The Making of TPC-DS

Conference Paper · January 2006

Source: DBLP

CITATIONS
155

READS
1,661

2 authors:

Raghu Nambiar
Cisco Systems, Inc
79 PUBLICATIONS **1,590** CITATIONS
SEE PROFILE

Meikel Poess
Oracle Corporation
78 PUBLICATIONS **1,911** CITATIONS
SEE PROFILE

The Making of TPC-DS

Raghunath Othayoth
Hewlett-Packard Company
20555 Tomball Parkway
Houston, TX-77070, USA
281-518-2748
raghunath.othayoth@hp.com

Meikel Poess
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA-94107, USA
650-633-8012
meikel.poess@oracle.com

ABSTRACT

For the last decade, the research community and the industry have used TPC-D and its successor TPC-H to evaluate performance of decision support technology. Recognizing a paradigm shift in the industry the Transaction Processing Performance Council has developed a new Decision Support benchmark, TPC-DS, expected to be released this year. From an ease of benchmarking perspective it is similar to past benchmarks. However, it adjusts for new technology and new approaches the industry has embarked on in recent years. This paper describes the main characteristics of TPC-DS, explains why some of the key decisions were made and which performance aspects of decision support system it measures.

1. INTRODUCTION

The origins of Decision Support Systems (DSS) reach as far back as the 1960s when model-oriented DSS systems were developed. At that time, emphasis was on building Management Information Systems as "integrated, man/ machine system for providing information to support the operations, management, and decision-making functions in an organization" [2]. In 1981 Bonczek, Holsapple, and Whinston created a theoretical framework for understanding the issue associated with designing knowledge-oriented Decision Support Systems [1]. Advocated by Bill Immon and Ralph Kimball, Relational Database systems have been increasingly used to build DSS systems starting in 1990 [6][7].

Recognizing the need for a standard benchmark to measure the performance of DSS systems, the Transaction Processing Performance Council (TPC) released its first data warehouse benchmark, TPC-D, in April 1994. For the technology available at that time, TPC-D imposed many challenges both on hardware and on DBMS systems. It implemented a data warehouse using a 3rd Normal Form (3NF) schema consisting of 8 tables. Anticipating the data explosion of data warehouses in the industry, TPC-D was developed to scale from 1 gigabyte to 3 terabytes (TB) of raw data pushing IO subsystems to their limit. Benchmarks used predetermined sizes, namely, scale factors. Each scale factor corresponded to the raw data size of the data warehouse. 17 complex and long running queries combined with 2 data maintenance functions (insert and delete) confronted most query optimizers with hefty problems. 6 of the 8 tables grew linearly with the scale factor and were populated with data that was uniformly distributed. The development of aggregate/summary structures (e.g. join indices, summary tables, materialized views, etc.) that are automatically maintained and transparently used by the query optimizer via query rewrite was spurred by TPC-D because this technology decreased query elapsed times resulting in an over proportional increase in the main performance metric. As a result the TPC decided that TPC-D was effectively broken and spun off two modified versions of TPC-D, namely TPC-H and TPC-R in April 1999.

Using the same schema, the same data generator with an additional scale factor of 10 TB, the same data distributions and 6 more queries, TPC-H - an ad-hoc decision support benchmark, and TPC-R - a business reporting decision support benchmark are nearly identical to TPC-D. The difference between TPC-H and TPC-R is the prior knowledge of the workload that they assume. TPC-H represents an environment where database administrators do not know which queries will be executed against a database system; hence, knowledge about its queries and data may not be used to optimize the DBMS system. In TPC-R, pre-knowledge of the queries is assumed and may be used for defining aggregate/summary structures. Since TPC-R never attracted any notable attention, it was decommissioned in January 2006.

It is without doubt that the family of decision support benchmarks, the TPC has released over the last decade, satisfied the need for data warehouse benchmarking. It spurred competition, improved system performance and motivated manufacturers to push technology to the limits. Numerous research papers using the TPC-H data generator and its query set [9] and the large body of published TPC-H results vouch for it. Last December the combined number of results has surpassed 200. In contrast, one has to admit that the requirements and implementations of today's decision support systems differ quite significantly from the original benchmark ideas that were developed for TPC-D and that were carried over into TPC-H and TPC-R. In an effort to extend the lifetime of TPC-H, the TPC increased its publishable data volumes by two scale factors, 30,000 and 100,000, equivalent of 30TB and 100TB of raw data.

The differences between today's decision support systems and the TPC-H benchmark specification are manifold. The TPC-H schema, although sufficiently complex to test the early systems,

VLDB '06, September 12–15, 2006, Seoul, Korea.

it is not representative of today's more complex DSS implementations. Today's schemas are typically composed of a larger number of tables and columns. Furthermore, the industry's choice of schema implementation has shifted from pure 3NF schemas to variations of the star schema, such as snowflake schemas. The purity of TPC-H's 3NF and the low number of tables and columns does not fully reveal the differences in indexing techniques and query optimizers. Since the main tables scale linearly with the database size (scale factor), the cardinalities of some tables reach unrealistic proportions at large scale factors. For instance, at scale factor 100,000 the database models a retailer selling 20 billion distinct parts to 15 billion customers at a transaction rate of 150 billion per year - the dream of every CEO, but quite unrealistic. The database population, consisting of mostly un-skewed and synthetic data imposes little challenges on statistic collection and optimal plan generation by the query optimizer. The TPC-H data maintenance functions (rf1, rf2) merely constrain a potential excessive use of indexes rather than testing the DBMS' capability of performing realistic data maintenance operations, common during Extraction Transformation and Load (ETL). Data maintenance functions insert and delete orders randomly rather than ordered by time. The inserted data is assumed to be clean so that no data transformations are necessary. Data are loaded and deleted from 2 out of 8 tables. There are relatively few distinct queries in TPC-H, and since they are known before benchmark execution, engineers can tune optimizers and execution paths to artificially increase performance of the system under test. Also, actual data warehouses are not subject to the TPC-H benchmark constraints and will define indexes on non-date and non-key columns as well as contain summary tables.

Having realized these deficiencies, the TPC has developed its next generation decision support benchmark, TPC-DS. This paper serves two purposes. Firstly, it describes the main characteristics of TPC-DS and explains rationale of the key decisions. Secondly, it abstracts from the development of TPC-DS to a general approach on how decision support benchmarks can be developed efficiently. The first paper about TPC-DS, published in 2002 [8], outlined the ideas for a new decision support benchmark. It was followed by several publications about detailed areas of TPC-DS such as the data and query generators [10][11]. This paper presents the current state of the specification as it will be presented for TPC member company review[1].

TPC-DS takes the marvels of TPC-H and TPC-R and fuses them into a modern DSS benchmark. The main focus areas:

- Multiple snowflake schemas with shared dimensions
- 24 tables with an average of 18 columns
- 99 distinct SQL 99 queries with random substitutions
- More representative skewed database content
- Sub-linear scaling of non-fact tables
- Ad-hoc, reporting, iterative and extraction queries
- ETL-like data maintenance

While TPC-DS may be applied to any industry that must transform operational and external data into business intelligence, the workload has been granted a realistic context. It models the decision support tasks of a typical retail product supplier. The goal of selecting a retail business model is to assist the reader in relating intuitively to the components of the benchmark, without tracking that industry segment so tightly as to minimize the relevance of the benchmark. The schema, an aggregate of multiple star schemas, contains essential business information, such as detailed customer, order, and product data for the classic sales channels: store, catalog and Internet. Wherever possible, real world data are used to populate each table with common data skews, such as seasonal sales and frequent names. In order to realistically scale the benchmark from small to large datasets, fact tables scale linearly while dimensions scale sub linearly. The benchmark abstracts the diversity of operations found in an information analysis application, while retaining essential performance characteristics. As it is necessary to execute a great number of queries and data transformations to completely manage any business analysis environment, TPC-DS defines 99 distinct SQL-99 (with OILAP amendment) queries and 12 data maintenance operations covering typical DSS like query types such as ad-hoc, reporting, iterative (drill down/up) and extraction queries and periodic refresh of the database.

Due to strict implementation rules it is possible to amalgamate ad-hoc and reporting queries into the same benchmark, i.e., it is possible to use sophisticated auxiliary data structures for reporting queries while prohibiting them for ad-hoc queries. Although the emphasis is on information analysis, the benchmark recognizes the need to periodically refresh the database (ETL). The database is not a one-time snapshot of a business operations database, nor is it a database where OLTP applications are running concurrently. The database must be able to support queries and data maintenance operations against all tables. Some TPC benchmarks (e.g., TPC-C and TPC-APP) model the operational aspect of the business environment where transactions are executed on a real time basis, other benchmarks (i.e. TPC-H) address the simpler, more static model of decision support. The TPC-DS benchmark, however, models the challenges of business intelligence systems where operational data is used both to support sound business decisions in near real time and to direct long-range planning and exploration. The TPC-DS operations address complex business problems using a variety of access patterns, query phrasings, operators, and answer set constraints.

The following sections detail the cornerstones of TPC-DS. The first two sections describe the database schema and the data set that populates it. The following section is about the workload, both query and data maintenance. While the Execution Rules and Metric section describes how the queries and data maintenance operations are run against the schema and how the execution times form the final metric, which can be used to compare system performances.

2. LOGICAL SCHEMA CREATION

DSS systems have evolved over time from model oriented, proprietary implementations to systems based on commercial database implementations. TPC-D and its successors, TPC-H and TPC-R assumed a 3rd Normal Form (3NF) schema. However, over the years the industry has expanded towards star schema approaches. Accommodating this development TPC-DS implements a hybrid schema between a 3NF and a pure star schema, namely multiple snowflake schemas.

2.1. Snowstorm Schema

A star schema includes a large fact table and several small dimension (lookup) tables. The fact table stores frequently added transaction data such as sales, returns and inventory changes. Each dimension table stores less frequently changed or added data supplying additional information for fact table transactions, such as customers who made purchases. An extension to the pure star schema, the snowflake schema separates static data in the outlying dimension tables from the more dynamic data in the inner dimen-

[1] TPC member company review is the final step before voting. During this phase, the specification will be made available to all TPC Members and the public for formal review.

sion tables and the fact tables. That is, in addition to their relation to the fact table, dimensions can have relations to other dimensions.

Conversely, 3NF modeling, being the classical relational-database modeling technique, minimizes data redundancy through normalization. When compared to a star schema, a 3NF schema typically has a larger number of tables due to its normalization process.

A smart schema design lays the foundation for a good query set. If the schema does not allow for the designing of queries that test the performance of all aspects of a DSS, the benchmark has failed one of its main goals. Choosing a multiple-snowflake schema allows the TPC-DS to exercise all aspects of commercial DSS, built with today's DBMS.

The multiple snowflake-schema approach challenges query execution of both star schema and 3NF execution paths. Typical executions in a star schema involve bitmap accesses, bitmap merges, bitmap joins and conventional index driven join operations. The access paths in a 3NF DSS system are dominated by large hash-joins, and conventional index driven joins are also common. In both systems large aggregations and sort operations (group by) are widespread. As will be shown in Section **Error! Reference source not found.** the query workload uses the characteristics of the snowflake schema to test both star schema and the 3NF executions. This diversity imposes challenges both on hardware and software systems. High sequential I/O-throughput is very critical to excel in large hash-join operations. At the same time, index driven queries stress the I/O subsystems ability to perform small random I/Os. Additionally, this diversity also challenges the query optimizer in its decision to either use a star schema approach, such as star transformation, or a more traditional approach, such as nested loops, hash-joins etc. This seems to be an area in which today's query optimizers have huge deficits.

The size of the schema and its three sales channels allow for amalgamating both ad-hoc and reporting queries into the same benchmark. An ad-hoc querying workload simulates an environment in which users connected to the database system send individual queries that are not known in advance. The system's database administrator (DBA) cannot optimize the database system specifically for this set of queries. Consequently, execution time for those queries can be very long. In contrast, queries in a reporting workload are very well known in advance. As a result, the DBA can optimize the database system specifically for these queries to execute them very rapidly by using clever data placement methods (e.g. partitioning and clustering) and auxiliary data structures (e.g. materialized views and indexes). Amalgamating both types of queries has been traditionally difficult in benchmark environments since per the definition of a benchmark all queries, apart from bind variables, are known in advance. TPC-DS accomplishes this fusion by dividing the schema into reporting and ad-hoc parts. For the reporting part of the schema complex auxiliary data structures are allowed, while for the ad-hoc part only basic auxiliary data structures are allowed. The idea behind this approach is that the queries accessing the ad-hoc part constitute the ad-hoc query set while the queries accessing the reporting part are considered the reporting queries.

Systems (database, OS and hardware) which can perform the entire spectrum of today's DSS algorithms, such as bitmap look-ups, bitmap merges, complex query rewrites, index driven joins, hash driven joins and large sort operations, will excel in TPC-DS.

2.2. Snowstorm Schema

As mentioned earlier, TPC-DS models the decision support functions of a retail product supplier. The supporting schema contains vital business information such as customer, order, and product data. The imaginary retail company sells goods through the three distribution channels, store, catalog and Internet (*web*). An inventory fact table is shared between the catalog and the Web channel. For simplicity and space reasons, Figure 1 shows only an excerpt of the entire schema. It focuses on the store sales channel. For the entire schema, please refer to the public release of a preliminary draft [3].

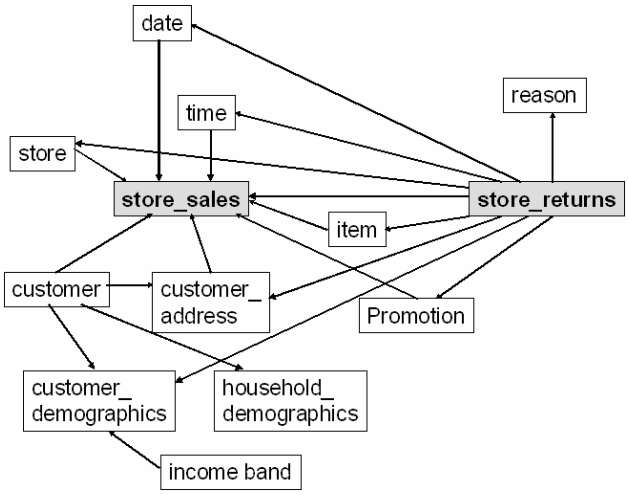


Figure 1: Store Sales Snow Flake Schema

The store sales channel consists of two fact tables, modeling sales and return transactions. Each fact table is organized as a snowflake schema with the traditional dimensions, such as *date_dim*, *time*_dim[2], *store* etc. Customer information is normalized into *Customer*, *Customer Address*, *Household Demographics*, and *Customer Demographics* which is further normalized into *Income Band*. The same constellation exists for store returns. Most dimensions are shared between the two fact tables, while the *reason* dimension is added only to the *Store Return* fact table.

This allows for writing challenging queries. For instance, Customer Address is both referenced from the Store Sales fact table and the Customer dimension. Having two relationships allows expressing both the current address and the address at the time of the sales transaction. This circular relationship makes it challenging to identify the optimal star transformation, especially, if a query references and defines predicates on a Customer Address table joined to Customer and on a second Customer Address table joined to Store Sales tables.

The two fact tables, Store Sales and Store Returns are related through the foreign key combination *Ticket Number* and *Item_sk*. Through this relationship large fact-to-fact table joins are possible. In the traditional single table star schema this would have been possible only through a self- join. Other large fact-to-fact table joins are possible by joining two fact tables from different sales channels such as store and web sales. Although there is no primary/foreign key defined between different sales channels, joins can be performed on mutual dimensions such as Item and Customer.

Two of the three sales channels, Store and Web constitute the ad-hoc part, while the Catalog channel constitutes the reporting

[2] _dim was added to the date and time dimensions to avoid conflicts with vendor specific reserved table names.

part of the schema. Hence, queries referencing[3] the Catalog channel are ad-hoc queries while all other queries are reporting queries.

Another distinctive characteristic of the TPC-DS schema is the number of columns in each table. The average number of columns is 18, allowing for a rich query set with predicates on many columns. Table 1 presents some statistics about the schema, e.g. number of tables and columns.

Number of fact tables	7	
Number of dimension tables	17	
Number of columns	min	3
	max	34
	avg	18
Number of foreign keys	104	
Row length [bytes][4]	min	16
	max	317
	avg	136

Table 1: Schema Statistics

3. DATA SET

The design of the data set is not only driven by the needs to challenge the statistic gathering algorithms and the query optimizer, but also by the need to challenge data placement algorithms, such as clustering, vertical or horizontal partitioning. A good data set design includes proper data set scaling, both domain and tuple scaling. In TPC-DS we use a hybrid approach of domain and data scaling (see Section 3.1). The data domain is also of very high importance. While pure synthetic data generators have great advantages, TPC-DS follows a hybrid approach of both synthetic and real world based data domains. Synthetic data sets are well understood, easy to define and implement. However, following the TPC's paradigm to create benchmarks that businesses can relate to, a hybrid approach to data set design scores many advantages over both pure synthetic and pure real world data (see Section 3.2). Additionally, as will be explained in Section **Error! Reference source not found.**, it is pertinent for a good workload design to cover the entire data set. This imposes great challenges on both the workload generator and the data generator, which resulted in a closely coupling of the two tools.

Similarly to previous decision support benchmarks the data set for TPC-DS scales in terms of discrete scale factors. These are 100, 300, 1000, 3000, 10000, 3000 and 100000. Benchmark publications using other scale factors are not valid. Each scale factor corresponds to the data size in Gigabyte of the data that needs to be loaded. It is also called the raw-data size. Depending on the techniques used in the data management software, e.g. compression, the size of the internal representation of the raw-data can be different than the raw-data itself.

The following sections introduce and rationalize the decisions of TPC-DS' approach to data scaling. It further discusses its impact to decision support system benchmarking. For a detailed description of the data generator, please refer to [10].

3.1. Data Set Scaling

Scaling within a data set can take on two different characteristics. In one case, the number of tuples in the dataset is expanded, but the underlying value sets (the domains) remain static. The business analogy here would be a system where the number of items remains static, but the volume of transactions per year increases. In the other case, the number of tuples remains fixed, but the domains used to generate them are expanded. For example, there could be a new type of store introduced in a retail data set, or it could cover a longer historical period. Clearly there are valid reasons for both types of scaling within a dataset, just as there are valid reasons to stress a hardware system to highlight particular features or concerns, and often a test will employ both approaches to expanding the dataset.

In the case of TPC-DS, the choice was made to use a hybrid approach. Most table columns employ dataset scaling instead of domain scaling, especially fact table columns. Some columns in small tables employ domain scaling. The domains have to be scaled down to adjust for the lower table cardinality. For instance the domain for *county* is approximately 1800. At scale factor 100 there exist only about 200 stores. Hence the county domain had to be scaled down for stores.

Fact tables scale linearly with the scale factor while dimensions scale sub-linearly. Consequently, the unrealistic scaling issues that TPC-H is encountering are avoided. Even at larger scale factors the TPC-DS number of customers, items, etc. remain realistic. The following table shows the scaling of some of the fact tables and dimensions. At scale factor 100, the database contains approximately 100GB of raw data. That is, 58 Million items[5] are sold per year by 2 Million customers in 200 stores. On average each shopping card contains 10.5 items. For detailed scaling information of all tables refer to the public release of the preliminary draft [3].

Table	**Rowcounts**			
	100GB	**1TB**	**10TB**	**100TB**
store_ sales	288M	2.9B	30B	297B
store_ returns	14M	147M	1.5B	15B
store	200	500	750	1,500
customer	2M	8M	20M	100M
Items	200K	300K	400K	500K

Table 2: Table Cardinalities (K=10^3, M=10^6)

3.2. Synthetic vs. Real Data Set

Synthetic data sets that are built using well studied distributions such as the Normal or the Poisson distributions have many advantages. They certainly challenge the optimizer and data placement algorithms. They are mathematically very well defined and easy to implement in a data generator. One could even find parameters for distributions that simulate quite closely real world distributions, such as sales by season. However, they have one inherent problem. They do not work very well for decision support benchmarks that need to dynamically substitute bind variables.

Bind variables are used to make a benchmark less predictable and to cover the entire dataset as opposed to a subset that is defined by query predicates. Refer to [10] for a detailed explanation of the benefits of bind variables in a benchmark. Contrary to this goal is the necessity that the workload be identical every time the benchmark is run. Otherwise results are not comparable. For SQL queries, this means that only those substitutions are permissible that keep the following nearly identical:

[3] Referencing is well defined in the TPC-DS specification [3].

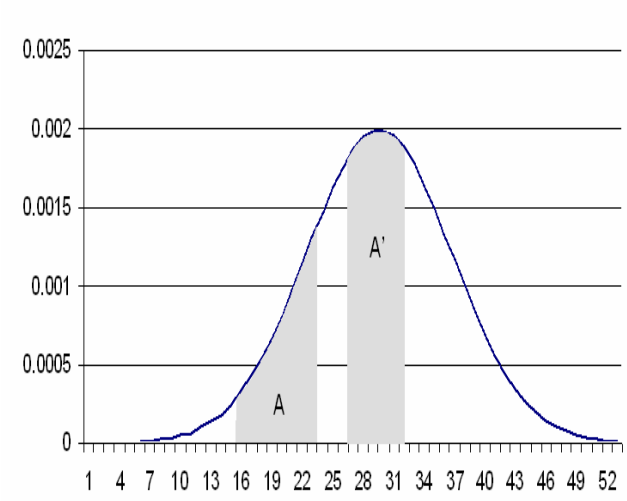
[4] Raw size of flat files as created by the data generator.

[5] Each row in the sales fact table represents the purchase of one item.

1. Number of qualifying rows in all tables,
2. Distribution of all primary and foreign keys involved in joins,
3. Distribution of group by columns and
4. Distribution of order by columns.

The number of qualifying rows must be nearly identical since it influences the number of rows that need to be fetched. This is especially important if data is clustered or partitioned using selectivity predicates. The distribution of primary and foreign keys influences the performance of join algorithms. For instance, the number of unique keys used for building the hash table determines the amount of storage (main memory and disk) and potentially the number of passes over the data. Similarly, the distribution of *group by* and *order by* columns influences performance of sort algorithms. Unfortunately, some of the above goals are conflicting, if a column participates in more than one role. For instance when a column participates as a selectivity predicate and as a group by predicate not both goals can be met. In addition to the above rules the dataset must not have any correlations between columns. Correlations introduce side effects which make queries incomparable.

Let's assume the synthetic sales date distribution in Figure 3. This distribution shows how sales are distributed over time (weeks): Sales are very low in the first weeks and then ramp up gradually to peak in Week 28 before they slow down gradually towards the end of the year. Ignoring the fact that sales tend to peak right before the holiday season, this distribution captures quite closely today's sales behavior.



$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$ with μ=200 and σ=50

Figure 3: Synthetic Sales Distribution

Now let's consider the following SQL query:

```
SELECT s_date, sum(s_sales)
FROM sales
WHERE s_date between D1 and D2
GROUP BY s_date;
```

Figure 4: Simple query Q1 with date predicates

In this example the selectivity predicate and a group by predicates are both on the same column. One can either determine (D1, D2) pairs such that the number of qualifying rows are identical or the key distribution.

Following the above rules, TPC-DS for a majority of its data employs traditional synthetic distributions, yielding uniformly distributed integers, or word selections with a Gaussian distribution. For a number of crucial distributions, however, TPC-DS synthesizes real world data to create so called comparability zones. Data in comparability zones have a uniform distribution. Substitutions for each of the selectivity, group by, and order by predicates in the TPC-DS query set, are such that they always pick values from the same comparability zone.

As an example distribution Figure 2 shows the store sales distribution for each year as used in TPC-DS. It mimics the census sales distribution [12] by defining three comparability zones; 1) January to July; 2) August to October; 3) November to December. The census distribution is indicated by the diamond graph while the TPC-DS distribution is indicated by the square graph. Domain values in the first zone occur with a low likelihood in the

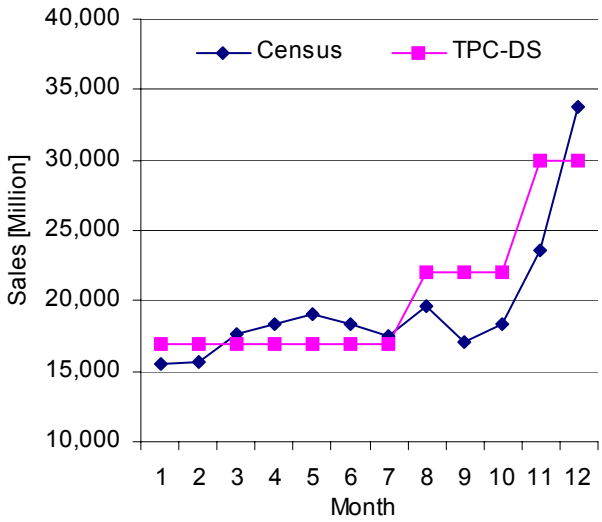


Figure 2: Store Sales Distribution

data set (low zone), domain values in the second zone occur with a medium likelihood, and domain values in the third zone occur with a high likelihood in the dataset. The data generator guarantees that all domain values in one domain have the same likelihood. The query generator is aware of this distribution. For each query involving predicates on sales date a decision was made whether a query targets Zone 1, Zone 2 or Zone 3. Based on these decisions the query generator generates queries such that each substitution guarantees query comparability.

3.3. Specific Multi Dimensional Issues in TPC-DS's Data Populations

3.3.1. Hierarchies

The hierarchies defined in the TPC-DS schema all display simple, single-inheritance. That is, there is always exactly one parent for any given level in the hierarchy. With this assurance, and a set cardinality for each level of a given hierarchy, the data generation becomes straightforward. Figure 5 shows the traditional item hierarchy. In TPC-DS each Brand belongs to exactly one Class and each class belongs exactly to one Category.

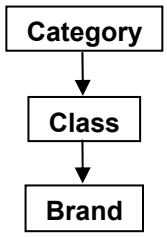


Figure 5: Traditional Item Hierarchy

3.3.2. Slowly Changing Dimensions

In addition to the transaction-focused fact tables and attribute-focused dimension tables described above, a typical multi-dimensional system includes some dimensions whose data evolves over the life of the system. Referred to as slowly changing dimensions (SCD), they capture the historical evolution of a data set. Each entity in a SCD can change attributes. As an example, consider a system that tracks retail sales over a period of months or years. During that time, the underlying product line, pricing structure, sales region geography – virtually every part of a transaction's context – is likely to change. In order for meaningful analysis to be possible, it is often important that the user of the data set be able to recreate that context – comparing sales using the old pricing model with those using the new, for example. Though there are a number of ways to address this sort of data within the dimensional data warehouse [6][7], a common technique is to include versioning information, often in the form of begin and end dates, in the dimension tables. A query can then qualify which revision of a dimension entry should be used to probe the fact tables. In TPC-DS the initial data population, meaning before any data maintenance is performed, contains the effects of previous data maintenance operations. That is, in SCD, there are up to 3 revisions of any dimension entry. This is important as the second performance run, as described in Section **Error! Reference source not found.**, serves as a repetition of the first performance and, therefore, ought to have the same data characteristics.

4. WORKLOAD

TPC-DS benchmark models the two most important components of any mature decision support system: user queries and data maintenance. The queries convert operational facts into business intelligence while the data maintenance operations synchronize the operational side of a business with the data warehouse. Taken together, the combination of these two workloads constitutes the core competencies of any decision support system. The TPC has carefully evaluated the TPC-DS workload to provide a robust, rigorous and complete means for the evaluation of systems meant to provide that competency.

4.1. Query workload

In order to address the enormous range of query types and user behaviors encountered by a decision support system, TPC-DS utilizes a generalized query model. This model allows the benchmark to exercise important aspects of the interactive and iterative nature of on-line analytical processing (OLAP) queries, the longer-running, complex queries of data mining and knowledge discovery, and the more planned behavior of well known report queries. The queries modeled by the benchmark cover:

- Ad-hoc, reporting, iterative OLAP and data mining type workloads,
- DSS relevant SQL99 functionality,
- A variety of access patterns, query phrasings, operators, and answer set constraints,
- Possibility of a wide variety to query optimizations,
- The entire data set of all TPC-DS tables and
- Complex DSS business problems.

Amalgamating ad-hoc, reporting, iterative OLAP and data mining queries into the same benchmark is a difficult task. Ad-hoc queries per se cannot exist in a benchmark environment because, for a benchmark to be fair and repeatable, it has to run the same queries on the same dataset every time it is executed. Consequently, a database administrator (DBA) with the prior knowledge about a benchmark's queries and dataset can tune the system to increase performance of the benchmark. In real life, prior knowledge of a dataset is limited to a certain point in time while knowledge of a query set is limited to special types of queries. A DBA can investigate the dataset, but as businesses change, the dataset, especially distributions are likely to change. Reporting and data mining queries might be known in advance, while ad-hoc and iterative OLAP queries can be issued at any time against the database by online users and, hence, their tuning is limited to general assumption of the workload.

TPC's previous DSS benchmark, TPC-H, prohibits constructs and mechanisms that would only realistically be beneficial in a reporting environment. It does that by prohibiting the exploitation of data and query set characteristics beyond a certain limit. For instance, no pre-join and pre-aggregation of tables can be executed and only single column indexes can be defined. Although artificial this method has proven to be very robust. For detailed wording, refer to the TPC-H specification [4], Chapter 1.5 "Implementation Rules".

TPC-DS applies this technique to a portion of the schema, dividing it into two parts, an ad-hoc part and a reporting part. Queries against the ad-hoc part are, per definition, pure ad-hoc queries, while queries against the reporting part are pure reporting queries. Queries which reference both parts are considered hybrid queries. For the specific wording refer to Clause 2.6 "Implementation Requirements" of the preliminary draft [3]. Iterative OLAP queries are implemented as a sequence of syntactically independent, but logically affiliated queries. Data Mining queries are characterized as returning a large output. This output, although not in the scope of TPC-DS, is intended for feeding data mining tools. Both Iterative OLAP and Data Mining queries can be classified as either ad-hoc or reporting.

The TPC-DS query set has been developed to cover most of the syntax of SQL99 including its OLAP amendment. This includes:

SQL states the problem rather than defining the exact execution path of a program. This gives the query optimizer great opportunities to find the best possible execution plan, that is, the best possible execution path, which may vary between hardware platforms. Since a SQL query cannot mandate a certain execution path, the queries have to be numerous and diverse enough to cover the entire spectrum of execution paths in today's DSS. It is out of the scope of this paper to describe all execution paths. However, the most important ones are index driven operations (e.g. Bitmap operations), table scan driven operations (e.g. hash-joins), Sort and Group By operations. As described in Section 2 the schema supports both star schema queries as well as the more traditional 3NF operations. The following SQL statements are examples,

taken from TPC-DS, to demonstrate ad-hoc and reporting queries. Figure 6 shows Query 52, an Ad-Hoc query computing the sum of the extended sales price for all items sold in a year, grouped by brand. It references store_sales, item and date_dim. Figure 7 shows Query 20, a reporting query computing for each item in a list of given subcategories, during a specific time period, in the catalog sales channel, the ratio of sales of that item to the sum of all of the sales in that item's class. It references the catalog_sales, item and date_dim tables. For a detailed description of the remaining queries refer to Appendix A of the preliminary draft [3].

```
SELECT dt.d_year ,item.i_brand_id brand_id
      ,item.i_brand brand
      ,SUM(ss_ext_sales_price) ext_price
FROM date_dim dt ,store_sales ,item
WHERE dt.d_date_sk = store_sales.ss_sold_date_sk
  AND store_sales.ss_item_sk = item.i_item_sk
  AND item.i_manager_id = 1
  AND dt.d_moy=11
  AND dt.d_year=2000
GROUP BY dt.d_year,item.i_brand,item.i_brand_id
ORDER BY dt.d_year,ext_price desc,brand_id;
```

Figure 6: Query 52, Ad-Hoc Query Example

```
SELECT i_item_desc,i_category,
       i_class,i_current_price
      ,SUM(cs_ext_sales_price) AS itemrevenue
      ,SUM(cs_ext_sales_price)*100/
           SUM(SUM(cs_ext_sales_price)) OVER
           (PARTITION BY i_class)AS revenueratio
FROM catalog_sales,item,date_dim
WHERE s_item_sk = i_item_sk
  AND i_category in ('Sports', 'Books', 'Home')
  AND cs_sold_date_sk = d_date_sk
  AND d_date BETWEEN '1999-02-21'AND'1999-03-21'
GROUP BY i_item_id,i_item_desc,i_category
        ,i_class,i_current_price
ORDER BY i_category,i_class,i_item_id
       ,i_item_desc,revenueratio;
```

Figure 7: Query 20, Reporting Query Example

The query set is designed to cover the entire dataset. This is guaranteed by a sophisticated query template model. Template-based queries are defined as sets of one or more pseudo-random, valid SQL statements produced at the time of benchmark execution. Template-based queries are intended to model common, well-understood queries. It is assumed that the precise values or targets of a given instance of a template-based query is random, but that the general format and syntax for the query is tightly tied to a business process and the syntax is therefore largely predictable and well-known. A template-based query relies on a query template by substituting SQL fragments and scalar constants into the query template to produce a set of one or more valid SQL statements that are then submitted to the SUT. There are numerous types of substitutions. They include filter predicates, such as *equality*, *in-list* and *between* predicates. More complex text substations are also possible, such as exchanging aggregations, such as max, min. The query generator, responsible for choosing the substitutions and the data generator are tightly coupled to guarantee query comparability across substitutions. It is expected that there is some run-to-run variability on a per query basis. However, since the main metric is an arithmetic mean, it has been proven that such variability does not result in any significant metric variability. Please refer to [10] for a detailed description of how the query generator generates executable SQL queries from query templates.

4.2. Data Maintenance workload

TPC-DS highlights the ability of a system to absorb new database data as a decision support system grows. Whether the business intelligence is drawn from existing operational systems or enhanced through the integration of external data sources, such as geographical data, it must be able to respond to additions and modifications to its underlying data in a timely and cost effective manner. By focusing on the fundamental SQL-based transformations upon which all data manipulations rely, TPC-DS provides the first industry-standard evaluation of the ETL process (Extract, Transformation and Load).

A periodic data refresh process is an integral part of the data warehouse lifecycle inherent to most decision support environments. In comparison to previous TPC benchmarks, which emphasize the data analysis component of decision support systems, TPC-DS offers a more balanced importance to a realistic refresh process as part of the benchmark. Decision support database refresh processes usually involve three distinct and important steps: data Extraction, data Transformation, and data Load (ETL). The data extract step accomplishes just that; the accurate extraction of pertinent data from production OLTP databases or other relevant data sources. In the transformation step, the extracted data are typically cleansed and massaged into a common format suitable for assimilation by the decision support database. Lastly, the data load step performs the actual insertion, modification and deletion of decision support database table data. In a production system environment, the data extraction step may be comprised of numerous separate extract operations, executed against multiple OLTP databases and ancillary data sources. As it is unlikely that the full compliment of these OLTP data sources reside on the decision support server(s), it is doubtful the measurement of OLTP data extraction performance would result in a metric appropriate or meaningful to the scope of the TPC-DS benchmark. In light of this, the data extraction step of the ETL process (E) is assumed and represented in the benchmark in the form of generated flat files.

In addition to a surrogate key, which is the primary key, each dimension contains a "business key". The business key resembles the primary key from the OLTP system. The update data can be joined with the data warehouse dimensions using the business key. History keeping dimensions contain two additional fields, a rec_begin_date and a rec_end_date indicating the date range for which a specific row is valid. The row containing NULL in the rec_end_date for a specific business key is the most current row.

The data maintenance workload contains the updating of dimension rows, the inserting and the deleting of fact table rows. Dimensions are categorized in static, non-history keeping and history keeping dimensions. Static dimensions such as Date_dim, Time_dim, and Reason are loaded once at the beginning of the benchmark and are not updated during the data maintenance phase.

The algorithms in Figure 8 and Figure 9 show how history and non-history keeping dimensions are maintained. The intention is to cover the most widely used data maintenance operations. Requiring both types of history operations, enables the performance for both insert and update operations to be measured.

```
for every row to be updated{
  find the row for the business key
  update all changed fields
}
```

Figure 8: Update non-history keeping dimension

```
for every row to be updated{
  find the row for the business key and
       with rec_end_date = NULL
  insert current date into rec_end_date
  insert new row with update date and
       set rec_end_date to NULL
}
```

Figure 9: Update history keeping dimensions

Fact table data are deleted and inserted in a logically clustered fashion. According to a randomly picked data range, fact table data are deleted and substituted with similar data during the insert phase. The data are clustered on a date such that performance of drop partition and data insertion into partitions can be measured. During the insert of fact table data the business keys from the input flat files need to be converted into surrogate keys. In order to find the most up to date surrogate key, the input data are joined with dimension data. Figure 10 shows the pseudo code for fact table insert operations. For a detailed description on the data maintenance operations refer to Chapter 5 of the preliminary draft [3].

```
for every row to be inserted{
   for keys to a non-history keeping dimension{
      find the row for the business key
      exchange business key with surrogate key
   }
   for keys to a history keeping dimension{
      find the row for the business key
           and where rec_end_date is NULL
      exchange business key with surrogate key
   }
   insert row into fact table.
}
```

Figure 10: Fact Table Insert

5. EXECUTION RULES AND METRIC

The execution rules and the metric are two fundamental components of any benchmark definition and are probably the most controversial when trying to reach an agreement between different companies. The execution rules define the way a benchmark is executed, while the metric emphasizes the pieces that are measured. We describe them in one section since they are intrinsically connected to each other and they are equally powerful in how they control performance measurements. Both components can change the focus of a benchmark because only those parts of a system that are executed, as described in the execution rules, can be measured in the metric. Conversely, even though a part is executed, if it is not part of the metric, it goes un-noticed. For instance, TPC-H's execution rules mandate the measurement of the initial database load. However, the primary metric (QphH) does not take the load metric into account. Consequently, little consideration is given to it when running the benchmark.

5.1. General Metric Considerations

TPC is best known for providing robust, simple and verifiable performance data. The most visible part of the performance data is the performance metric and the rules that lead to it. Producing benchmark results is expensive and time consuming. Hence, the TPC's goal is to provide a robust performance metric which allows for system performance comparisons for an extended period and, thereby, preserving benchmark investments. A performance metric needs to be simple such that easy system comparisons are possible. If there are multiple performance metrics (e.g. A, B, C), system comparisons are difficult because vendors can claim they perform well on some of the metrics (e.g. A and C). This might still be acceptable if all components are equally important, but without this determination, there would be much debate on this issue. In order to unambiguously rank results, the TPC benchmarks focus on a single primary performance metric, which encompass all aspects of a system's performance weighting the individual components. Taking the example from above the performance metric M is calculated as a function of the three components A,B and C (e.g. M=f(A,B,C)). Consequently, TPC's performance metrics measure system and overall workload performance rather than individual component performance. In addition to the performance metric, the TPC also includes other metrics, such as price-performance metrics.

One of the key essentials to the success of a benchmark is a sound metric. In the process of benchmark development the measurable components (e.g. query elapsed time) and variables (e.g. scale factor) were analyzed in respect to their impact to the metric.

5.2. Execution Rules

The TPC-DS workload is expected to test the upward boundaries of hardware system performance in the areas of CPU, memory and I/O subsystem utilization. At the same time it measures the database software and operating system's ability to perform various complex functions, important to DSS, such as examining large volumes of data, computing and executing the best execution plan for queries with a high degree of complexity, efficient scheduling of a large number of user sessions, and giving answers to critical business questions.

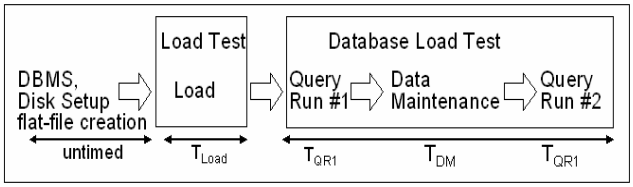


Figure 11: Benchmark Execution Order

The benchmark test is defined as the execution of a database load test followed by a performance test (see Figure 11). The process of building the test database is denoted as database load. It is the elapsed time to create the tables, load data, create auxiliary data structures, define and validate constraints, gather statistics for the test database and configure the system under test as it will be during the performance test. It is also important to note that all requirements to assure ACID properties including synchronizing loaded data on RAID devices and the taking of a backup of the database, if necessary, are part of the load test. Some portions of a database load are not timed, such as creating the database, preparing tablespaces and generating the data to be loaded. However, the intent is to include all activity required to bring the system under test to the configuration that immediately precedes the beginning of the performance test.

The performance test consists of two query runs and one data maintenance run. The first query run (Query Run 1) measures the query execution power of the system immediately after it is loaded. The data maintenance run measures the system's ability to load, delete and update data and to maintain auxiliary data structures. The second query (Query Run 2) measures the query execution power after the system has been updated and auxiliary data structures have been maintained, thereby, revealing any query performance changes due to the maintenance of auxiliary data structures. Without including a second query run it would be possible to avoid or defer maintenance of auxiliary data structures, thereby, not including their time in the measured interval.

Each query run executes multiple concurrent query streams. Concurrent query streams simulate multiple users executing que-

ries against the database concurrently. Each stream simulates one user. The query mix is designed to provide varying degrees of workload complexity and the concurrent execution of queries in query streams emulates typical workload characteristic of many decision support systems today. The maximum number of streams is not limited. However, each scale factor requires a minimum number of streams, denoted in Figure 12. Linking the minimum number of streams to the scale factor requires that larger systems not only execute queries on more data, but also serve more concurrent user.

Scale factor	Minimum Number of Streams
100	3
300	5
1000	7
3000	9
10000	11
30000	13
100000	15

Figure 12: Minimum Required Query Streams

5.3. Primary Metrics

TPC-DS defines three primary metrics: Performance Metric (QphDS@SF), Price-Performance metric, ($/QphDS@SF) and System availability date. The Performance Metric reflects the effective query throughput of the benchmarked configuration, defined as:

$$QphDS@SF = SF * 3600 \frac{198 * S}{(T_{QR1} + T_{DM} + T_{QR2} + 0.01 * S * T_{Load})}$$

- T_{QR1}: total elapsed time of Query Run 1.
- T_{QR2}: total elapsed time of Query Run 2.
- T_{DM}: total elapsed time of the Data Maintenance run.
- T_{Load}: total elapsed time of the database load test.
- S: number of streams the benchmark executed.
- SF: scale factor.

The numerator represents the total number of queries executed on the system "198 * S", where 198 is the 99 individual queries times two query runs. For instance, a 1000 scale factor benchmark test with minimum number of required query streams executes 1386 (198 * 7 streams) queries. A 10,000 scale factor benchmark test with a minimum number of required query streams of 11 executes 2970 (198 *15) queries. The TPC DS subcommittee believes that workloads with such a large number of queries insure an adequate DSS workload for todays and future high performance systems.

The denominator represents the total elapsed time as the sum of Query Run1, Data Maintenance Run, Query Run 2 and a fraction of the Load Time. By dividing the total number of queries by the total elapsed time, this metric represents queries executed per time period. Unlike previous TPC decision support benchmarks, this metric does not include a power test. A power test is also known as single user test. The power metric in previous benchmarks is defined as the geometric mean of the query response time of all queries including the time to complete the update functions. The queries and update functions are run sequentially. For this kind of metric, it is crucial to tune each and every query to get the best results. A reduction of elapsed time for a query from 6 hours to 2 hours has the same effect on the metric as reducing a query from 6 seconds to 2 seconds – which is a major weakness of the metric. We believe that in most real life situations the reduction of elapsed time of a query from 6 hours to 2 hours is much more important than from 6 seconds to 2 seconds. In the absence of a power metric, engineers will concentrate their effort in tuning long running queries, which matches the business case, because that is where customers spend most of their tuning resources. It is important to note that the minimum number of query streams provides a highly concurrent workload. This ensures that tuning of just one query does not significantly impact the primary metric, but only improving the overall system performance will improve the metric significantly.

The load time is part of the denominator to realistically limit the use of auxiliary structures without disallowing them. As mentioned in earlier sections, the usage of auxiliary data structures is allowed for one of the sales channels, namely the catalog sales channel, which represents 25% of the data set. This enables database vendors to showcase their technologies in handling queries that are primarily reporting in nature. But there is a cost associated to using these auxiliary data structures. If unlimited auxiliary structures were allowed without penalty, as it was the case in TPC-D, the benchmark metric would not grow linearly with the resources used. This behavior causes a benchmark less interesting to hardware vendors, who desire to show that higher performance systems will also have the highest benchmark performance metric. Consequently, the subcommittee decided to factor the database load and the data maintenance times into the primary performance metric. The fraction of the load time is multiplied by the number of streams. This is necessary to avoid diminishing the impact of the load time on the metric. Without considering the scale factor one could decrease the impact of the load time by increasing the number of streams, thereby diminishing the overhead of creating auxiliary data structures. A 1000 scale factor benchmark test with minimum number of required streams will have 10% (0.01*10) of the database load time added to the total elapsed time. The authors acknowledge that although 10% might sound arbitrary, the purpose of a benchmark metric is to provide impartial information that can be used to evaluate and compare the performance of DSS systems. It is not the intention of a benchmark metric to constitute a perfect simulation of a particular DSS environment.

Finally, the metric is normalized to queries per hour by multiplying the results by 3600, and on scale factor by multiplying the metric by the scale factor. Normalization to queries per hour is necessary since the workload represents both ad-hoc and reporting queries, which can include very long running queries. A query per second metric might result in fractional values, which is not desirable. Normalization by scale factor is done for two reasons; factoring in the complexity of running the benchmark workload on a larger data set and increasing the marketability of the benchmark. The complexity of the workload in terms of the amount of data that needs to be scanned as well as the minimum required query streams increases as the scale factor increases. Marketing teams would like to see larger benchmark results at larger scale factors. For example, assuming ideal scalability; if a system performs 100 queries per hour on a 100 scale factor database; the same setup will only run 10 queries per hour at a 1000 scale factor database. This is not desired by marketing teams. They would like to see the same number of queries per hour. As a result, the metrics are normalized based on scale factors.

The Price-Performance metric is defined as the ratio between the 3 year total cost of ownership (TCO) of the system and the primary metric (queries per hour). The 3 year TCO includes hardware, software and 24 x 7 maintenance with a 4 hour response time which customers would pay in a real sales situation for similarly sized configurations. The price-performance metric for the benchmark is defined as:

$$\$/QphDS@SF = \frac{P}{QphDS@SF}$$

P is the 3 year total cost of ownership of the configuration. Complete pricing guidelines for TPC benchmarks are available from the TPC web site [5]. This specification guides customers, vendors implementing a benchmark, and auditors on what is acceptable pricing for the purposes of publication. These restrictions are intended to make publication both tractable and comparable during the lifetime of the publication for the majority of customers and vendors.

The third primary metric, the system availability date, is the date when the system is generally available to any customer.

6. CONCLUSION

The new TPC-DS benchmark is intended to provide a fair and honest comparison of various vendor implementations to accomplish an identical, controlled and repeatable task in evaluating the performance of DSS systems. The TPC-DS workload is expected to test the upward boundaries of hardware system performance in the areas of CPU utilization, memory utilization, I/O subsystem utilization and the ability of the operating system and database software to perform various complex functions important to DSS - examine large volumes of data, compute and execute the best execution plan for queries with a high degree of complexity, schedule efficiently a large number of user sessions, and give answers to critical business questions. In this paper we have introduced the pivotal parts of the benchmark, such as schema, data set, workload, metric and execution rules. We have also explained the reasons behind key decisions made for TPC-DS. For a more comprehensive description of the TPC-DS benchmark, the entire specification can be obtained from the TPC website [3].

The TPC is expected to vote on this benchmark in 2006. In order to protect investments made in TPC-H benchmarks the TPC is expected to allow TPC-H benchmark publications until a large body of TPC-DS results are established, which typically takes one to two years.

7. ACKNOWLEDGEMENT

The authors would like to acknowledge Mike Nikolaiev, Ray Glasstone, Umesh Dayal, Ken Jacobs, Tracey Stewart, Christopher Buss and Ivan Mcphee for their comments and feedback and all TPC-DS subcommittee members, especially Susanne Englert, Alain Crolotte, Mary Meredith, Sreenivas Gukal, Doug Johnson, Lubor Kollar, Murali Krishna, Robert Lane, Juergen Mueller, Robert Murphy, Doug Nelson, Ernie Ostic, Haider Rizvi, Kwai Wong, Bryan Smith, Eric Speed, Cadambi Sriram, Jack Stephens, John Susag, Tricia Thomas, Dave Walrath, Gene Purdy, Priti Mishra, Anoop Sharma and Guogen Zhang.

8. REFERENCES

[1] Robert H. Bonczek, Clyde W. Holsapple, and Andrew Whinston: *Foundations of Decision Support Systems*. Academic Press, 1981 ISBN 0-12-113050-9

[2] Gordon B. Davis: Management Information Systems: *Conceptual Foundations, Structure and Development*. McGraw-Hill, 1974

[3] Public release of TPC-DS (v0.32) preliminary draft: http://www.tpc.org/tpcds/default.asp.

[4] TPC-H Specification 2.3.0: http://www.tpc.org/tpch/default.asp

[5] TPC pricing specification. http://www.tpc.org/pricing/spec/Price_V1.0.1.pdf

[6] William H. Inmon: *EIS and the Data Warehouse, Data Base Programming/Design*, November 1992.

[7] Ralph W. Kimball, Warren Thornthwaite, Laura Reeves and Margy Ross: *The Data Warehouse Lifecycle Toolkit*, New York, NY: John Wiley and Sons, 1998

[8] Meikel Pöss, Bryan Smith, Lubor Kollár, Per-Åke Larson: *TPC-DS, taking decision support benchmarking to the next level*. SIGMOD Conference 2002: 582-587

[9] Meikel Pöss, Chris Floyd: *New TPC Benchmarks for Decision Support and Web Commerce*. SIGMOD Record 29(4): 64-71 (2000)

[10] Meikel Poess, John M. Stephens: *Generating Thousand Benchmark Queries in Seconds*. VLDB 2004: 1045-1053

[11] John M. Stephens, Meikel Poess: *MUDD: a multi-dimensional data generator*. WOSP 2004: 104-109

[12] US Census Bureau, *Unadjusted and Adjusted Estimates of Monthly Retail and Food Services Sales by Kinds of Business:2001*, Department stores (excl.L.D) 4521. http://www.census.gov/mrts/www/data/html/ nsal01.html