factors such as in SELECT and JOINs.

The Wisconsin Benchmark measured performance of all basic relational operations. It has two operational modes, one that took advantage of a primary, clustered index and a second mode that only allowed access to a secondary, non-clustered index. When no indices were created, both modes operated the same. It contained 32 queries. The size of the relations must be at least a factor of five times larger than the main memory buffer space available. Elapsed time was used as the performance measurement. The decision to use elapsed time versus other metrics was because of variation and unpredictability among different operating systems and a different database system running on the same operating system (such as CPU time, disk I/O operations performed). Original versions of the Wisconsin Benchmark did not incorporate operating cost, because development and testing originally occurred on the same hardware systems.

One criticism of the Wisconsin benchmark is that it runs in a single-user mode. The developers of the Wisconsin benchmark began to develop a multi-user version. However, by the time the multi-user version of the benchmark was complete, other benchmarks began to be adopted instead. Technical disagreements led to competing multi-user benchmarks being created. Both failed to gain popularity or adaptation. The developers of the Wisconsin Benchmark speculated that the reason for this failure was that neither reduced each system to a single number which resulted in it being difficult to compare two systems. The Wisconsin Benchmark created a competition war among vendors. The two off-shoots did

not initiate the same competition war, as vendors only considered the multi-user performance of a single system, meaning they could to ignore the rules because they did not have a reason to keep the war going. [7]

3.3 AS^{3}AP

ANSI SQL Standard SQL and Portable Benchmark (AS^{3}AP) [30] was designed to provide a comprehensive set of tests to measure database processing power. AS^{3}AP was designed to be portable meaning the tests could be executed across a wide range of operating systems and architectures. AS^{3}AP provided the equivalent database ratio metric which allowed for straightforward and non-ambiguous interpretations of the results. AS^{3}AP developed a maximum SF, where the system can execute a series of multi-user tests a given time frame. The database size, and scale factor can be used as a performance metric. AS^{3}AP may be used for comparing cost and performance [30]. They calculated the equivalent ratios for two systems being tested as the ratio of their equivalent database sizes [30]. Current relational database systems have varying degrees of functionality, capabilities, performance, and cost. Defining meaningful metrics of database processing power can be a difficult task [30]. Most database benchmarks have a major fault as they provide no useful guidelines for fixing or improving the system. These issues were addressed in AS^{3}AP by emphasizing scalability, portability, and ease of use and interpretation [30].

AS^{3}AP tests are divided into single user tests, and multi-user tests. Single user tests include utilities for testing load and structuring the database, use queries designed to test access methods and basic query optimization.

Multi-user tests model different types of database workloads such as OLTP workloads, information retrievals, and mixed workloads such as relational scans and report queries.

Elapsed time within a 12-hour window is the only measurement captured by the AS^{3}AP

benchmark. Other metrics such as CPU and I/O utilization have to be collected for an in-depth analysis for DBMS performance. These additional measurements are not specified as part of the AS^{3}AP metrics definitions [30].

AS^{3}AP queries are precompiled, except for the multi-user cross section queries. There are many simplifications with the benchmark. These simplifications allow the benchmark to be installed and run with ease. Modules are grouped according based on the functionality needing to be tested. Depending on the needs of the user and what needs to be tested, the benchmark can be run in its entirety, or only the modules needing testing. The database is not corrupted because of the process of interleaving special queries that save deleted tuples, reinsert the deleted tuples, and restore updated tuples to their original values, so long as the user runs the complete suite in the specified order [31]. DBGEN was used to generate the test database [31].

3.4 Measuring Transaction Processing Power

Tandem Computers [32] defined three benchmarks called Sort, Scan, and DebitCredit. DebitCredit went on to be implemented in TPC-A. They developed a method for measuring transaction processing power. Performance can be difficult to measure, and not all measures are appropriate for every application domain. CPU power measures typically do not account for parallel processing systems that can utilize multiple processors, or multiple cores. In these scenarios only cost and throughput are meaningful metrics. Historically I/O metrics have been ignored. Tandem Computers showed, however, that I/O has a direct correlation to performance and should be calculated and considered a meaningful metric.

3.5 Separating the database from the application hosting environment

A comprehensive set of tests [33] to study the advantages of using back-end database machine architectures against using conventional computer database systems were developed. Database machines are specialized software and hardware configurations dedicated to managing database systems. Database systems are typically processor-bound and not IO-bound, which results in a performance decrease on a host system. The test results concluded that hosted back-end database machine architecture yield superior performance in most cases. By hosting the back-end separate from the application level, it is possible to offload a majority of the database processing activity which releases system resources. The trade-off in this performance gain is increased cost due to additional hardware [33].

3.6 Data Warehousing

The term data warehouse [34] was Bill Inmon in 1990. Data warehouses aid analysts in making informed decisions within an organization. A data warehouse focuses on change over time. Data stored in data warehouses are nonvolatile. Nonvolatile is the concept that once data is entered into the warehouse, it should not change. Data warehouses enable the user to analyze what has occurred, and define data by subject matter rather than the organization's ongoing operations.

Data warehousing focuses on modeling and analysis of data for decision-making. Data warehouses aggregate data from disparate sources into a consistent format. As mentioned in Section 2.1, data warehouses are often tightly coupled with OLAP transactions, to allow for mining of knowledge at deeper levels. This is why data warehouses are impor-

tant for data analysis and online analytical processing. Data warehouses keep data separate from the organization's operational database. Frequent updating is not performed in a data warehouse.

Data warehouses aid decision makers in organizing their internal and external data to make strategic decisions. Data warehouses consolidate historical data analysis. Operational databases allow read and write operations on data, whereas OLAP queries only need data read access.

Operational databases maintain current data, while data warehouses maintain historical data. Data warehouses, helps business executives organize, analyze data for decision making [34]. Analytical processing can be split up into: slice-and-dice, drill down, drill-up, and pivoting. Data mining assists knowledge insight by uncovering hidden patterns and associations, constructing models, and performing classification.

Since the main goal of data warehouse systems is to make information access as easy as possible, the data must be obvious and intuitive to work with. This information must be consistent and credible, and should be scrubbed and its quality assurance verified before being inserted into the system. The system must serve as the ultimate authority on the data used for decision making [10]. At the core of a data warehouse are fact tables. Fact tables store various performance measurements from the organization's business processes. Each row in a fact table represents a measurement event. The grain of a fact table specifies the level of detail. All rows in a fact table must have the same grain, which ensures errors in the data such as double-counting are avoided. Of the three categories of grain (transaction, periodic snapshot, accumulating snapshot) transaction level granularity is the most common. The lowest level of data captured in a business process is referred to as the atomic grain [10].

Ralph Kimball is the original architect of data warehousing and is the father of the modern data warehouse [10]. Per Ralph Kimball, the TPC-H schema should be denormalized into a single SALES table. Denormalization attempts to optimize database read perfor-

mance by adding redundant data or by grouping data. Denormalization is frequently used as a technique for addressing performance and scalability and allows us to avoid excessive joins. Kimball believed that a star schema helps to reduce the number of complex and unnecessary joins [10].

The star schema is the simplest data mart schema and consists of one or more fact tables. Fact tables reference dimension tables. Star schemata are effective for handling both simple data marts and large data warehouses. Kimball's methodology for defining and designing data warehouse schema is the most frequently used method [10]. This method is popular because analytical users can begin to see results and query against their data quicker, as the model does not require a master plan to begin designing. However, this method does have risks associated with duplicate data and re-work in the future due to the lack of design process in the beginning stages. Star schemata were developed by Red Brick Systems (now IBM) to speed up queries on data loaded from operational databases at some given intervals [35].

Ralph Kimball favors a data warehouse design approach utilizing a bottom-up process in which dimensional data marts are identified and created for reporting and analytical purposes. Kimball's methodology for defining and designing data warehouse schema is the most frequently used method [10].

Kimball founded Red Brick Systems. Red Brick Systems is a Relational Database Management System (RDBMS) heavily optimized for data warehousing. IBM has acquired Red Brick Systems. Per IBM "IBM Red Brick Warehouse is a client/server decision-support RDBMS for information systems (IS) and business managers who want to improve the quality and performance of their decision-support applications. The superior performance of IBM Red Brick Warehouse is based on: Indexing and joining technologies designed to accelerate the retrieval of database information [10]." The relevance of this quote will be demonstrated in Chapter 4 and Chapter 6.

OLAP systems deal with analytical business tasks. They cope with large volumes of

data and are expected to have short response times. OLAP databases tend to be optimized for querying and reporting which is contrary to OLTP. OLAP data are derived, aggregated, and structured from historical data into sophisticated structures allowing data analysis. OLAP can be used for data-mining and finding relationships and correlations. OLAP focuses on analyzing data coming in about the business while OLTP runs the business. Table 3 lists the key differences between Online Analytical Processing and Online Transactional processing.

Table 3: Comparison of OLAP and OLTP

OLAP	**OLTP**
Analyzes the business	Run the business
Information out	Data in
Star schema	Entity relationship model
Historical data	Current data
Summarized consolidated data	Primitive and highly detailed data
Summarized multidimensional view	Detailed and flat relational view

DSS are tools to support the decision making process. DSS are typically used by users in management, operations, and planning roles. They serve as a framework for analyzing business data, and presenting data in a human-readable format, allowing the user to make business decisions more easily. DSS are only as good as the underlying DBMS providing data. DSS provide functionality for data storage and retrieval and assist in model building and model-based decisions [4]. This thesis, when referring to DSS, mean the OLAP-DSS hybrid.

3.7 Development of a TPC-H Tool-kit

In [36], the authors realized that there were no publicly available tool-kits for TPC-H. People needed a system that was easily deploy-able, could support many different

commercial (R)DBMSs, was open source, and did not require an advanced degree to figure out how to use it. [36] developed a toolkit that provided scripts, tuning parameters, DBMS drivers, query template generation, and verification scripts. The toolkit showed promising results, however as mentioned in [37], the fast-changing technology landscape makes it difficult for tools like this to get adopted and gain popularity. By the time that they start to get noticed and gain interest they are already outdated.

Over the years there have been many attempts to develop a set of benchmarks that could yield meaningful metrics [4, 38, 39, 40, 41].

3.8 Development of an Isolation Layer

Building upon Kimball's work, [40] developed and implemented a method for abstracting the OLAP data from the physical structure. One problem which was identified with the traditional architecture method lies in the way in which data are viewed and manipulated at the individual level. However, this level is strictly dependent on the implementation at the data warehouse level. Views are refreshed when changes to the organization occur [40]. The interoperability between different data warehouses must be solved case by case. This new layer provided an intermediate layer to isolate queries from the physical details.

3.9 A New Approach for Profiling Data Warehouse Benchmarks

A new proposal [41] for a data warehouse design process that attempted to solve some of the problems that are present in the Inmon and Kimball models was presented in 2002. The new approach relied on the assumption that some facts are asked more frequently than others. The authors advocated that some dimensional attributes are rarely used while others are frequently referenced, and that some facts and dimension attributes are

used together, while some will almost never be used together. They also stated that some mathematical operations would be frequent while others will not. They concluded that typically, data performance considerations are introduced in later stages of data warehouse design process [41]. As a result of this late consideration, nearly all enhancements are applied after the physical schema has been obtained. They proposed that data warehouse design needs to consider business and performance considerations from the beginning. They also proposed that perhaps combining smaller dimension or fact tables, or splitting bigger tables into smaller tables, can serve for performance gains. These conclusions have been adopted and incorporated into the Star Schema Benchmark.

3.10 Benchmarking Open Source Database Management Systems

Research [38] was conducted in comparing proprietary database management systems' performance against open source database management systems [38]. This researched compared six different database management systems; Sybase, PostgreSQL, Oracle, MySQL, Firebird, and DB2. The author developed a benchmark from scratch as well as the tools to automate running it using Perl. At the time the research was conducted there were significant issues with the Perl DBI module when running transactions against Oracle servers. The DBI module was unable to use persistent connections which meant that the overhead of re-connecting to the database server was added to every SQL query request. These limitations could have significantly skewed the benchmark results in regards to the data collected for the Oracle database. This methodology has had little to no support from the community, possibly because the queries and test suite found within are not comprehensive. The lack of research for Microsoft SQL Server, which is a popular DBMS, has potentially contributed to the research being ignore.

3.11 Scaling Down TPC-H

DBmbench [12] was created to scale down the TPC-H benchmark using a systematic scaling framework tailored to serve for micro-architecture research [12]. DBmbench works by scaling down the existing benchmark suite and optimizing the relational query operations. These results showed that it was possible to scale down TPC-H and still have it be a meaningful benchmark, but the applications of these improvements are only directly beneficial in micro-architecture environments and applications which violate Gray's law of scalability.

Building [42] on the work of [43], Vandierendonck presented a method for reducing the number of queries needed in TPC-H. By creating a new process for the selection of cluster representatives they were able to reduce TPC-H from twenty-two to six queries, which yielded a 60% reduction in execution time. The authors then validated their method by comparing case studies to their findings to show that the subsets were appropriated. Basing quantitative decisions on the TPC-H subset lead to the same design decisions being made as when based on the full TPC-H suite. While these efforts may have been a step in the right direction, this TPC-H subset benchmark still suffers from the issues outlined in the introduction chapter.

In [44] a theory is presented behind transaction management, storage structures, concurrency control and availability, and how these affected performance and scalability of the database management systems. [44] developed a benchmark based on frequently occurring activities in the telecommunications industry. This resulted in a highly specialized benchmark not useful for applications outside of that industry. However, [44] was helpful in showing the work a single individual could accomplish, as most benchmarks today are developed by large organizations such as the TPC. All [44] tests were performed on a single platform and operating system, as an attempt at reducing the number of external variables. [44]'s benchmark suite simulated an electronic commerce application, with the

goal of providing a more realistic load to the databases tested. The benchmark suite used general terms, which could easily be applied to a wide range of industries. These choices were made to improve upon previous efforts at database benchmarks without making the system overly complex. Despite [44]'s advancements in performance comparisons among open source database systems, [44] did not discuss the witnessed variance of the results, nor did he compare the results of one database system's score to another in a statistically significant way. Both [38, 44] made a mistake trying to develop a completely new benchmark, and instead should have focused on making improvements to the existing TPC-H benchmark.

3.12 Set Query Benchmark

The Set Query Benchmark [45] was built upon the work of the Wisconsin benchmark [7]. Set queries are queries that consider data from multiple table rows, whereas TPC and DebitCredit only deal with row-at-a-time updates, which fail to meet the criteria for being a DSS. Set Query was designed with the goal to become portable, provide functional coverage, and selectivity coverage, and have scalability [45]. Set Query Benchmark is designed to evaluate the performance of OLAP-type applications.

Set Query generates uniform random values for column data. It also only allowed a single user stream at a time, which was a major disadvantaged compared to TPC-H which allows multiple concurrent streams. Set Query helped pave the way for the Star Schema Benchmark, as a true OLAP/data-warehousing benchmark. The Set Query Benchmark showed that computer resource usage can be high, with varying performance variations among different products, which highlights how critical performance issues can be. The Set Query Benchmark was created to assist information systems managers' gaining insight on performance on their data and strategic data applications. The Set Query benchmark differs from the Wisconsin Benchmark and AS^{3}AP in that it primarily focuses on

these intensive set queries. Document search, direct marketing, and decision support are three operations explored within the Set Query Benchmark, as they are the most common activities found in commercial applications [45].

The Set Query benchmark also allows customization of the price and performance rating variables, and the weighing of individual queries relative to their expected prevalence, a key feature that is missing in TPC. After heavy analysis, several query types were chosen that covered the various workload scenarios. The authors of the Set Query benchmark also contacted leading decision support manufactures and asked them to evaluate the query work-flows, and the results were that the Set Query benchmark captured nearly all decision support work-flows perfectly. TPC is not modeled after real-world scenarios nor were software manufacturers consulted while TPC was being developed.

Chapter 4: TPC-H

Presently, there is one widely accepted standards organization whose purpose is to develop database benchmarking standards. The TPC is a non-profit corporation founded to define transaction processing benchmarks and database benchmarks [46]. The TPC-H standard schema has design choices and restrictions imposed that do not adhere to Ralph Kimball's view of a data warehouse, nor Codd's definition of a third normal form schema [10].

TPC-H is a database decision support system benchmark. It models the activity of a product supplying business. TPC-H is broken down into two sets of tests, loading the database with data, and then measuring the system performance during some workload. During the performance testing, one run occurs to measure the power being used, and another to measure the throughput. The power test measures the raw query execution power of the system by a single user in a single session. The throughput test is a metric of the system's ability to process the most queries in the least amount of time.

4.1 Metrics

TPC-H provides three types of timing measurements: database load time, measurement interval, and timing intervals. The measurement interval is defined as the total time needed to execute the throughput test. "Timing intervals are the execution times for each query or refresh function." QphH@size is the query per hour performance metric, and is supposed to weigh evenly "the contribution of the single user power metric and the multi-user throughput metric [46]." Per-price QphH@Size is a price/performance comparison between the two systems.

4.2 Schema

The TPC-H schema consists of eight separate and individual tables; PART, PART-SUPP, LINEITEM, ORDERS, SUPPLIERS, NATION, CUSTOMER, and REGION as depicted in Figure 2 . TPC-H's schema is designed to represent a simple data warehouse that holds facts about customer, sales, and part suppliers. The TPC-H schema is not the way in which the real-world is modeled. The ORDERS and LINEITEM tables are normalized in TPC-H [10]. Per [10], in a Kimball data warehouse, LINEITEM AND ORDERS would be denormalized into a single SALES table. NATION would be denormalized into tables holding data about regions, customers, and suppliers.

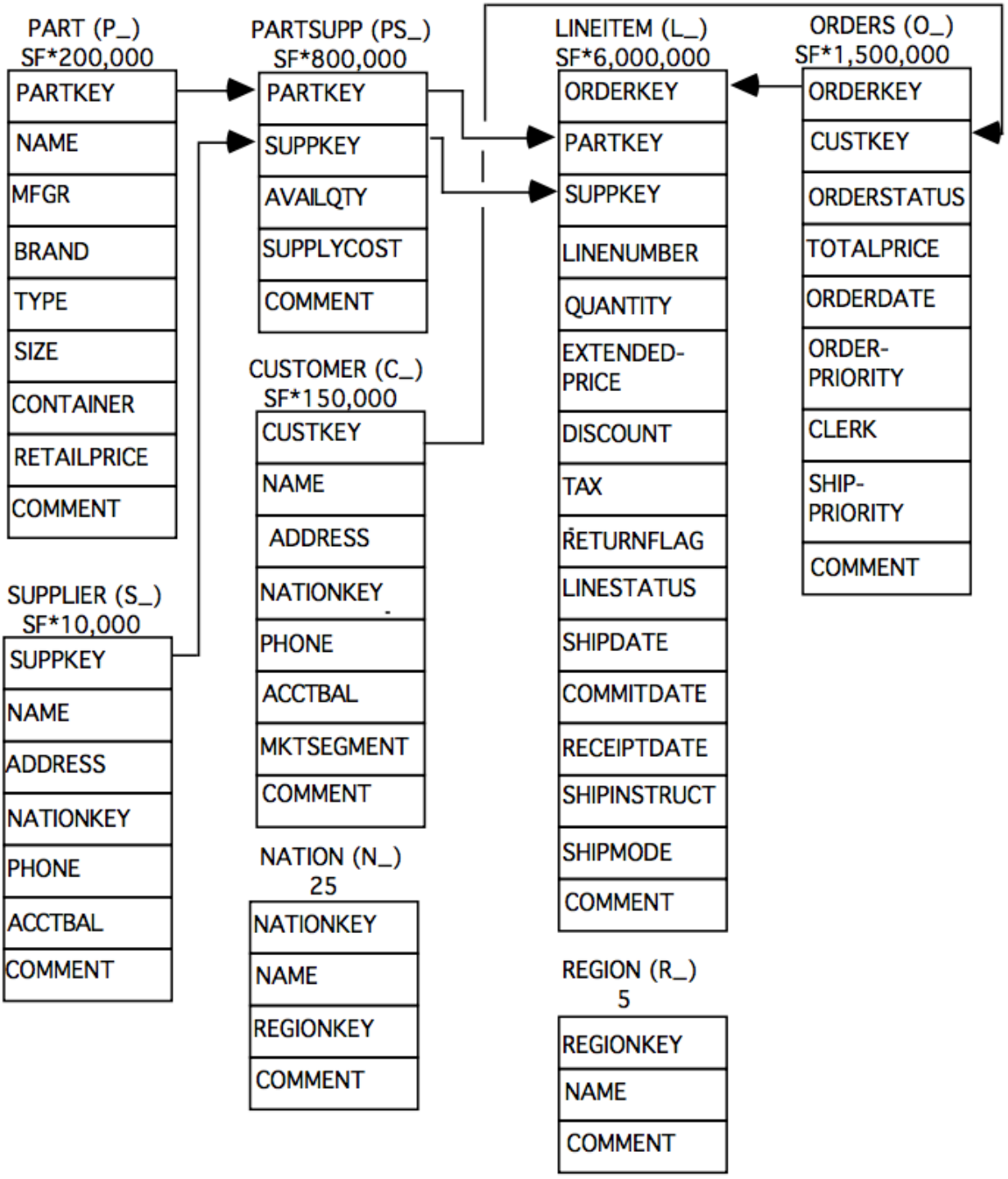

Figure 2: TPC-H schema

Ad-hoc queries are used to simulate some decision support activity occurring in conjunction with operational transactions. Since the queries must be fair to all vendors, they

must be truly ad-hoc and they must be defined in advance. This foreknowledge presents the vendors with some interesting tuning opportunities. To prevent cheating, TPC-H defines strict rules about the partitioning and indexing strategies allowed. Allowing tricks like materialized views (caching of earlier queries), would make it simple to fine tune the database workload. However, in data warehouses, the ability to partition data is a necessity, and disabling query caching can negate the effects of materialized views anyhow. Partitioning optimizes hardware performance by bypassing table scans and instead jumping directly to the data being queried.

4.3 Indexing

As shown in Table 4 the rules for allowed indexing and partitioning methods are defined in TPC-H. Clauses 1.4.2.2, 1.4.2.3 and 1.5.4 [46] define the index and partition rules than vendors may implement.

Table 4: TPC-H indexing rules

Rule	Clause
Primary keys may be indexed	1.4.2
Foreign keys may be indexed	1.4.2.3
Partitions are allowed on any table, on only one column of data type DATE.	1.5.4
The usage of JOIN indices is prohibited	5.2.7
Materialized/indexed views are disallowed	1.5.7
The use of computed columns is not allowed	1.5.7
Vertical partitioning is disallowed	1.5.5
Indices on more than one column are prohibited (Exception: LINEITEM (L_ORDERKEY, L_LINENUMBER)	1.4.2.2

TPC-H does not define the underlying index type, meaning that vendors are free to choose the index types that they wish to implement such as a B-tree, a Hash, General-

ized Search Tree (GiST), etc. The only exception that is imposed is that the index type may not be explicitly defined within the Data Definition Language (DDL). DDL is used to define data structures in SQL systems. For instance, in the MySQL RDBMS, the server will automatically create indices on columns that are declared as foreign keys. If consistency is followed, this practice is not prohibited. As can be seen in Table 4, many performance enhancing tricks are explicitly disallowed (join indexes, materialized/indexed views, etc.). The only exception is the composite primary key in LINEITEM (L_ORDERKEY, L_LINENUMBER), and any index on a non-key or foreign key column. Tables can be indexed heavily without significant increases in storage overhead; however, this is something that is explicitly disallowed by TPC-H.

4.4 Data Distribution

Section 4.2.3 of TPC-H describes the data distribution. The two tables, LINEITEM and ORDERS are roughly 80% of the entire data-set [46]. Per Kimball, these tables would typically make up a single fact table [10]. Data warehouses should represent real-world scenarios. The data distributions in TPC-H are uniform and there are five regions and five nations per region. Customers and suppliers per nation are constant [47]. TPC-H is a random uncorrelated data set. This means that any compression that is gained is not reflective of real-world scenarios.

4.5 Scaling

The scale factor determines the ratio at which the data loaded into a database. “Scale factor is used to increase the size of the database throughout the benchmarking process [46].” When the scale factor is increased, the number of rows added to each table is increased. TPC-H allows for the following scale factors: 1, 10, 30, 100, 300, 1000, 3000,

10000, 30000, and 100000. Table 5 illustrates how the TPC-H benchmark table size at a scale factor of 1, which is equivalent of 1 gigabyte (GB) of disk space. Some tables such as NATION and REGION have a fixed length cardinality and do not grow with the scale factor.

Table 5: TPC-H estimated database size at scale factor 1

Table Name	Cardinality (rows)	Length (bytes)	Table Size (MB)
SUPPLIER	10,000	159	2
PART	200,000	155	30
PARTSUPP	800,000	144	110
CUSTOMER	150,000	179	26
ORDERS	1,500,000	104	149
LINEITEM	6,001,215	112	641
NATION	25	128	<1
REGION	5	124	<1

4.6 Criticism

The TPC organization is responsible for defining benchmarking standards and collecting vendor supplied results [46]. TPC-H is a standard to serve as an ad hoc decision support benchmark [48]. TPC-H has been heavily criticized for not strictly adhering to the principles of Ralph Kimball's model of data marts "not allowing freedom in indexing and partitioning [18, 49, 48]." Kimball argues that a traditional 3NF approach to decision support systems is not suitable due to poor query performance and usability issues. Kimball supported the idea that a decision support system should use star schema [10].

TPC-H is not representative of a decision support system and as a result, [50] proposed

a set of modifications to TPC-H which they call the Star Schema Benchmark. Star Schema Benchmark re-implements the logical data of TPC-H in a traditional star schema. The Star Schema Benchmark was designed to test star schema optimization and to address the problems outlined in TPC-H. The Star Schema Benchmark is significantly based on the TPC-H benchmark with modifications to improve upon it. The Star Schema Benchmark implements a star-schema. The The Star Schema Benchmark allows column and table compression. The Star Schema Benchmark is a simple DSS benchmark consisting of four query flights and a simple roll-up style hierarchy [50]. The Star Schema Benchmark will be discussed in further detail in Chapter 5.

TPC-H has complex and unnecessary joins and a pseudo 3NF schema despite the fact data warehouses should use a star schema [51]. Per Stonebraker, TPC-H was cleverly constructed to avoid using a star schema, so that materialized views are rendered unproductive [51]. The PARTSUPP table is used as an OLTP table, not used in OLAP or querying. PARTSUPP lists suppliers and parts to give answers on SUPPLYCOST and AVAILQTY. However, there are seven years of orders, and as orders are filled, business users and the application need to know how many parts are currently available, so this becomes meaningless. AVAILQTY and SUPPLYCOST are never refreshed in TPC-H which means they have the wrong temporal grain.

The Star Schema Benchmark measures performance of DBMS against a traditional data warehouse schema. The Star Schema Benchmark implements the same logical data in a traditional star schema. TPC-H models the data in pseudo 3NF schema [4]. The Star Schema Benchmark queries are simplified versions of the queries defined in TPC-H, organized into four query flights [52].

TPC-H has been criticized as being “complicated to set-up and use and that running the benchmarks takes a substantial amount of time [42].” In DBmBench the authors said that the TPC-H benchmark fell short of being effective. They stated that the TPC-H benchmark is too complex, and the configuration is too large and precludes its usage [12]. They

also criticized the TPC-H benchmark for being designed to test functionality and evaluate performance on real hardware. However these tests required orders of magnitude larger execution times for use in both simulation and virtual environments [12].

The vendors and members that make up TPC are free to publish their results; or in the case of negative results not publish the results. Michael Majdalany an administrator at TPC, defended this saying “This is a voluntary organization, so we can’t force people to publish benchmarks...The majority of vendors in the market do participate and publish benchmarks...[but] some established vendors don’t feel the need to publish benchmarks because their attitude is that their customers known them already [53].”

Customers want references from companies that fit their description and that have similar business challenges. Doug Henschen of InformationWeek criticized the TPC and its vendor reports as being irrelevant, as the vendors used the reports to claim to be X times faster than their competitor [39]. There have also been accusations that vendors are publishing their own skewed results to make themselves look better. Curt Monash performed an investigation of TPC vendor reports and found that most TPC benchmarks are being run on absurdly unrealistic hardware configurations [54]. Monash found that the TPC-H benchmark can be greatly influenced by the hardware it’s used on. This influence is far more than by the DBMS it’s testing [54]. Monash later went on to criticize ParAccel’s published findings, stating:

> “Monash cited a result from ParAccel, where ParAccel performed a thirty terabyte benchmark on 43 nodes, each with 64 gigabytes of RAM and 24 terabytes of disk. That’s 961,124.9 gigabytes of disk, officially, for a 32:1 disk/data ratio. By way of contrast, real-life analytic DBMS with good compression often have disk/data ratios of well under 1:1… Meanwhile, the RAM: data ratio is around 1:11. It’s clear that ParAccel’s early TPC-H benchmarks ran entirely in RAM [54].”

Despite the problems in TPC, IT decision makers tend to turn to TPC because their

results are widely available and up to date. Until recently, the importance of set query functionality had not been known and not implemented properly in benchmarks.

Chapter 5: Star Schema Benchmark

5.1 Schema

Several schema modifications were made to the TPC-H to translate TPC-H into a more efficient star schema form. TPC-H tables LINEITEM and ORDERS were combined into LINEORDER, a single fact sales table. This model consistently adheres to the Kimball model [10, 50]. “The LINEORDER table is a 17-column table with information about individual orders, with a composite primary key across the ORDERKEY and the LINENUMBER attributes [55].” Other attributes on the LINEORDER table include the foreign key references to the CUSTOMER, PART, SUPPLIER, and DATE tables, as well as attributes for each order including priority, quantity, price, and discounts applied. “This simplifies the schema considerably, both for writing queries and computing queries as the two largest tables of TPC-H are not pre-joined [52].”

Figure 3 illustrates the Star Schema Benchmark design. The arrows from the outer dimensional tables point towards the inner fact table.

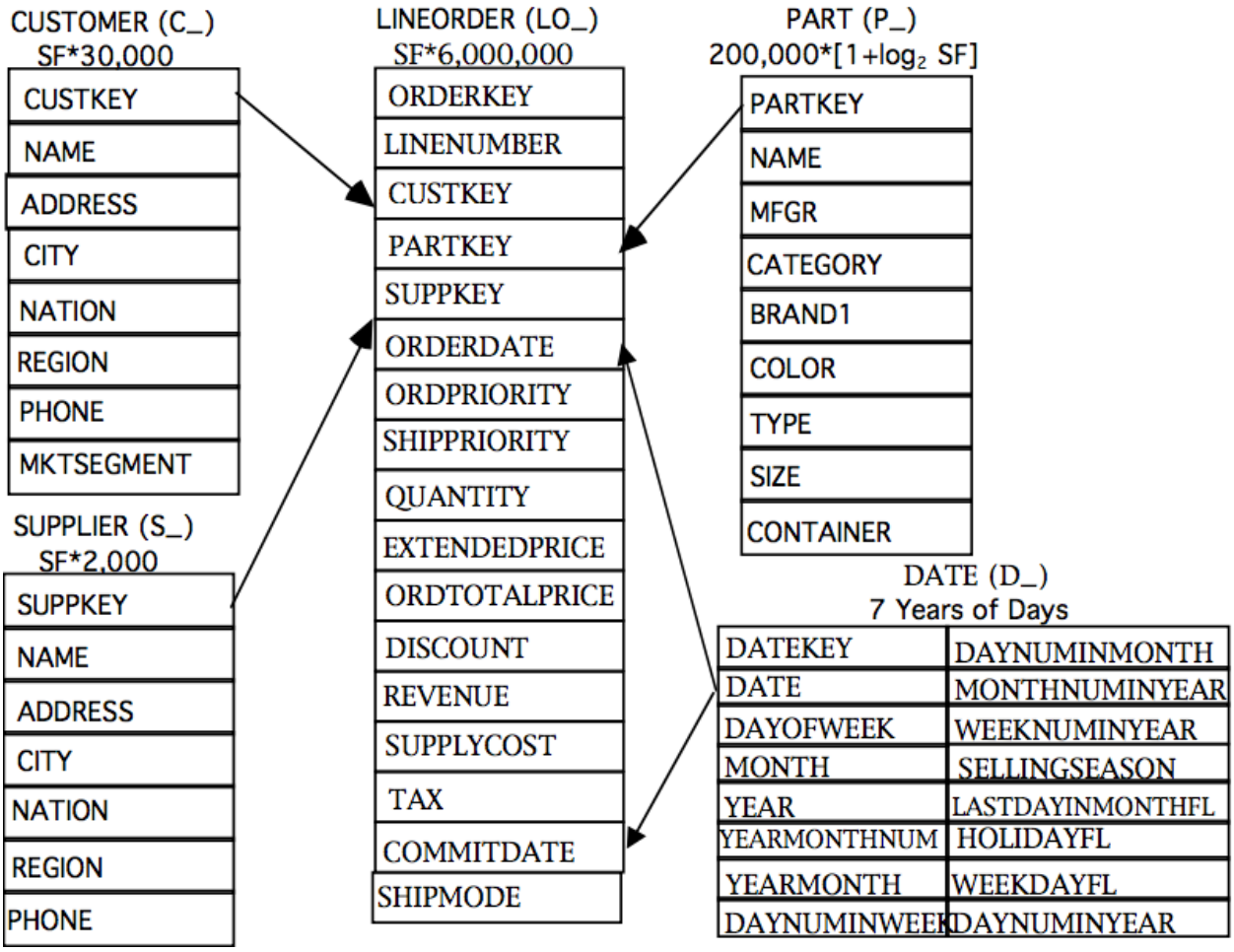

Figure 3: Star Schema Benchmark schema

The PARTSUPP table is removed since "it would belong to a different data mart than the ORDERS and LINEITEM data [50]." The reason that PARTSUPP is dropped is because it contains varying temporal data.

Temporal data are data that varies over time. Typically, databases have limited capacity to carry temporal information as they store current data snapshots [56]. However, in real-world applications, there are definite needs for both current and historical data. One example of this need is applications existing in the financial sector. Temporal databases efficiently store a time series of data, usually by having some fixed timescale and then storing only changes in the measured data. The grain is defined to mean exactly what one fact table record represents [10]. Temporal is the frequency in which the data are

tracked.

The LINEITEMS comment attributes, orders, and shipping instructions are dropped. The reason for this is “because a warehouse does not store such information in a fact table...They cannot be aggregated and take significant storage [50].”

A table named DATE was created to serve as a dimensional table. However, DATE is a reserved word in most DBMS [57]. If a better table name had been chosen, it would be easier to avoid SQL errors, or having to wrap the table name in back-tick identifiers. Dimensional tables were created for CUSTOMER, PART, SUPPLIER, and DATE. Tables to encompass SHIPDATE, RECEIPTDATE, and RETURNFLAG, should be added. However, O’Neil notated that to accomplish this would be “too complicated a schema for our simple star schema benchmark [50].” Since the Star Schema Benchmark prohibits self-joins or sub-queries referencing LINEORDER, Star Schema Benchmark mainly focuses on queries that reference LINEORDER only once. [50]. Star Schema Benchmark tries to support the spirit of the queries that appear in TPC-H [50]. Star Schema Benchmark contains LINEORDER, a single fact table. It also breaks up the TPC-H schema into four dimension tables (CUSTOMER, DATE, PART, SUPPLIER). “It is common practice to combine LINEITEM and ORDER in TPC-H to get LINEORDER in Star Schema Benchmark. LINEORDER represents one row for each one in LINEITEM [50].”

As seen in Figure 3, the PARTSUPP table was removed to adhere to Kimball’s principles [50]. LINEITEM and ORDERS are composed of fine temporal grain which causes an issue, as PARTSUPP is composed with a periodic snapshot grain [50]. Kimball defines Periodic snapshot grains as “fact tables that summarize many measurement events occurring over some period, such as a day, a week, or a month [10].” Because of the difference in temporal grain between PARTSUPP and LINEORDER, problems arise with PS_SUPPLYCOST as the data would not remain constant for past years [50].

Rows in PARTSUPP are not augmented when rows are added or augmented to the LINEORDER table as PARTSUPP is frozen in time. If PARTSUPP and LINEORDER are

treated as separate fact tables. This would allow isolating queries and forgoing the joins altogether together [50]. Per O'Neil, "This is done in all but one of the queries where PART-SUPP is in the WHERE clause (Q1, Q11, Q16, Q20) but not in Q9, where PARTSUPP, ORDERS, and LINEITEM all appear Q9 is intended to find, for each nation and year, the profits for certain parts ordered that year [50]." One criticism is that it is more than likely that the PS_AVAILQTY would not have remained constant during all these past years. "One reason for having the PARTSUPP is to break up what might be a star schema and so that query plans do not appear to be too simple [50]."

"The presence of a Snapshot PARTSUPP table in this design seems suspicious anyway, as if placed there to require a non-trivial normalized join schema [50]." O'Neil noted that in the TPC-H benchmark the column "PS_AVAILQTY is never updated, not even during the refresh that inserts new rows into the ORDERS table [58]." In the Star Schema Benchmark data warehouse, it is more acceptable to drop the PARTSUPP table, replacing it with a new column SUPPLYCOST for each LINEORDER Fact row [58]. Since data warehouses contain only derived data, "there is no reason to normalize to guarantee one fact in one place [58]."

It is expected that subsequent orders for the same PART and SUPPLIER may repeat this SUPPLYCOST. If the last PART of some type were to be modified or deleted, a result in the loss of the original price charged may occur. Star Schema Benchmark adds the LO_PROFIT column to LINEORDER. This additional column allows for simpler queries, and a decrease in query execution time [50].

The LINEITEM and ORDERS table were merged into a single sales fact table called LINEORDER, reducing the need for many complex joins. "All columns in ORDERS and LINEITEMS that make us wait to insert a Fact row after an order is placed on ORDERDATE is dropped [58]." One example given is that business decision makers do not want to wait "until we know when the order is shipped, when it is received, and whether it is returned before we can query the existence of an order [58]."

Per Rabl, NATION and REGION were denormalized into the Customer and Supplier tables with a city column added to both tables [52]. This simplifies the schema considerably. This eases writing and computing queries, seeing as the largest two tables of TPC-H are pre-joined [58]. “Queries do not have to perform the join and users writing queries against the schema do not have to express the join in their queries [59].” The NATION table and REGION table may be considered appropriate in an OLTP system to enforce integrity. However, in a data warehouse system, the data are cleaned prior to being loaded. Dimension tables are not so limited in space usage as are the fact tables [60]. NATION and REGION are added to the ADDRESS columns [50].

5.1.1 Differences from TPC-H

LINEITEM and ORDER are combined to make a LINEORDER table, which eliminates the need for many complex joins [58]. “The grain is the business definition of what a single fact table record represents [10].” The PARTSUPP table of TPC-H has a grain mismatch, and was removed from Star Schema Benchmark. P_RETAILPRICE was dropped because the retail price is likely to change, often changing “too frequently to be held in a dimension [10].” P_NAME was changed from being a 55-byte character column, to a 22-byte character column. In TPC-H, P_NAME is populated by concatenating five different colors together. “It is assumed that this 55-byte length was intended to make the PART table larger and more difficult to query [50].” Star Schema Benchmark consists of fewer queries than TPC-H and has more relaxed requirements on which configurations of tuning are allowed and those which are forbidden. It is easier to implement.

In addition to dropping the column P_RETAILPRICE, the columns C_ACCTBAL and P_COMMENT and O_COMMENT were also dropped. P_COMMENT and O_COMMENT are unparsed comment text and as such have no business in a data warehouse query [50]. TPC-H P_BRAND has 25 distinct values which is small for a set of brands for a data warehouse [50]. P_BRAND is replaced with P_BRAND1 which holds 1000 values, subdi-

viding each P_CATEGORY into forty values. Star Schema Benchmark introduces a DATE dimension table, which is standard for a sales data warehouse [10]. The DATE dimension table providers numerous value attributes for querying such as DAYOFWEEK, MONTH, and SELLINGSEASON. These schema modifications result in a proper star schema data mart where the LINEORDER table serves as the middle and has dimension tables CUSTOMER, PART, SUPPLIER, and DATE.

5.2 Queries

There are multiple reasons for abandoning the TPC-H queries. Many TPC-H queries do not translate directly into Star Schema Benchmark. Only a few TPC-H queries can be implemented in Star Schema Benchmark with only minimal modification. The queries are constructed to cover the range of tasks performed by an important set of Star Schema queries [58]. This assists users in deriving a performance rating from the weighted data subset in which they expect to use in production [58]. "It is difficult to provide true functional coverage with a small number of queries, but Star Schema Benchmark at least tries to provide queries that have up to four dimensional restrictions [58]." Star Schema Benchmark attempts to vary the selectivity, especially when many fact table rows are retrieved. Star Schema Benchmark attempts to add both functional and selectivity coverage [58].

5.2.1 Query Flights

The Star Schema Benchmark contains four flights of queries whereas TPC-H contains twenty-two queries. Each flight query is made up of three to four queries. Each query has varying selectivity. Per Rabl, Star Schema Benchmark "introduces selectivity hierarchies in all dimension tables [52]." Query flights are modified TPC-H query sets, modified for variation of selectivity. A flight is made up of a set of queries that would possibly be needed, such as a drill down [4]. These query flights and their descriptions are depicted in Table 6.

Table 6: Star Schema Benchmark query flights

Query Flight	Description
Q1	Calculates revenue increase by year as a result of removing certain discount codes in a range of products.
Q2	Compares revenue for certain products, suppliers in a defined region, aggregated by product and order year.
Q3	Produces a report of revenue based on transactions from the LINEORDER table, grouped by the customer's nation and supplier, limited by region and over some defined time period.
Q4	The most complex flight. It joins all the tables together to simulate drilling down into region and manufacturer. Retrieves aggregate fit grouped by year and customer nation.

5.2.2 Caching

Query caching, and overlap of data being accessed between Q1 and Q2 in the Star Schema Benchmark reduces the number of disk accesses necessary [58]. Attempts to minimize the effects of overlap are in place, however in some situations it may become necessary to introduce steps to reduce caching effects of one query on another [50].

5.2.3 Compression

The Star Schema Benchmark provides freedom in the compression of column values, so long as the data-set retrieved contains equal values specified in the Schema Definition Language (SDL) [50]. The importance of data compression and how it relates to database storage is explained in Section 2.8.

5.3 Data Distribution

The Star Schema Benchmark's data are uniformly distributed like that of TPC-H. Selectivity hierarchies are present in all dimension tables like the manufacturer/brand hierarchy present in the TPC-H benchmark [52]. SSB-DBGEN is a tool like TPC-H DBGEN. SSB-DBGEN helps create uniformly distributed data and assists in seeding the database, which decreases the time between transactions and each test suite [61]. However, adaption to different data distributions is not easy with SSB-DBGEN since the meta-data and actual data generation implementations are not segmented [52].

5.4 Scaling & Metrics

Like the TPC-H benchmark, the Star Schema Benchmark generates data at different scales factors. Unlike TPC-H, Star Schema Benchmark data are generated proportionally to its scale factor. The PARTS table scales logarithmically not linearly [50]. In addition to the metrics included in the TPC-H benchmark, the Star Schema Benchmark also measures performance in the areas listed below.

1. The memory space.
2. Processor model.
3. Number of processors.
4. Breakdown of schema by:
    (a) Processor
    (b) Disk setup.
    (c) Other parameters of the system that interfere with performance

Chapter 6: Evaluation of the Star Schema Benchmark and the TPC-H Benchmark

We chose two methodologies for evaluating the Star Schema Benchmark as a replacement decision support system benchmark for TPC-H. The first method involved mapping SSB queries to their TPC-H counterparts. The average execution times on different database vendors at varying scale factors were recorded per query, unmodified and again with indices, added as this technique would be allowed in a traditional data warehouse. The average execution time does not provide us with an apples-to-apples comparison as the schemata and queries are greatly different. However, the average execution times can be used in conjunction with the query mapping to determine work-flow and work-load coverage. They can be used to compare average suite execution times between two systems to see if there is a trade off on work-flow coverage and average suite execution time. Mapping allows us to determine work-flow and work-load coverage for relevance, which is used in our second method covered within Section 6.2.

We wanted to answer the question Which benchmark adhered closest to Gray's Laws? Our second method involved comparing Gray's Laws of Good Benchmarking characteristics, as defined in Table 1, against TPC-H and Star Schema Benchmark. Each characteristic was given a weight. Each benchmark was then evaluated and scored.

6.1 Mapping

6.1.1 Environment

We utilized Amazon Web Services (AWS) Elastic Computer Cloud (EC2) servers. Linux CentOS 7 64-bit Amazon Machine Image (AMI) was used for the host operating system. Linux was chosen as it does not require licensing to use, is open-source, widely supported, and allowed us to run our chosen DBMSs as they are platform independent.

We chose these DBMS vendors as they are open source and platform independent, and offer community editions that do not require licensing. Each database vendor was installed using the CentOS package management system using the default configuration operations.

To install MySQL 5.7 and PostgreSQL 9.6 server on CentOS 7, the user is required to add the development repository for community release editions, since RHEL/CentOS tend to run a few versions behind. Figures 4 to 6 depict the installation process used to install each of these DBMSs. To provide fairness, we did not tune any of the vendor's configuration files, leaving them as stock configurations.

Installation instructions for installing MySQL community server edition can be seen in Figure 4.

```
$ sudo rpm -Uvh http://dev.mysql.com/get/mysql-community-release
    -el7-5.noarch.rpm
$ sudo yum -y install mysql-community-server
```

Figure 4: MySQL Server 5.7 installation instructions on CentOS 7

PostgreSQL sever installation instructions can be seen in Figure 5.

```
$ rpm -Uvh https://download.postgresql.org/pub/repos/yum/9.6/
    redhat/rhel-7-x86_64/pgdg-centos96-9.6-3.noarch.rpm
$ sudo yum -y install postgresql-server
```

Figure 5: PostgreSQL server installation instructions on CentOS 7

SQLite installation instructions can be seen in Figure 6.

```
$ sudo yum -y install sqlite
```

Figure 6: SQLite server installation instructions on CentOS 7

The specification of our EC2 hosting environments can be seen in Table 7. AWS offers several tiers of EC2, we chose to utilize tier 2 micro, small, and medium sized.

Table 7: Experiment execution environment

EC2 Type	CPU	GB RAM	Operating System
t2.micro	1	1	Linux
t2.small	1	2	Linux
t2.medium	2	4	Linux

6.1.2 Query Mapping

Ten of the twenty-two TPC-H queries were mapped to the closest matching SSB query flight and query. Some TPC-H queries have direct mappings to SSB queries. Queries which did not have a direct mapping. Those unmatched queries were paired based on the query's business logic and the complexity of the query. For each DBMS, two databases were created; one database to store the TPC-H schema, and the other to store the SSB schema. Next, DBGEN and SSB-DBGEN were utilized to seed the databases with data

using a SF of 1. The ten query pairs and their randomized parameters generated by the QGEN utilities were executed one hundred times and the results recorded. This process was repeated using a scale factor of 10.

As a control, the schemata of each benchmark were not altered, except for the primary key. The mapping of TPC-H to SSB queries can be seen in Table 8. Although TPC-H Q3 does not map to a specific query flight, it was chosen as it has similar business logic as well as similar query structures, and query complexity to SSB Q2.

Table 8: TPC-H to Star Schema Benchmark query mapping

Query	TPC-H	SSB
1	Q6	Q1.1
2	Q6	Q1.2
3	Q6	Q1.3
4	Q3	Q2.1
5	Q3	Q2.2
6	Q3	Q2.3
7	Q5	Q3.1
8	Q5	Q3.2
9	Q5	Q3.3
10	Q5	Q3.4

1. TPC-H Q6 & SSB [Q1.1, Q1.2, Q1.3] Forecasts revenue change at various levels of temporal grain.

2. TPC-H Q3 & SSB [Q2.1, Q2.2, Q2.3] Deals with shipping priority, at different levels of granularity on order date and shipping date. Compares revenues for product classes in a region, by class and year of order.

3. TPC-H Q5 & SSB [Q3.1, Q3.2, Q3.3, Q3.4] Total revenue for transaction during time period by customer nation, supplier nation, and year and local supplier volume.

The output from the EXPLAIN, as well as the WHERE clauses from each of the ten queries in both TPC-H and SSB were analyzed. From this analysis, we determined which

columns would benefit from indexing, and created indices on these columns. After applying the indices to the columns determined above, the experiments were re-run and the data collected.

We added indices on 20 columns to 5 tables on the Star Schema Benchmark schemata, which can be seen in Table 9. Every column referenced in the WHERE clause of the Star Schema Benchmark query flights was included resulting in 100% index coverage. We added 18 indices to columns across 8 tables to the TPC-H schemata as represented in Table 10. Like the Star Schema Benchmark indexing choices, every column specified in the WHERE clause was covered. However, 15 of the 18 (83.33%) indices were only used for one query compared to the Star Schema Benchmark where 33.33% of the indices were used once.

Table 9: SSB query flight indices

Table	Column	Queries
CUSTOMER	C_CITY	3.3, 3.4
CUSTOMER	C_CUSTKEY	3.2, 3.3,3.4
CUSTOMER	C_NATION	3.2
DATE	D_DATEKEY	1.2, 1.3, 2.1, 2.2, 2.3, 3.3, 3.4
DATE	D_WEEKNUMINYEAR	1.3
DATE	D_YEAR	1.1, 1.3, 3.1, 3.2, 3.3
DATE	D_YEARMONTH	3.4
DATE	D_YEARMONTHNUM	1.2, 3.4
LINEORDER	LO_CUSTKEY	3.1, 3.2, 3.3, 3.4
LINEORDER	LO_DISCOUNT	1.1, 1.2, 1.3
LINEORDER	LO_ORDERDATE	All
LINEORDER	LO_PARTKEY	2.3
LINEORDER	LO_QUANTITY	1.1, 1.2,
LINEORDER	LO_SUPPKEY	2.2, 3.1, 3.2, 3.3, 3.4
PART	P_BRAND1	2.2, 2.3
PART	P_CATEGORY	2.1
PART	P_PARTKEY	2.1, 2.2, 2.3
SUPPLIER	S_CITY	3.3, 3.4
SUPPLIER	S_REGION	2.1, 2.2, 2.3, 3.1
SUPPLIER	S_SUPPKEY	2.1, 2.2, 2.3, 3.1, 3.2, 3.3, 3.4

Table 10: TPC-H indexing

Table	Column	Queries
CUSTOMER	C_MKTSEGMENT	Q3
LINEITEM	C_CUSTKEY	Q3, Q5
LINEITEM	L_DISCOUNT	Q6
LINEITEM	L_ORDERKEY	Q3,Q5
LINEITEM	L_QUANTITY	Q6
LINEITEM	L_SHIPDATE	Q3, Q5, Q6
NATION	N_NATIONKEY	Q2
NATION	N_REGIONKEY	Q2
ORDERS	O_ORDERDATE	Q3
PART	P_PARTKEY	Q2
PART	P_SIZE	Q2
PART	P_TYPE	Q2
PART	PS_SUPPKEY	Q2
PARTSUPP	PS_SUPPYCOST	Q2
PARTSUPP	PS_PARTKEY	Q2
REGION	R_REGIONKEY	Q2
SUPPLIER	S_SUPPKEY	Q2
SUPPLIER	S_NATIONKEY	Q2

6.2 Comparison of the Star Schema Benchmark and TPC-H

Next we evaluated both TPC-H and SSB against Gray's Four Laws of Good Benchmarks. The results of this comparison can be seen in Table 14. We weighted each of law and gave scores accordingly. The scores of each category as well as the overall ranking can be seen in Table 11.

We ranked Relevance the highest because a DSS that does not model real-world scenarios are useless. Scalability was ranked second highest as a DSS benchmark must be able to accurately predict how the system will perform now and as it grows, allowing for the correct decisions to be made now, instead of costly decisions later. Portability was ranked lower since if a DSS benchmark is based around ANSI SQL then it should not be difficult to move from one DBMS vendor to another. Simplicity was ranked the lowest, per Gray's definition simplicity refers to the output of the results. Given that SSB is a derivative of TPC-H and has similar metrics, we felt this rank definition did not apply. Instead we modified this definition to be a measure of simplicity in terms of design, business coverage, and number of queries during execution.

Table 11: Decision support system characteristics ranking

Point	Weight %
Relevance	40
Portability	20
Scalability	30
Simplicity	10

Chapter 7: Results

7.1 Query Mapping

Our tests for TPC-H showed that with a scale factor of 1 and the default out of the box schema configuration that MySQL had an average execution time of 21.7 seconds, PostgreSQL 20.0 seconds, and SQLite 18.76 seconds. After applying the indexing to the scale factor 1 TPC-H schema, the averages dropped to 16.275 seconds, 16.4 seconds, and 17.971 seconds respectively. It is interesting to note that the average execution time did not translate at the same scale from the first test to the second. This means that some DBMS can use indexes more efficiently than others.

Figure 7 show the average execution time of TPC-H SF 1 without modifications. SQLite was the fastest DBMS in 9 out of 10 queries, only being beaten in Q6 by MySQL.

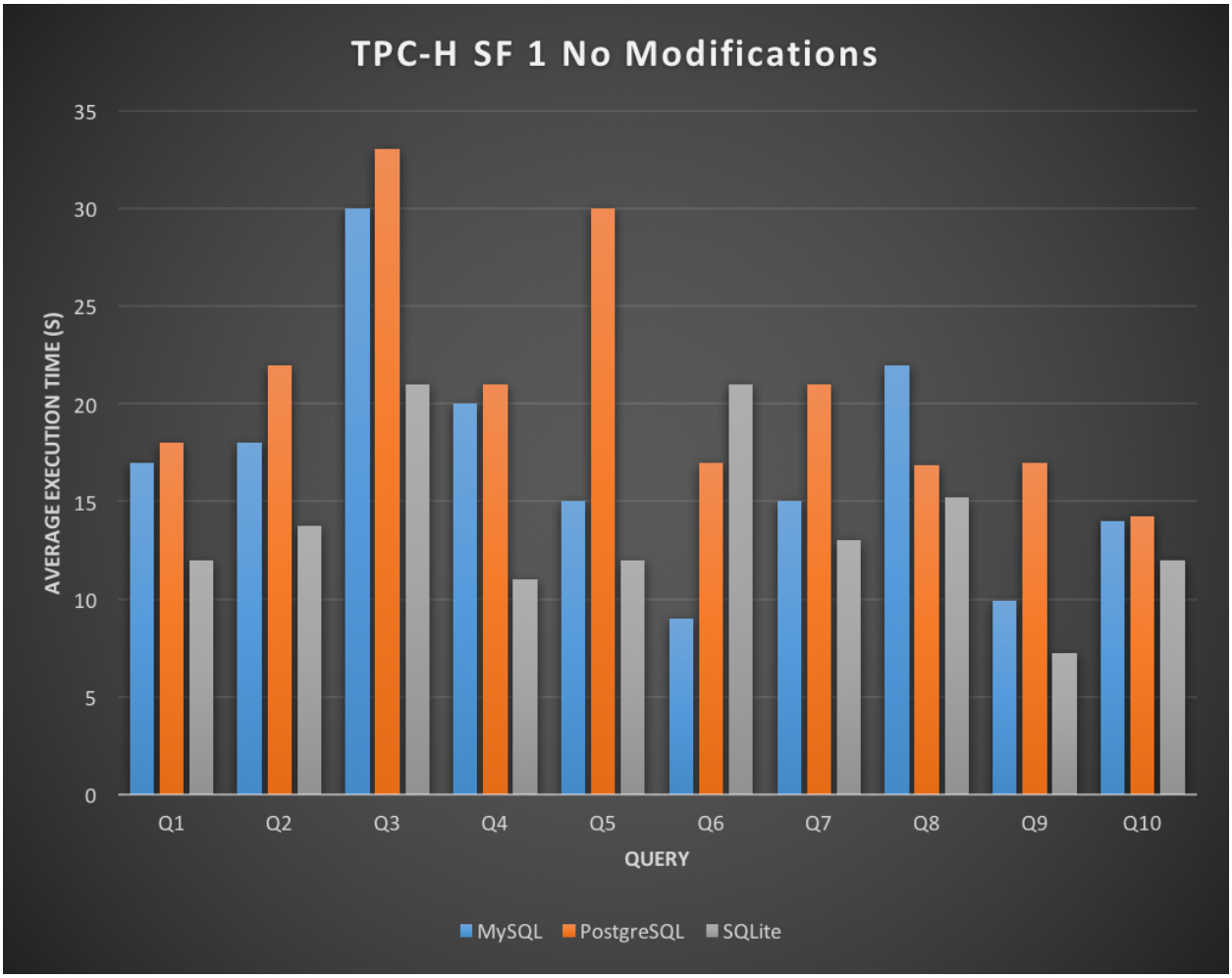

Figure 7: TPC-H average query execution time at scale factor 1 without modifications

After applying indices to the schemata there was a decrease across the board in average query execution time (see Figure 8). SQLite continued to be the fastest performing DBMS, having the fastest average execution time in 9 out of 10 queries. MySQL remained faster than PostgreSQL in 9 out of 10 queries. The distance between average execution times amongst MySQL and PostgreSQL increased significantly after adding indices.

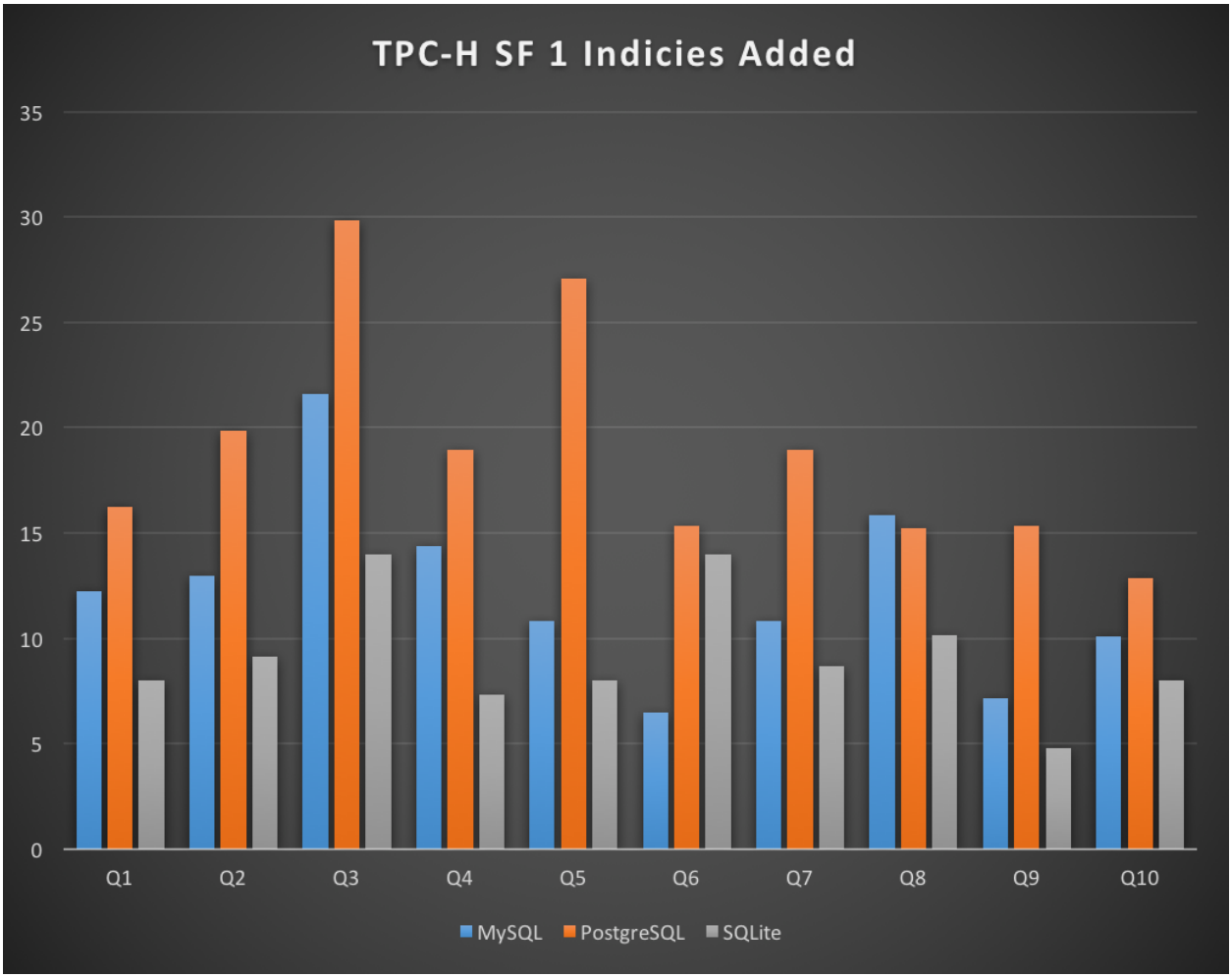

Figure 8: TPC-H average query execution at scale factor 1 after applying indices

At SF 10 for TPC-H without schema modifications SQLite was the top performing DBMS in 5 queries, MySQL 4 queries, and PostgreSQL 1 query (see Figure 9). This result shows that from SF 1 to SF 10, SQLite remained the top performing DBMS in terms of average query execution time. However, these results also show that the number of queries that SQLite was top performing in dropped from 9/10 to 5/10. This drop in the number of winning queries is in line with what would be expected per the SQLite documentation [62].

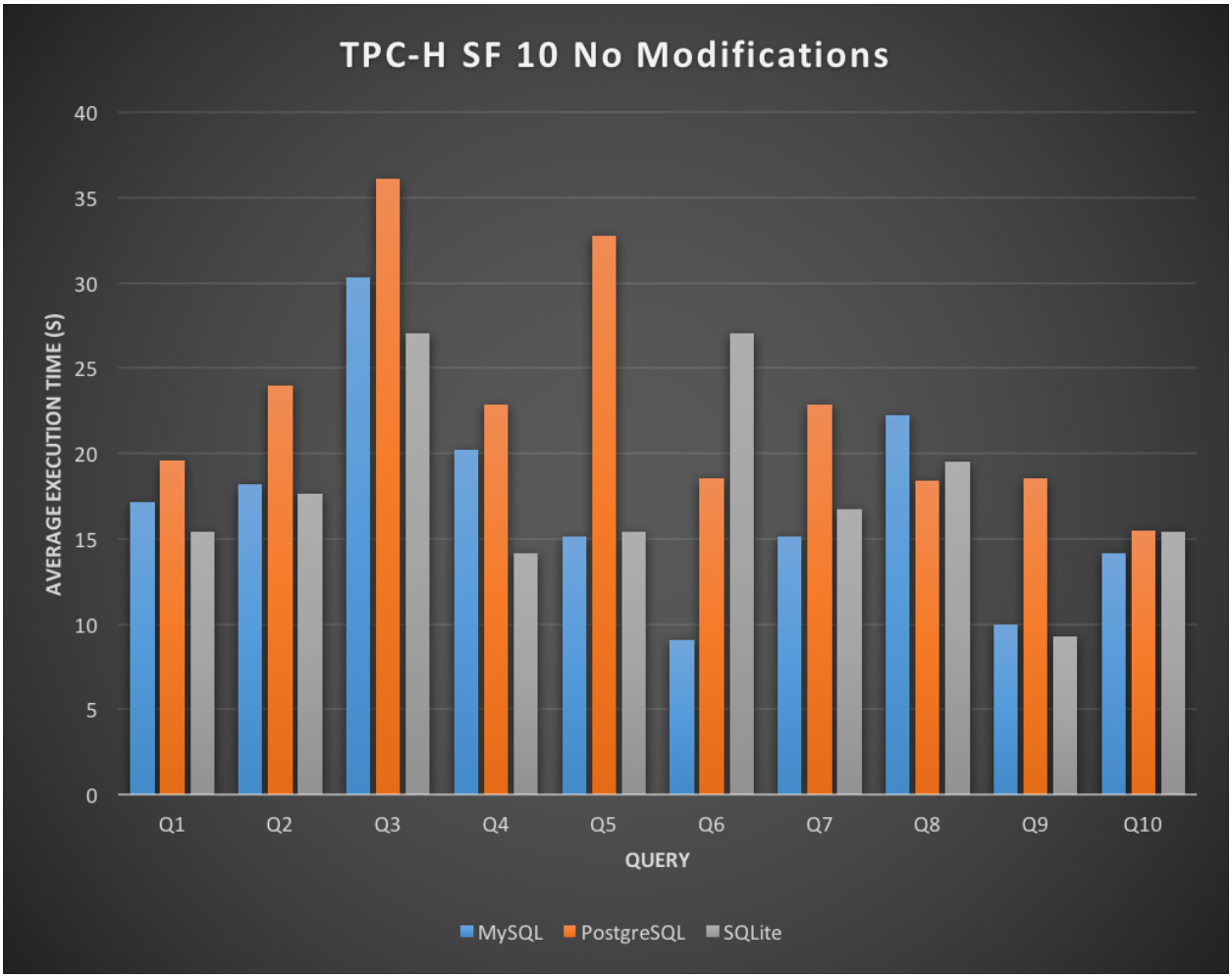

Figure 9: TPC-H average query execution time at scale factor 10

As can be seen in Figure 10, after applying indices to the TPC-H SF 10 tables, SQLite transitioned from being the top performing DBMS to the worst performing. If the benchmark results of TPC-H at scale factors 1 and 10 without index modifications were used to choose a DBMS based on speed, SQLite would be the clear top performing DBMS. This further illustrates the importance of allowing indexing and partitioning in decision support system benchmarks.

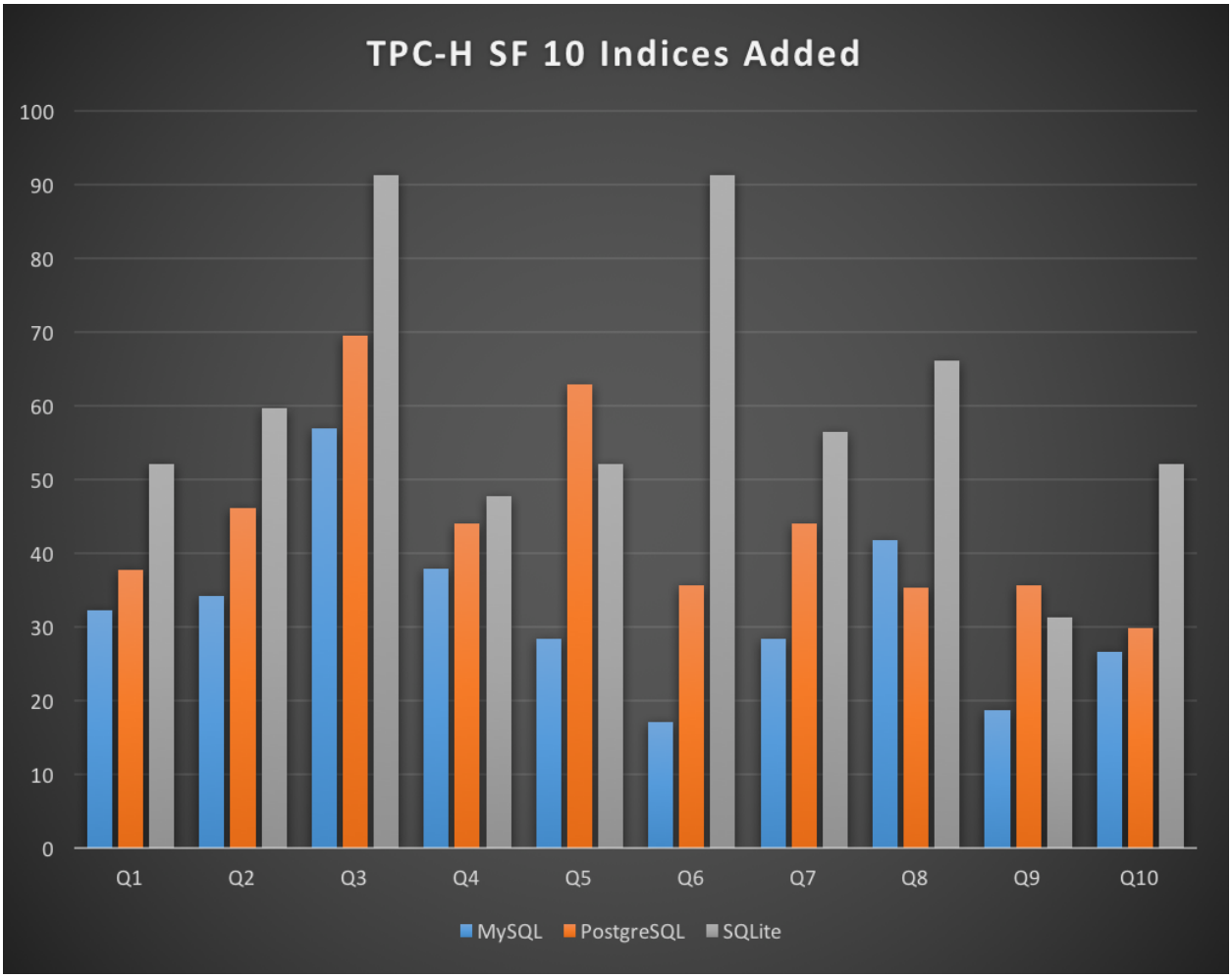

Figure 10: TPC-H indices average query execution time at scale factor 10

The TPC-H results showed that SQLite was the top performing DBMS, based on average query execution time. Choosing SQLite for an enterprise DSS/Data Warehouse (DW) would be a mistake. Per SQLite's own documentation it is often better to choose a RDBMS system for client/server applications, high-volume websites, large data-sets, or high concurrency. SQLite suggests that if the application's data are separated from the application by a network, then SQLite is not a good solution [62]. As mentioned in [33], there are many advantages to housing an application's database on separate hardware than the application is running on, which also serves as a strike against SQLite.

Our SSB tests showed that with a scale factor of 1 and the default out of the box schema

configuration that MySQL had an average execution time of 19.8 seconds, PostgreSQL 16.2 seconds, and SQLite 16.2 seconds.

Figure 11 shows the average execution time of our Star Schema Benchmark test at scale factor 1. In contrast to our TPC-H SF 1 test, MySQL emerged as the fastest DBMS, beating PostgreSQL and SQLite.

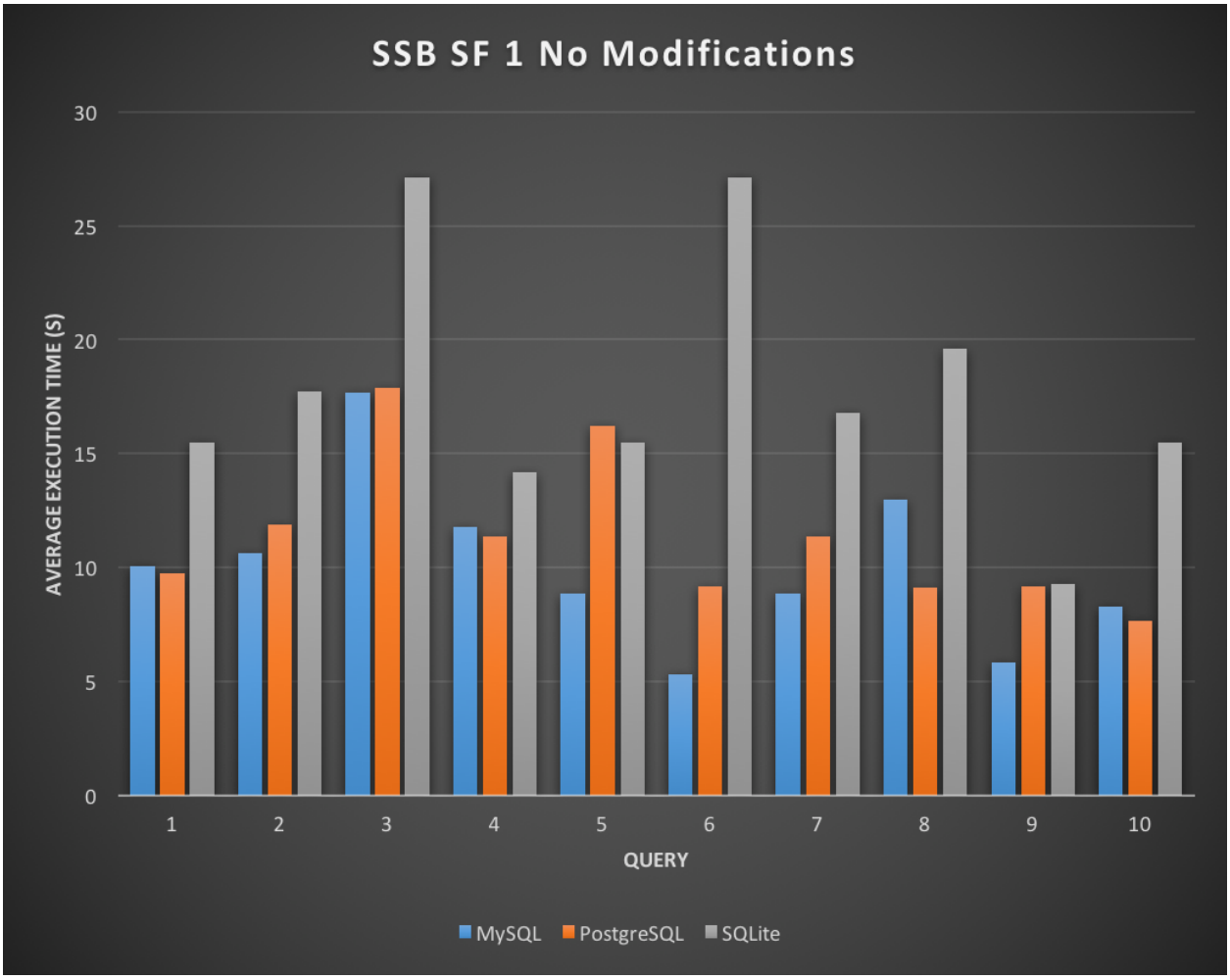

Figure 11: Star Schema Benchmark average query execution time at scale factor 1 without modification

After applying the indexing to the scale factor 1 SSB schema, the averages dropped to 14.64 seconds, 14.76 seconds, and 16.17 seconds respectively. As can be seen in Figure 12, the average query execution time dropped, but not at the same rate as in our

TPC-H tests, further demonstrating the impact of indexing on query performance.

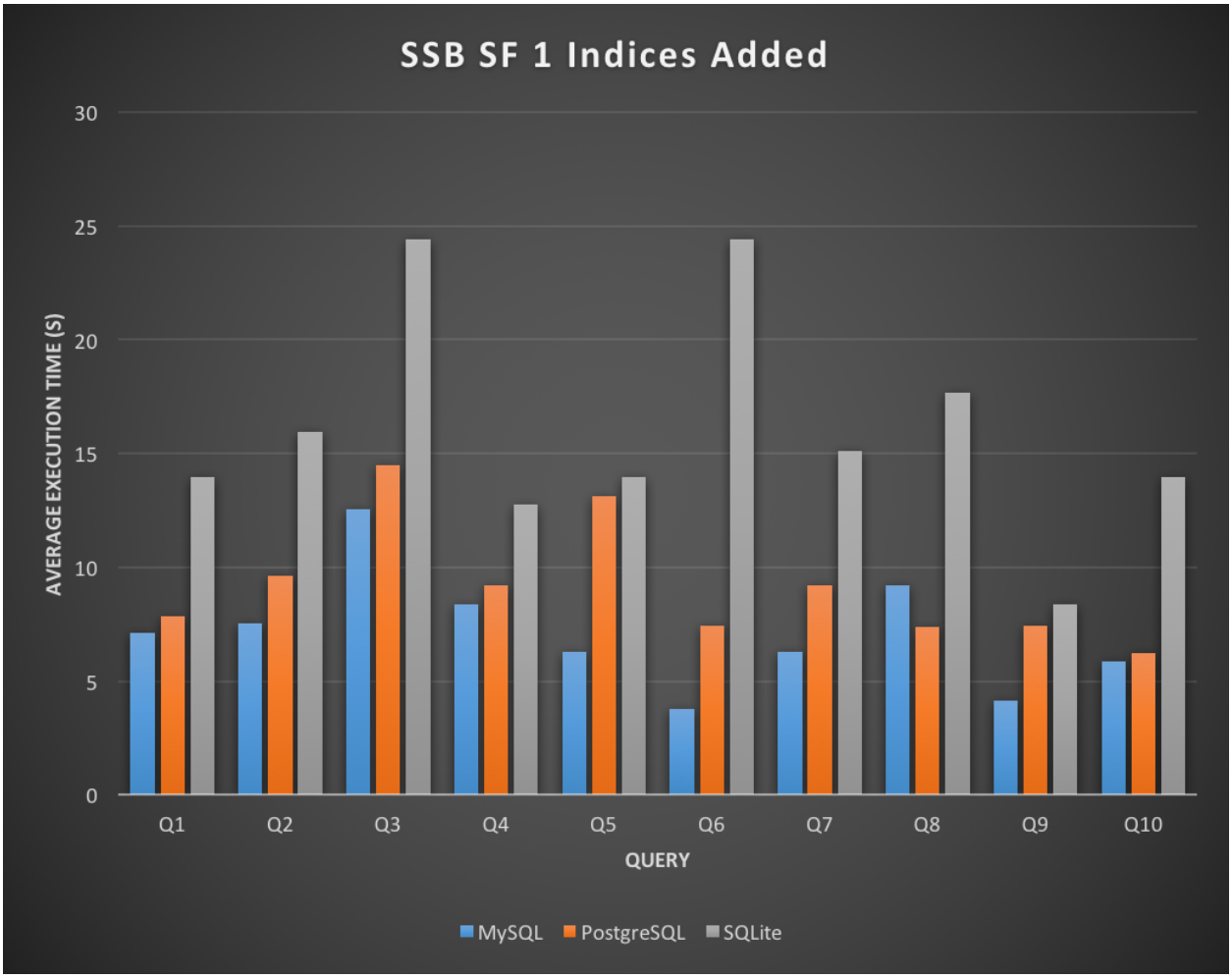

Figure 12: Star Schema Benchmark average query execution time at scale factor 1 after applying indices

Our results for the SSB showed that SQLite was the lowest performing DBMS at all scale factors, with and without indices being applied. At SF 1 without modifications, MySQL was the fastest DBMS for 6 out of the 10 queries run. After the indices were applied, the average execution time decreased across the board for each DBMS. With index modifications in place, MySQL remained the fastest performing DBMS while performing faster in 7 out of 10 queries. This result further demonstrates the influence of indexing on performance.

Figure 13 shows that at SF 10 without modification, MySQL was the top performing DBMS in 6/10 queries, followed by 4/10 for PostgreSQL. As expected SQLite was not the fastest DBMS in any of the 10 queries.

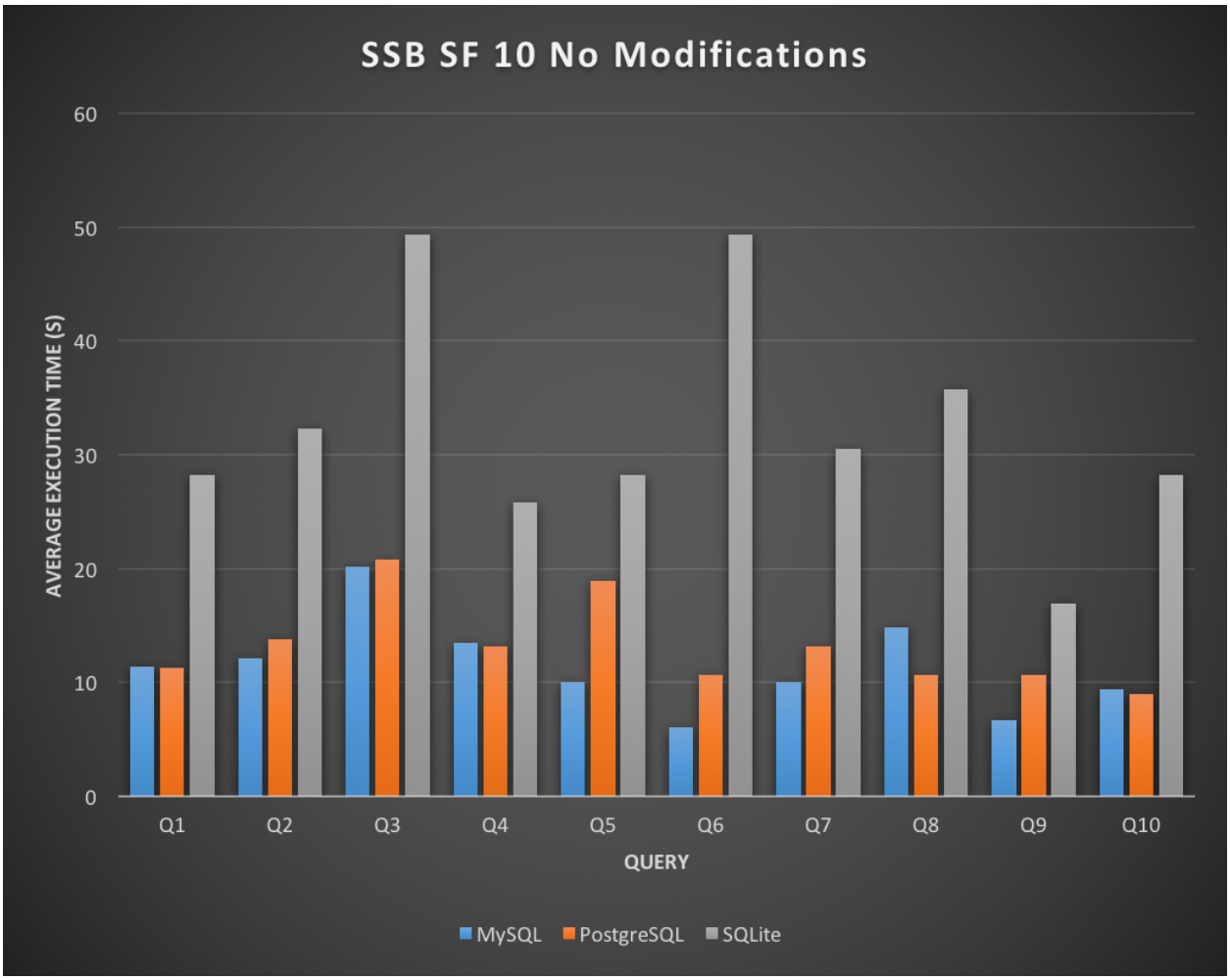

Figure 13: SSB average query execution time at scale factor 10

After applying our index choices to the SF 10 Star Schema Benchmark tables, the average execution time decreased. MySQL was the top performing DBMS in 9/10 queries, gaining an additional 3 queries from the non-modified tests. PostgreSQL was the fastest in 1/10 queries. The results are show in Figure 14.

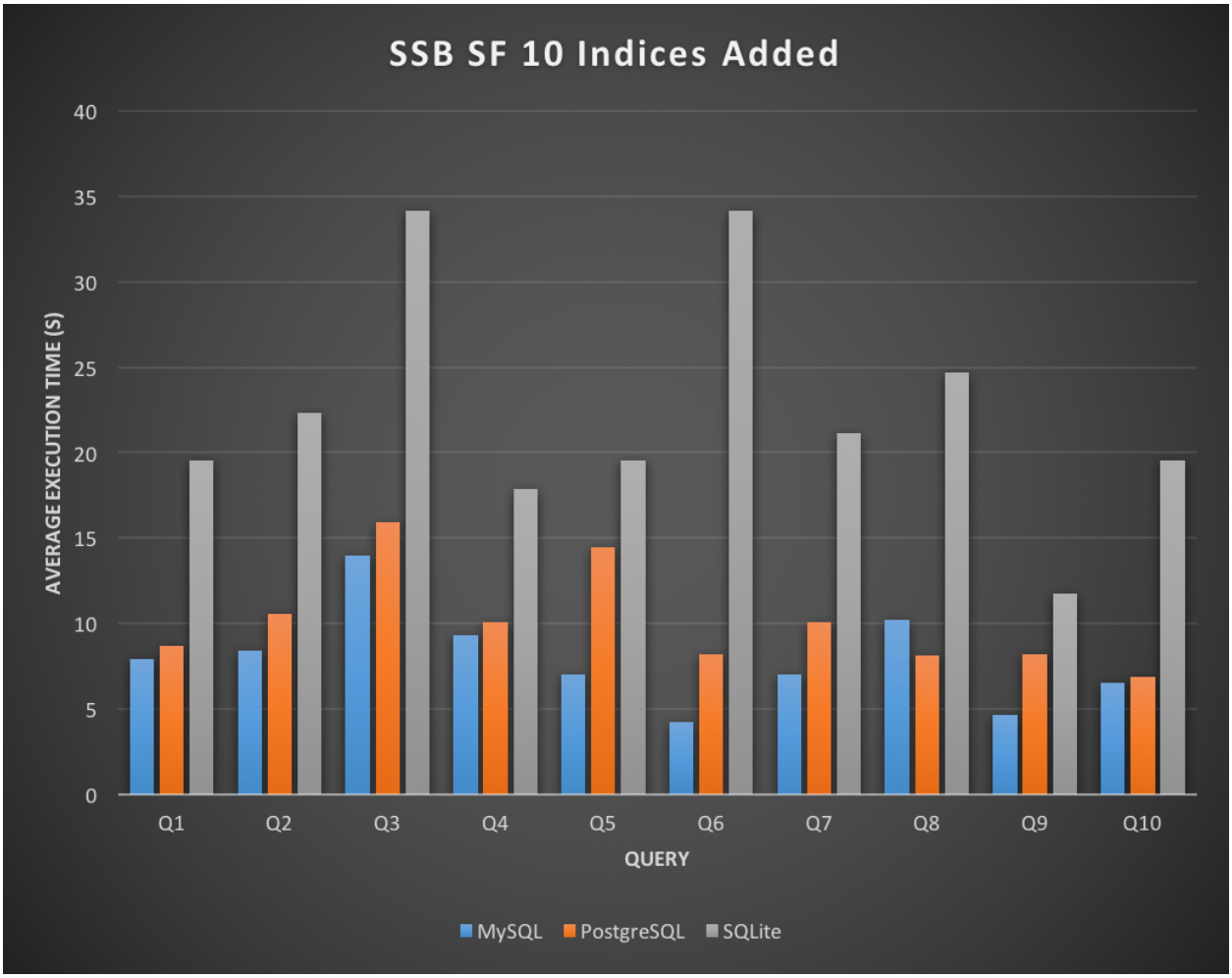

Figure 14: Star Schema Benchmark indices average query execution time at scale factor 10

The Star Schema Benchmark average query execution times show that both with and without modifications to the indices, MySQL remained the top performing DBMS, while SQLite consistently was ranked as the worst performing DBMS.

The queries defined in the TPC-H benchmarking standard are not written in a way so that database management systems can optimize query planning and therefore are inherently skewed. Here we examine the query path for TPC-H Q3 running on MySQL out of the box configuration, and no modifications outside of the traditional TPC-H schema. The unmodified query generated for TPC-H Q3 as seen in Figure 15 is a poorly written

query.

```
SELECT
    TOP 10 L_ORDERKEY,
    SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
    O_ORDERDATE, O_SHIPPRIORITY
FROM
    CUSTOMER, ORDERS, LINEITEM
WHERE
    C_MKTSEGMENT = 'BUILDING'
AND
    C_CUSTKEY = O_CUSTKEY
AND
    L_ORDERKEY = O_ORDERKEY
AND
    O_ORDERDATE < '1995-03-15'
AND
    L_SHIPDATE > '1995-03-15'
GROUP BY
    L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
ORDER BY
    REVENUE DESC, O_ORDERDATE
```

Figure 15: TPC-H Q3 unmodified WHERE clause

As mentioned in Section 2.3, for proper optimization and performance it is important to have a mastery of the business data before beginning to construct queries. The maximum difference between O_ORDERDATE and L_SHIPDATE is 121 days. Utilizing this knowledge of the data, we appended:

```
AND L_SHIPDATE <= dateadd(dd,122, cast('1995-03-15' as date)
```

to TPC-H Q3 after line 16, which resulted in a significantly different query optimization plan. Figure 16 shows our modified TPC-H Q3 after adding the WHERE clause comparison predicate at line 18.

```
1        SELECT
2            TOP 10 L_ORDERKEY,
3            SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
4            O_ORDERDATE, O_SHIPPRIORITY
5        FROM
6            CUSTOMER, ORDERS, LINEITEM
7        WHERE
8            C_MKTSEGMENT = 'BUILDING'
9        AND
10           C_CUSTKEY = O_CUSTKEY
11       AND
12           L_ORDERKEY = O_ORDERKEY
13       AND
14           O_ORDERDATE < '1995-03-15'
15       AND
16           L_SHIPDATE > '1995-03-15'
17       AND
18           L_SHIPDATE <= dateadd(dd,122, cast('1995-03-15' as date)
19       GROUP BY
20           L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
21       ORDER BY
22           REVENUE DESC, O_ORDERDATE
```

Figure 16: TPC-H Q3 - modified WHERE clause depicting restriction on L_SHIPDATE

Figure 17 shows the TPC-H Q3 query plan before making the index modifications. This plan begins with a full table scan across the CUSTOMER table, resulting in approximately 148,000 rows being scanned, followed by two nested join loops to find the matching orders, and order line items. In total 6 rows are returned from. The query optimizer then groups the results by L_ORDERKEY, O_ORDERKEY, O_SHIPPRIORITY and sorts the records on the highest revenue descending and order date descending.

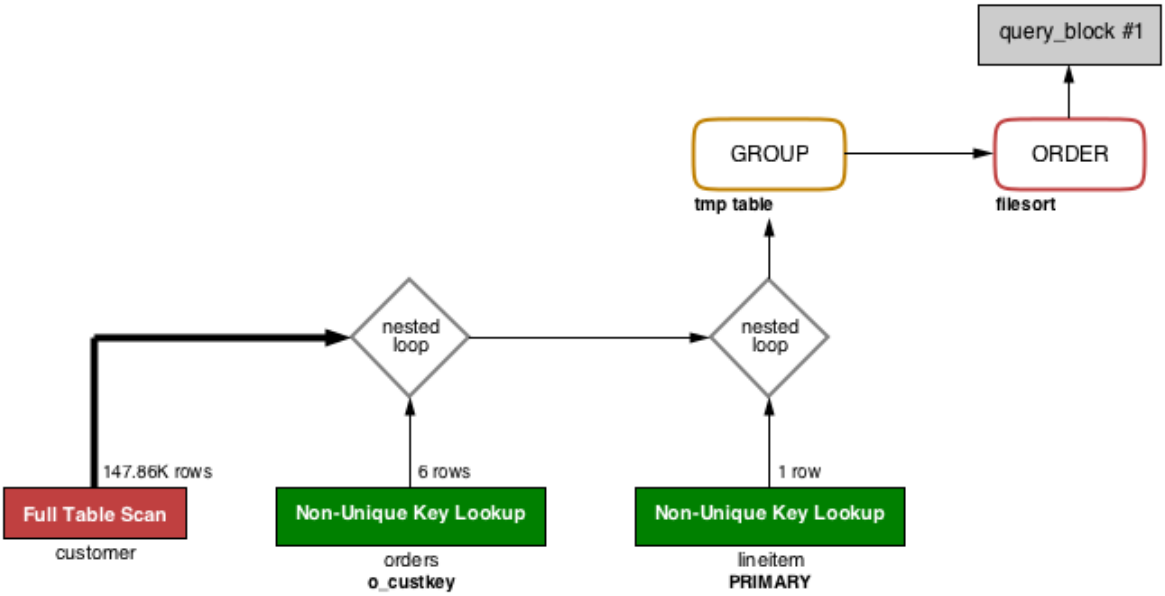

Figure 17: TPC-H Q3 unoptimized query plan

Figure 18 shows the TPC-H Q3 query plan after making the index modifications. Similar to the unoptimized plan, a full table scan occurs, however only 61,000 rows are required to be scanned to return the same six rows. As the size of the database increased the performance from the unoptimized query would begin to decrease. For higher values of the scale factor the system may become unstable or inoperative.

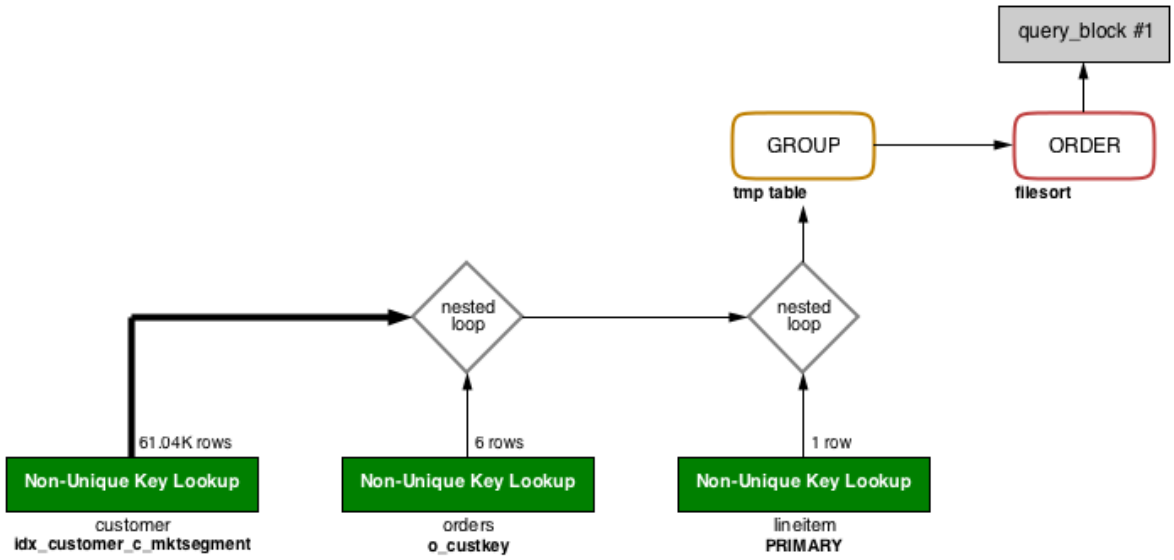

Figure 18: TPC-H Q3 optimized query plan with restriction on L_SHIPDATE

As can be seen in the TPC-H Q3 query plans shown in Figures 17 and 18, the TPC-H benchmark penalizes vendors for not optimizing to schemata that would not be found in the real-world. Both the original and modified TPC-H Q3 queries retrieve the same 6 orders, however, as can be seen in Figure 18, only 61,000 rows need to be scanned to return the same rows. No indices were added for this test. No query optimizer tuning or tricking occurred. The only difference between the two query paths is a result of the AND clause we added on L_SHIPDATE. This further supports the claim that TPC-H is intentionally designed overly complicated, echoing Stonebraker's criticisms.

7.2 Query Mapping Comparison

Our SQL benchmarking analysis outlined in Section 6.1 are not true one-to-one comparisons. We have mapped the queries as closely as possible. However, since the schemata are different, it is impossible to generate a true apples-to-apples comparison.

What can be measured between the two benchmarks is: how well the query path optimizer is able to perform, how the benchmark performs as the scale factor increases, coverage of business work-flow scenarios, and query simplicity.

Thirteen of TPC-H's 22 queries contain at least one correlated subquery, several contain multiple. This is a bad practice that is completely avoidable using JOINs. The Star Schema Benchmark contain correlated subqueries. As avoiding correlated subquery is a defined and well known process, points were reduced from TPC-H in both the Relevance and Simplicity categories. This fact adds weight to Stonebraker's claims.

Relevance: Gray says that it must accurately predict peak performance while operating within that problem domain. TPC-H is no longer considered relevant as it does not reflect industry best practices nor does the model that is presented within model reflect real-world scenarios that would be found in a typical data warehouse. During the mapping process we realized that TPC-H does not have a scenario that covers orders that have not shipped yet. This occurs because of the way in which the TPC-H schema is designed. TPC-H requires that an order be complete, having been shipped, received, and if it has been returned, a returned flag set. Business users may need to query orders before they shipped. TPC-H contains many columns such that contain comments, shipping instructions, and other fields that are irrelevant and cannot be used for decision making such as comment and shipping instruction columns. There are no aggregations or role-up functions that can be used on comment or text columns. SSB adds columns for LINEORDER supply cost for a given part, LINEORDER order supply cost summing for orders, and total order price. The Star Schema Benchmark is modeled after an actual real-world application which serves as a functioning template in many enterprise applications.

Portability: Both TPC-H and SSB are built around ANSI-SQL which means that they are theoretically portable to any SQL vendor if the vendor supports ANSI-SQL. In some cases, tweaks may be required for compatibility, but for all intents and purposes both should be considered portable.

Scalability: TPC-H 4.1.3.1, TPC-H does not scale past SF 100,000 which is a recognized limit and downfall. The Star Schema Benchmark has no such defined scale factor limitation, and due to the changes in the way data are distributed, it is assumed that it easily surpasses TPC-H. As the scale factor increases, the data becomes less meaningful.

There is a negative correlation between the benchmark's relevance and the size of the scale factor. As seen in Table 12, as the scale factor increases, the size of the CUSTOMER table grows. At the max scale factor for TPC-H, the CUSTOMER table grows to 15 billion customers.

Table 12: TPC-H CUSTOMER table scale factor

Scale Factor	Rows
1	150K
10	1.5M
100	15M
1,000	150M
100,000	15B

Like the CUSTOMER table, a negative correlation also exists between relevance and the size of the scale factor. Table 13 shows that as the PART table grows in proportion to the scale factor. Once the scale factor reaches the predefined TPC-H maximum, there are 20 billion parts in the database.

Table 13: TPC-H PART table scale factor

Scale Factor	Rows
1	200K
10	2M
100	20M
1,000	200M
100,000	20B

It makes sense that the tables LINEITEM and ORDERS would need to scale large, but to have all the tables (with the exception of NATION and REGION which remain constant) grow at the same does not make sense.

Simplicity: TPC-H is self-described as a simple decision support system benchmark [46], however as shown in Chapter 4, both the schema and query flights are far from simple. In contrast, the Star Schema Benchmark drastically reduces complexity of the database schema, eliminates unnecessary joins, and reduces the number of queries needed to successfully benchmark the system.

The results from our comparison against Gray's laws are listed in Table 14. These results show that the Star Schema Benchmark adheres more closely to Gray's laws of good benchmarks the TPC-H benchmark does.

Table 14: Comparison of the characteristics of decision support system benchmarks

	Relevant	Portable	Scalable	Simple	Score	
TPC-H	58%	78%	61.5%	65%	63.75%	✗
SSB	88%	78%	88%	91.5%	86.35%	✓

7.3 Research Resource Limitations

We faced several limitations throughout the research conducted in this thesis. Due to financial restrictions, such as hardware rental, operating system licensing, licensing costs of software systems such as Oracle DB and Microsoft SQL Server, we decided to utilize Amazon Web Services Micro free-tier environments, as well as running on (free and open source) Linux operating systems.

Table 15 shows an estimated break down of including Microsoft EC2 instances with Oracle DB and Microsoft SQL Server based on an estimated 80 hours of AWS EC2 usage. We averaged 334.14 hours per instance configuration running tests, re-running, and verifying data.

Table 15: Estimated cost of open source DBMS vs enterprise DBMS on EC2 Tier 2

Type	CPU	RAM	OS	Hours	Price	DBMS	DB Cost	Total Cost
micro	1	1	Linux	80	$0.01	MySQL	$0.00	$1.04
micro	1	1	Linux	80	$0.01	PostgreSQL	$0.00	$1.04
micro	1	1	Linux	80	$0.01	SQLite	$0.00	$1.04
micro	1	1	Windows	80	$0.07	SQL Server	$0.02	$5.46
micro	1	1	Windows	80	$0.07	Oracle	$0.04	$5.48
small	1	2	Linux	80	$0.03	MySQL	$0.00	$2.08
small	1	2	Linux	80	$0.03	PostgreSQL	$0.00	$2.08
small	1	2	Linux	80	$0.03	SQLite	$0.00	$2.08
small	1	2	Windows	80	$0.14	SQL Server	$0.04	$10.92
small	1	2	Windows	80	$0.14	Oracle	$0.07	$10.95
medium	2	4	Linux	80	$0.05	MySQL	$0.00	$4.16
medium	2	4	Linux	80	$0.05	PostgreSQL	$0.00	$4.16
medium	2	4	Linux	80	$0.05	SQLite	$0.00	$4.16
medium	2	4	Windows	80	$0.27	SQL Server	$0.07	$21.83
medium	2	4	Windows	80	$0.27	Oracle	$0.14	$21.90
Totals				1,200	$1.23		$0.37	$98.37

Using the actual average of 334.14 hours and the pricing quoted in Table 15, we used the AWS bill calculator, and arrived at an estimate of $13,748.21 which can be seen in

Figure 19.

⊖	Amazon RDS Service (US-East)		$ 12570.72
	DB instances:	$ 12562.08	
	Storage:	$ 8.64	
⊖	AWS Support (Business)		$ 1179.79
	Support for all AWS services:	$ 1179.79	
Free Tier Discount:		**$**	-2.30
Total Monthly Payment:		**$**	13748.21

Figure 19: Estimated usage cost for Oracle and Microsoft SQL Server (Windows) on EC2 Tier 2

The actual usage and cost is listed in Table 16. The time difference between estimated and actual time spent comes from the distribution algorithm and how TPC-H scales. As the SF increased, the size of the database grew linearly, which required additional memory and CPU processing power (and did not grow linearly as we had expected).

Table 16: Actual total usage and cost per hour for Linux OS on EC2 Tier T2

Type	CPU	RAM	Hours	Price	DBMS	DB Cost	Total Cost
micro	1	1	224.00	$0.01	MySQL	$0.00	$2.91
small	1	2	363.78	$0.03	MySQL	$0.00	$9.46
medium	2	4	538.95	$0.05	MySQL	$0.00	$28.03
micro	1	1	255.36	$0.01	PostgreSQL	$0.00	$3.32
small	1	2	414.70	$0.03	PostgreSQL	$0.00	$10.78
medium	2	4	219.52	$0.05	PostgreSQL	$0.00	$11.42
micro	1	1	227.36	$0.01	SQLite	$0.00	$2.96
small	1	2	472.76	$0.03	SQLite	$0.00	$12.29
medium	2	4	380.80	$0.05	SQLite	$0.00	$19.80
Totals			3,097.23	$0.27		$0.00	$100.96

Given that open source operating systems like Linux, and open source databases like MySQL, SQLite, and PostgreSQL are widely used in production application environments, we felt that the additional cost of testing TPC-H and SSB against would not be justified.

Chapter 8: Conclusion

The purpose of the research conducted within this thesis was to show whether Star Schema Benchmark can serve as a replacement for the TPC-H DSS benchmark. To serve as a replacement the SSB needs to perform at or beyond the same level of TPC-H. Our research has shown that TPC-H has many problems. As mentioned in Section 4.6, TPC-H has been criticized heavily by authorities on database benchmarking and data warehousing. We have shown that TPC-H penalizes vendors who are unable to optimize for poorly designed schema or the use of correlated subqueries. TPC-H heavily limits indexing capabilities and partitioning options.

It is our conclusion that the experiments conducted throughout this research have successfully shown that:

- TPC-H
  - is problematic
  - out of date and no longer relevant
  - intentionally overly complex
  - does not model a traditional DSS warehouse
  - does not cover all business decision work-flows
  - penalizes vendors for not optimizing to schemata that would not be found in the real-world
  - does not adhere to industry standard SQL best practices
- Star Schema Benchmark
  - is a better DSS benchmark than TPC-H
  - offers a much simpler schema and query execution set

- models a real-world scenario
- covers all business decision work-flows
- adheres to the Kimball definition of a data warehouse

Star Schema Benchmark helps to level the playing field so that IT decision makers and software architects can make informed choices when deciding which DBMS solution to implement. Using the Star Schema Benchmark as a drop-in replacement to TPC-H allows for more accurate benchmarks, and potentially saves companies millions of dollars as seen in the preface.

As mentioned in Section 7.3, there were several limitations to our experiments due to cost. Our results and conclusion are based on open-source operating systems and open source database vendors. We would like to expand our current research to include paid operating systems such as Windows and Mac OSX, as well as enterprise paid license DBMSs such as Oracle and SQL Server. This future work could validate the research conducted in this thesis and apply it to all systems.

REFERENCES

[1] A. K. Dutta and R. Hasan, "How much does storage really cost? Towards a full cost accounting model for data storage," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*. Springer International Publishing, 2013, vol. 8193 LNCS, pp. 29–43.

[2] A. Tasistro, "Application of Business Intelligence Techniques to Analyze IT Project Management Data," in *2015 Int. Work. Data Min. with Ind. Appl.* IEEE, sep 2015, pp. 18–24.

[3] L. Endo *et al.*, "Supportive metrics to estimate the effort to develop Business Intelligence system," in *2016 11th Iber. Conf. Inf. Syst. Technol.* IEEE, jun 2016, pp. 1–6.

[4] M. Barata *et al.*, "An overview of decision support benchmarks: Tpc-ds, TPC-H and SSB," in *New Contributions in Information Systems and Technologies*, ser. Adv. Intell. Syst. Comput., Á. Rocha *et al.*, Eds., vol. 1. Springer, 2015, pp. 619–628.

[5] RTI, "The economic impact of inadequate infrastructure for software testing," National Institute of standards & Technology, Tech. Rep. (Planning) 02–3, 2002.

[6] R. A. Kadadi *et al.*, "Challenges and Opportunities with Big Data Visualization," *MEDES '15 Proc. 7th Int. Conf. Manag. Comput. Collect. Intell. Digit. Ecosyst.*, vol. 1542, no. 12, pp. 169 – 173, aug 2015.

[7] J. Gray, *Benchmark Handb. Database Trans. Syst.*, ser. The Morgan Kaufmann series in data management systems. M. Kaufmann Publishers, 1993.

[8] T. Teorey *et al.*, *Database Design: Know It All*. Morgan Kaufmann Publishers/Elsevier, 2008.

[9] T. Teorey, *Database Modeling and Design*. San Fransisco, CA: Morgan Kaufmann Publishers, 2006.

[10] R. Kimball and J. Caserta, *The Data Warehouse ETL Toolkit*. Wiley Pub, 2004.

[11] H. y. Lin and C. l. Chen, "Using compressed b trees for line-based database indexes," in *Signal Process. Inf. Technol. 2006 IEEE Int. Symp.*, aug 2006, pp. 258–263.

[12] M. Shao *et al.*, "Dbmbench: Fast and accurate database workload representation on modern microarchitecture," *Proc. 2005 Conf. Cent. Adv. Stud. Collab. Res.*, pp. 254–267, 2005.

[13] M. Leonard, "Database Design Theory," in *Relational Database Des.*, 3rd ed. Boston, MA: Morgan Kaufmann, 1992, p. 43.

[14] A. Sharma and M. Sood, “Utilizing materialized views to formulate business. intelligence,” in *Proc. 2014 3rd Int. Conf. Parallel, Distrib. Grid Comput. PDGC 2014*, dec 2014, pp. 22–26.

[15] V. K. Myalapalli *et al.*, “Accelerating sql queries by unravelling performance bottlenecks in dbms engine,” in *2015 Int. Conf. Energy Syst. Appl.*, oct 2015, pp. 7–12.

[16] P. Bosc and O. Pivert, “About projection-selection-join queries addressed to possibilistic relational databases,” *IEEE Trans. Fuzzy Syst.*, vol. 13, no. 1, pp. 124–139, feb 2005.

[17] T. Koppelaars and L. deHaan, “Tuple, table, and database predicates,” in *Appl. Math. Database Prof.* Berkeley, CA: Apress, 2007, pp. 117–138.

[18] C. J. Date, *Relational Theory for Computer Professionals*, 1st ed. ”O’Reilly Media, Inc.”, 2013.

[19] S. Sumathi, *Fundamentals of Relational Database Management Systems*. Springer, 2007.

[20] T. Willis and B. Newsome, *Beginning Visual Basic 2005*. Indianapolis, IN: Wiley Pub, 2006.

[21] D. J. Abadi *et al.*, “Column-stores vs. row-stores,” in *Proc. 2008 ACM SIGMOD Int. Conf. Manag. data*, ser. SIGMOD ’08. New York, NY: ACM Press, 2008, p. 967.

[22] S. Aghav, “Database compression techniques for performance optimization,” in *2010 2nd International Conference on Computer Engineering and Technology*, vol. 6, April 2010, pp. V6–714–V6–717.

[23] P. Seshadri *et al.*, “Complex query decorrelation,” in *Icde*, feb 1996, pp. 450–458.

[24] P. Boncz *et al.*, “TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark,” *Perform. Charact. Benchmarking*, vol. 8391, pp. 61–76, 2014.

[25] V. Simhadri *et al.*, “Decorrelation of user defined function invocations in queries,” *Proc. Int. Conf. Data Eng.*, pp. 532–543, mar 2014.

[26] W. Kim, “On optimizing an sql-like nested query,” *ACM Trans. Database Syst.*, vol. 7, no. 3, pp. 443–469, sep 1982.

[27] M. Elhemali *et al.*, “Execution strategies for sql subqueries,” in *Proc. 2007 ACM SIGMOD Int. Conf. Manag. data*, ser. SIGMOD ’07. Beijing, China: ACM, 2007, pp. 993–1004.

[28] U. Dayal, “Of nests and trees: A unified approach to processing queries that contain nested subqueries, aggregates, and quantifiers,” in *VLDB’87, Proceedings of 13th International Conference on Very Large Data Bases*. Morgan Kaufmann, sep 1987, pp. 197–208.

[29] E. F. Codd, *The Relational Model for Database Management: Version 2*. Addison-Wesley, 1990.

[30] B. Scalzo, *Database benchmarking : practical methods for Oracle & SQL Server*, ser. IT in-focus. Rampant TechPress, 2006.

[31] C. Turbyfill *et al.*, “As3ap - a comparative relational database benchmark,” in *Dig. Pap. COMPCON Spring 89. Thirty-Fourth IEEE Comput. Soc. Int. Conf. Intellect. Leverage*, feb 1989, pp. 560–564.

[32] D. Bitton *et al.*, “A measure of transaction processing power,” *Datamation*, vol. 31, no. 7, pp. 112–118, apr 1985.

[33] S. B. Yao *et al.*, “Analysis of database system architectures using benchmarks,” *IEEE Transactions on Software Engineering*, vol. SE-13, no. 6, pp. 709–725, June 1987.

[34] W. H. Inmon, *Building the Data Warehouse*, 2nd ed. New York, NY: John Wiley & Sons, Inc., 1992.

[35] X. Chen *et al.*, “Adjoined dimension column clustering to improve data warehouse query performance,” in *Proc. Int. Conf. Data Eng.* IEEE, apr 2008, pp. 1409–1411.

[36] K. Gao and I. Pandis, “Implementation of tpc-h and tpc-c toolkits,” Master’s thesis, Carnegie Mellon University, Pittsburg, PA.

[37] R. Nambiar and M. Poess, “Keeping the tpc relevant!” *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1186–1187, aug 2013.

[38] S. S. Coates, “Comparing the performance of open source and proprietary relational database management systems,” Ph.D. dissertation, Northcentral University, jan 2009.

[39] C. Phipps and K. C. Davis, “Automating data warehouse conceptual schema design and evaluation,” in *Proc. 4th Intl. Work. DMDW*, 2002, pp. 23–32.

[40] L. Cabibbo and R. Torlone, “The design and development of a logical system for olap,” in *Data Warehous. Knowl. Discov.* Springer Berlin Heidelberg, 2000, pp. 1–10.

[41] P. Bizarro and H. Madeira, “Adding a Performance-Oriented Perspective to Data Warehouse Design,” *Data Warehous. Knowl. Discov.*, vol. 2454, pp. 232–244, 2002.

[42] H. Vandierendonck and P. Trancoso, “Building and validating a reduced tpc-h benchmark,” in *Proc. - IEEE Comput. Soc. Annu. Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst. MASCOTS*.

[43] M. Shao *et al.*, “DBmbench: Fast and Accurate Database Workload Representation on Modern Microarchitecture.” IBM Press, oct 2005, pp. 254–267.

[44] T. Strandell, *Open Source Database Systems*. Vdm Verlag Dr. Muller Aktiengesellschaft & Co. Kg, mar 2010.

[45] P. E. O’Neil, “The set query benchmark,” in *Benchmark Handb. Database Trans. Syst.*, 2nd ed., ser. The Morgan Kaufmann series in data management systems. M. Kaufmann Publishers, 1991, pp. 209–245.

[46] T. P. P. C. TPC, *TPC Benchmark H Standard Specification*, Std., 1999. [Online]. Available: http://www.tcp.org/hspec.html

[47] M. Al-Kateb *et al.*, *Adding a Temporal Dimension to the TPC-H Benchmark*. Berlin, Heidelberg: Springer, 2013, pp. 51–59.

[48] M. Stonebraker, “Performance evaluation and benchmarking,” R. Nambiar and M. Poess, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. A New Direction for TPC?, pp. 11–17.

[49] C. Coronel *et al.*, *Design, Implementation, and Management*, 9th ed., ser. Management Information Systems. Boston, MA: Cengage Learning, 2009.

[50] P. O’Neil *et al.*, “The star schema benchmark and augmented fact table indexing,” in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, ser. Lecture Notes in Computer Science, R. Nambiar and M. Poess, Eds., vol. 5895 LNCS. Berlin, Heidelberg: Springer, 2009, pp. 237–252.

[51] M. Stonebraker *et al.*, “The end of an architectural era: (it’s time for a complete rewrite),” in *Proc. 33rd Int. Conf. Very Large Data Bases*, ser. VLDB ’07. Vienna, Austria: VLDB Endowment, 2007, pp. 1150–1160.

[52] T. Rabl *et al.*, “Variations of the star schema benchmark to test the effects of data skew on query performance,” in *Proc. ACM/SPEC Int. Conf. Int. Conf. Perform. Eng.*, ser. ICPE. New York, NY: ACM Press, 2013, pp. 361–372.

[53] D. Henschen. (2010, sep) Reconsidering TPC Database Performance Benchmarks - InformationWeek. [Online]. Available: http://www.informationweek.com/database/reconsidering-tpc-database-performance-benchmarks/d/d-id/1092179

[54] C. Monash. (2009) The tpc-h benchmark is a blight upon the industry. [Online]. Available: http://www.dbms2.com/2009/06/22/the-tpc-h-benchmark-is-a-blight-upon-the-industry/

[55] D. J. Abadi *et al.*, “Integrating compression and execution in column-oriented database systems,” in *Sigmod ’06*. New York, New York: ACM Press, jun 2006, pp. 671–782.

[56] A. U. Tansel, “Temporal relational data model,” *IEEE Trans. Knowl. Data Eng.*, vol. 9, no. 3, pp. 464–479, may 1997.

[57] *List of SQL reserved words*, 2010. [Online]. Available: https://www.drupal.org/docs/develop/coding-standards/list-of-sql-reserved-words

[58] E. O’Neil *et al.*, “Bitmap index design choices and their performance implications,” in *Proc. Int. Database Eng. Appl. Symp. IDEAS*. IEEE, sep 2007, pp. 72–84.

[59] K. Dehdouh *et al.*, *Columnar NoSQL Star Schema Benchmark*. Larnaca, Cyprus: Springer International Publishing, sep 2014, vol. 8748, pp. 281–288.

[60] K. J. Engemann, *Performance Evaluation and Benchmarking*, ser. Lecture Notes in Computer Science, R. Nambiar and M. Poess, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, oct 2006, vol. 13, no. 5.

[61] D. Phillips. (2010) ssb-dbgen. [Online]. Available: https://github.com/electrum/ssb-dbgen

[62] SQLite. (2015) Appropriate Uses For SQLite. [Online]. Available: https://www.sqlite.org/whentouse.html