# TPC BENCHMARK™ H

(Decision Support)
Standard Specification
Revision 2.3.0

Transaction Processing Performance Council (TPC)
777 N. First Street, Suite 600
San Jose, CA 95112-6311, USA
Phone (408) 295-8894
Fax (408) 295-9768
http://www.tpc.org
e-mail:admin@tpc.org

## Acknowledgments

The TPC acknowledges the work and contributions of the TPC-D subcommittee member companies in developing Version 2 of the TPC-D specification which formed the basis for TPC-H Version 1. The subcommittee included representatives from Compaq, Data General, Dell, EMC, HP, IBM, Informix, Microsoft, NCR, Oracle, Sequent, SGI, Sun, Sybase, and Unisys. The TPC also acknowledges the contribution of Jack Stephens, consultant to the TPC-D subcommittee, for his work on the benchmark specification and DBGEN development.

## TPC Membership

(as of June, 2005)

Adaptec

Bull S.A.

Computer Associates
Data General Corp.

Dell Computer Corporation

EDS

Fujitsu/ICL
Hewlett-Packard
Hitachi SW
IBM Corp.

Intel Corporation.

Microsoft Corporation.
.

NCR
NEC Systems Laboratory

Novell

Oracle Corporation
.

Fujitsu-Siemens
Silicon Graphics

Sun Microsystems
Sybase

Toshiba Corp.

Unisys Corporation

**Document History**

| Date | Version | Description |
|---|---|---|
| 26 February 1999 | Draft 1.0.0 | Mail ballot draft for Standard Specification |
| 24 June 1999 | Revision 1.1.0 | First minor revision of the Specification |
| 25 April 2002 | Revision 1.4.0 | Clarification about Primary Keys |
| 12 July 2002 | Revision 1.5.0 | Additions for EOL of hardware in 8.6 |
| 15 July 2002 | Revision 2.0.0 | Mail ballot draft 3 year maintenance pricing |
| 14 August 2003 | Revision 2.1.0 | Adding scale factors 30TB and 100TB |
| 29 June 2005 | Revision 2.2.0 | Adding Pricing Specification 1.0.0 |
| 11 August 2005 | Revision 2.3.0 | Changing pricing precision to cents and processor definition |

# Table of Contents

# 0: INTRODUCTION

## 0.1 Preamble

The TPC Benchmark™H (TPC-H) is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance while maintaining a sufficient degree of ease of implementation. This benchmark illustrates decision support systems that

- Examine large volumes of data;
- Execute queries with a high degree of complexity;
- Give answers to critical business questions.

TPC-H evaluates the performance of various decision support systems by the execution of sets of queries against a standard database under controlled conditions. The TPC-H queries:

- Give answers to real-world business questions;
- Simulate generated ad-hoc queries (e.g., via a point and click GUI interface);
- Are far more complex than most OLTP transactions;
- Include a rich breadth of operators and selectivity constraints;
- Generate intensive activity on the part of the database server component of the system under test;
- Are executed against a database complying to specific population and scaling requirements;
- Are implemented with constraints derived from staying closely synchronized with an on-line production database.

The TPC-H operations are modeled as follows:

- The database is continuously available 24 hours a day, 7 days a week, for ad-hoc queries from multiple end users and data modifications against all tables, except possibly during infrequent (e.g., once a month) maintenance sessions;
- The TPC-H database tracks, possibly with some delay, the state of the OLTP database through on-going refresh functions which batch together a number of modifications impacting some part of the decision support database;
- Due to the world-wide nature of the business data stored in the TPC-H database, the queries and the refresh functions may be executed against the database at any time, especially in relation to each other. In addition, this mix of queries and refresh functions is subject to specific ACIDity requirements, since queries and refresh functions may execute concurrently;
- To achieve the optimal compromise between performance and operational requirements, the database administrator can set, once and for all, the locking levels and the concurrent scheduling rules for queries and refresh functions.

The minimum database required to run the benchmark holds business data from 10,000 suppliers. It contains almost ten million rows representing a raw storage capacity of about 1 gigabyte. Compliant benchmark implementations may also use one of the larger permissible database populations (e.g., 100 gigabytes), as defined in Clause 4.1.3.

The performance metric reported by TPC-H is called the TPC-H Composite Query-per-Hour Performance Metric (QphH@Size), and reflects multiple aspects of the capability of the system to process queries. These aspects include the selected database size against which the queries are executed, the query processing power when queries

are submitted by a single stream, and the query throughput when queries are submitted by multiple concurrent users. The TPC-H Price/Performance metric is expressed as $/QphH@Size. To be compliant with the TPC-H standard, all references to TPC-H results for a given configuration must include all required reporting components (see Clause 5.4.6). The TPC believes that comparisons of TPC-H results measured against different database sizes are misleading and discourages such comparisons.

The TPC-H database must be implemented using a commercially available database management system (DBMS) and the queries executed via an interface using dynamic SQL. The specification provides for variants of SQL, as implementers are not required to have implemented a specific SQL standard in full.

TPC-H uses terminology and metrics that are similar to other benchmarks, originated by the TPC and others. Such similarity in terminology does not in any way imply that TPC-H results are comparable to other benchmarks. The only benchmark results comparable to TPC-H are other TPC-H results compliant with the same revision.

Despite the fact that this benchmark offers a rich environment representative of many decision support systems, this benchmark does not reflect the entire range of decision support requirements. In addition, the extent to which a customer can achieve the results reported by a vendor is highly dependent on how closely TPC-H approximates the customer application. The relative performance of systems derived from this benchmark does not necessarily hold for other workloads or environments. Extrapolations to any other environment are not recommended.

Benchmark results are highly dependent upon workload, specific application requirements, and systems design and implementation. Relative system performance will vary as a result of these and other factors. Therefore, TPC-H should not be used as a substitute for a specific customer application benchmarking when critical capacity planning and/or product evaluation decisions are contemplated.

Benchmark sponsors are permitted several possible system designs, provided that they adhere to the model described in Clause 6. A full disclosure report (FDR) of the implementation details, as specified in Clause 8, must be made available along with the reported results.

**Comment 1:** While separated from the main text for readability, comments and appendices are a part of the standard and their provisions must be complied with.

**Comment 2:** The contents of some appendices are provided in a machine readable format and are not included in the printed copy of this document.

## 0.2 General Implementation Guidelines

The rules for pricing are included in the current revision of the TPC Pricing Specification Version 1 located at www.tpc.org.

The purpose of TPC benchmarks is to provide relevant, objective performance data to industry users. To achieve that purpose, TPC benchmark specifications require that benchmark tests be implemented with systems, products, technologies and pricing that:

- Are generally available to users;
- Are relevant to the market segment that the individual TPC benchmark models or represents (e.g., TPC-H models and represents complex, high data volume, decision support environments);
- Would plausibly be implemented by a significant number of users in the market segment the benchmark models or represents.

The use of new systems, products, technologies (hardware or software) and pricing is encouraged so long as they meet the requirements above. Specifically prohibited are benchmark systems, products, technologies or pricing

(hereafter referred to as "implementations") whose primary purpose is performance optimization of TPC benchmark results without any corresponding applicability to real-world applications and environments. In other words, all "benchmark special" implementations that improve benchmark results but not real-world performance or pricing, are prohibited.

The following characteristics shall be used as a guide to judge whether a particular implementation is a benchmark special. It is not required that each point below be met, but that the cumulative weight of the evidence be considered to identify an unacceptable implementation. Absolute certainty or certainty beyond a reasonable doubt is not required to make a judgment on this complex issue. The question that must be answered is: "Based on the available evidence, does the clear preponderance (the greater share or weight) of evidence indicate that this implementation is a benchmark special?"

The following characteristics shall be used to judge whether a particular implementation is a benchmark special:

a) Is the implementation generally available, documented, and supported?

b) Does the implementation have significant restrictions on its use or applicability that limits its use beyond TPC benchmarks?

c) Is the implementation or part of the implementation poorly integrated into the larger product?

d) Does the implementation take special advantage of the limited nature of TPC benchmarks (e.g., query profiles, query mix, concurrency and/or contention, isolation requirements, etc.) in a manner that would not be generally applicable to the environment the benchmark represents?

e) Is the use of the implementation discouraged by the vendor? (This includes failing to promote the implementation in a manner similar to other products and technologies.)

f) Does the implementation require uncommon sophistication on the part of the end-user, programmer, or system administrator?

g) Is the implementation (including beta) being purchased or used for applications in the market area the benchmark represents? How many sites implemented it? How many end-users benefit from it? If the implementation is not currently being purchased or used, is there any evidence to indicate that it will be purchased or used by a significant number of end-user sites?

**Comment**: The characteristics listed in this clause are not intended to include the driver or implementation specific layer, which are not necessarily commercial software, and have their own specific requirements and limitation enumerated in Clause 6. The listed characteristics and prohibitions of Clause 6 should be used to determine if the driver or implementation specific layer is a benchmark special.

## 0.3 General Measurement Guidelines

TPC benchmark results are expected to be accurate representations of system performance. Therefore, there are certain guidelines that are expected to be followed when measuring those results. The approach or methodology to be used in the measurements are either explicitly described in the specification or left to the discretion of the test sponsor.

When not described in the specification, the methodologies and approaches used must meet the following requirements:

- The approach is an accepted engineering practice or standard;
- The approach does not enhance the result;
- Equipment used in measuring the results is calibrated according to established quality standards;
- Fidelity and candor is maintained in reporting any anomalies in the results, even if not specified in the bench-

mark requirements.

**Comment:** The use of new methodologies and approaches is encouraged so long as they meet the requirements above.

# 1: LOGICAL DATABASE DESIGN

## 1.1 Business and Application Environment

TPC Benchmark™ H is comprised of a set of business queries designed to exercise system functionalities in a manner representative of complex business analysis applications. These querieshave been given a realistic context, portraying the activity of a wholesale supplier to help the reader relate intuitively to the components of the benchmark.

TPC-Hdoes not represent the activity of any particular business segment, but rather any industry which must manage, sell, or distribute a product worldwide (e.g., car rental, food distribution, parts, suppliers, etc.). TPC-H does not attempt to be a model of how to build an actual information analysis application.

The purpose of this benchmark is to reduce the diversity of operations found in an information analysis application, while retaining the application's essential performance characteristics, namely: the level of system utilization and the complexity of operations. A large number of queries of various types and complexities needs to be executed to completely manage a business analysis environment. Many of the queries are not of primary interest for performance analysis because of the length of time the queries run, the system resources they use and the frequency of their execution. The queries that have been selected exhibit the following characteristics:

- They have a high degree of complexity;
- They use a variety of access
- They are of an ad hoc nature;
- patterns; theyexamine a large percentage of the available data;
- They all differ from each other;
- They contain query parameters that change across query executions.

These selected queries provide answers to the following classes of business analysis:

- Pricing and promotions;
- Supply and demand management;
- Profit and revenue management;
- Customer satisfaction study;
- Market share study;
- Shipping management.

Although the emphasis is on information analysis, the benchmark recognizes the need to periodically refresh the database. 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, however, be able to support queries and refresh functions against all tables on a 7 day by 24 hour (7 x 24) basis.

While the benchmark models a business environment in which refresh functions are an integral part of data maintenance, the refresh functions actually required in the benchmark do not attempt to model this aspect of the business environment. Their purpose is rather to demonstrate the update functionality for the DBMS, while simultaneously assessing an appropriate performance cost to the maintenance of auxiliary data structures, such as secondary indices.

**Comment**: The benchmark does not include any test or measure to verify continuous database availability or particular system features which would make the benchmarked configuration appropriate for 7x24 operation. References

to continuous availability and 7x24 operation are included in the benchmark specification to provide a more complete picture of the anticipated decision support environment. A configuration offering less that 7x24 availability can produce compliant benchmark results as long as it meets all the requirements described in this specification.

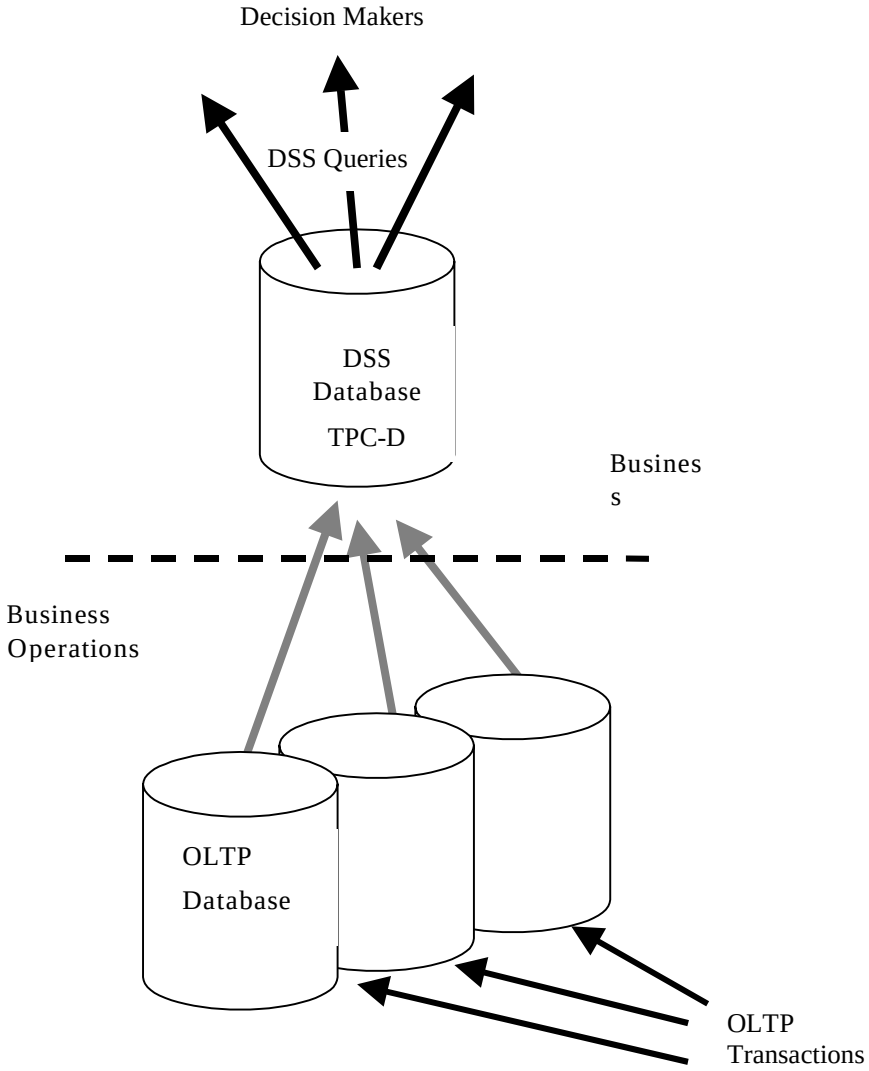



Figure 1: The TPC-H Business Environment illustrates the TPC-H business environment and highlights the basic differences between TPC-H and other TPC benchmarks.

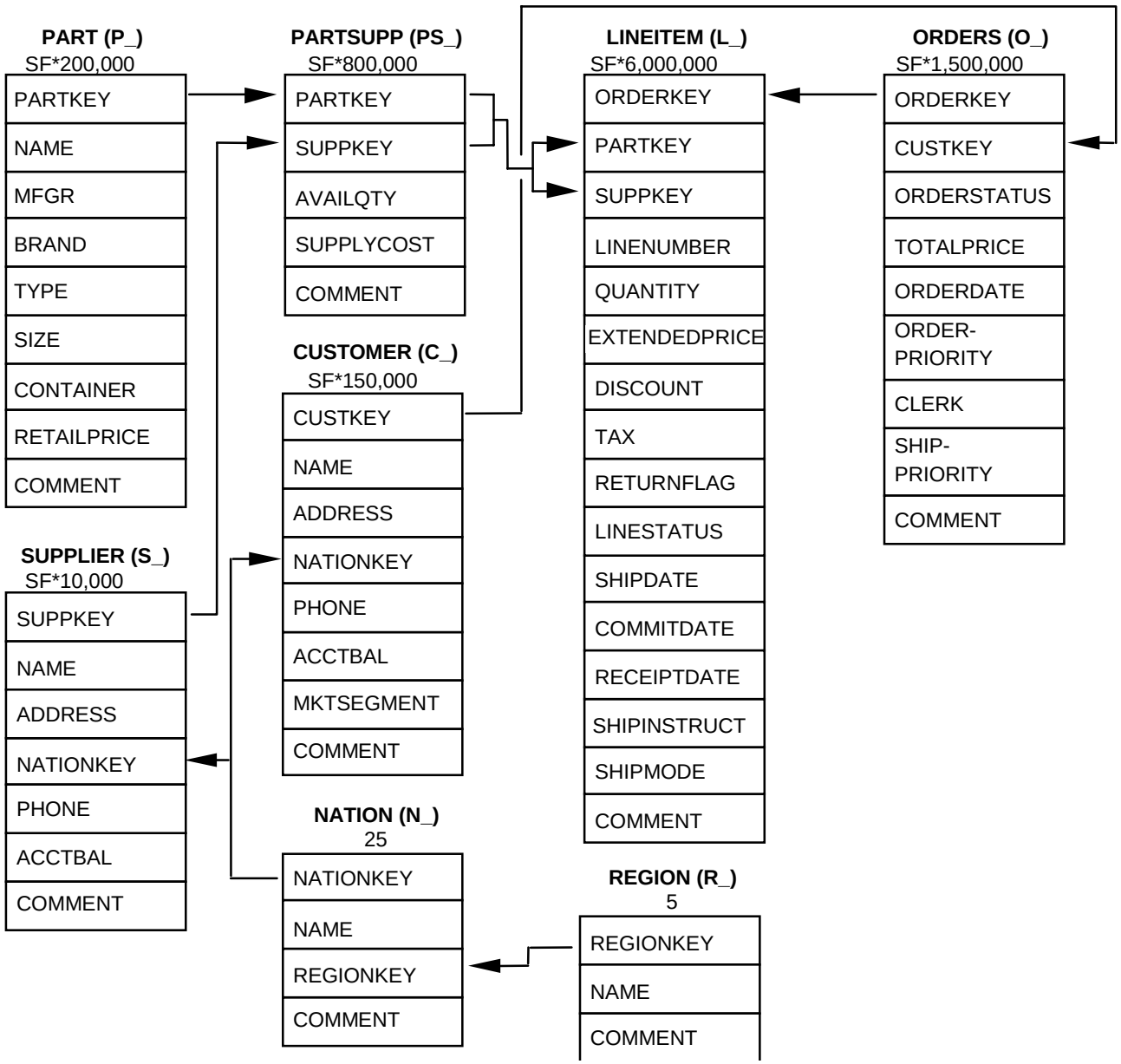
**Figure 1: The** TPC-H **Business Environment**

Other TPC benchmarks model the operational end of the business environment where transactions are executed on a real time basis. The TPC-H benchmark, however, models the analysis end of the business environment where trends are computed and refined data are produced to support the making of sound business decisions. In OLTP benchmarks the raw data flow into the OLTP database from various sources where it is maintained for some period of time. In TPC-H, periodic refresh functions are performed against a DSS database whose content is queried on behalf of or by various decision makers.

## 1.2 Database Entities, Relationships, and Characteristics

The components of the TPC-H database are defined to consist of eight separate and individual tables (the Base Tables). The relationships between columns of these tables are illustrated in Figure 2: The TPC-H Schema.

**Figure 2: The** TPC-H **Schema**




**Legend:**

- The parentheses following each table name contain the prefix of the column names for that table;
- The arrows point in the direction of the one-to-many relationships between tables;
- The number/formula below each table name represents the cardinality (number of rows) of the table. Some are factored by SF, the Scale Factor, to obtain the chosen database size. The cardinality for the LINEITEM table is approximate (see Clause 4.2.5).

## 1.3 Datatype Definitions

1.3.1 The following datatype definitions apply to the list of columns of each table:

- **Identifier** means that the column must be able to hold any key value generated for that column and be able to support at least 2,147,483,647 unique values;

**Comment**: A common implementation of this datatype will be an integer. However, for SF greater than 300 some column values will exceed the range of integer values supported by a 4-byte integer. A test sponsor may use some other datatype such as 8-byte integer, decimal or character string to implement the identifier datatype;

- **Integer** means that the column must be able to exactly represent integer values (i.e., values in increments of 1) in the range of at least -2,147,483,646 to 2,147,483,647.
- **Decimal** means that the column must be able to represent values in the range -9,999,999,999.99 to +9,999,999,999.99 in increments of 0.01; the values can be either represented exactly or interpreted to be in this range;
- **Big Decimal** is of the Decimal datatype as defined above, with the additional property that it must be large enough to represent the aggregated values stored in temporary tables created within query variants;
- **Fixed text, size N** means that the column must be able to hold any string of characters of a fixed length of N.

**Comment:** If the string it holds is shorter than N characters, then trailing spaces must be stored in the database or the database must automatically pad with spaces upon retrieval such that a CHAR_LENGTH() function will return N.

- **Variable text, size N** means that the column must be able to hold any string of characters of a variable length with a maximum length of N. Columns defined as "variable text, size N" may optionally be implemented as "fixed text, size N";
- **Date** is a value whose external representation can be expressed as YYYY-MM-DD, where all characters are numeric. A date must be able to express any day within at least 14 consecutive years. There is no requirement specific to the internal representation of a date.

**Comment:** The implementation datatype chosen by the test sponsor for a particular datatype definition must be applied consistently to all the instances of that datatype definition in the schema, except for identifier columns, whose datatype may be selected to satisfy database scaling requirements.

1.3.2 The symbol SF is used in this document to represent the scale factor for the database (see Clause 4).

## 1.4 Table Layouts

### 1.4.1 Required Tables

The following list defines the required structure (list of columns) of each table. The annotations for primary keys and foreign references are for clarification only and do not specify any implementation requirement such as integrity constraints:

**PART Table Layout**

| Column Name | Datatype Requirements | Comment |
|---|---|---|
| P_PARTKEY | identifier | SF*200,000 are populated |
| P_NAME | variable text, size 55 | |
| P_MFGR | fixed text, size 25 | |

| | |
|---|---|
| P_BRAND | fixed text, size 10 |
| P_TYPE | variable text, size 25 |
| P_SIZE | integer |
| P_CONTAINER | fixed text, size 10 |
| P_RETAILPRICE | decimal |
| P_COMMENT | variable text, size 23 |

**Primary Key**: P_PARTKEY

**SUPPLIER Table Layout**

| Column Name | Datatype Requirements | Comment |
|---|---|---|
| S_SUPPKEY | identifier | SF*10,000 are populated |
| S_NAME | fixed text, size 25 | |
| S_ADDRESS | variable text, size 40 | |
| S_NATIONKEY | identifier | Foreign key reference to N_NATIONKEY |
| S_PHONE | fixed text, size 15 | |
| S_ACCTBAL | decimal | |
| S_COMMENT | variable text, size 101 | |

**Primary Key**: S_SUPPKEY

**PARTSUPP Table Layout**

| Column Name | Datatype Requirements | Comment |
|---|---|---|
| PS_PARTKEY | identifier | Foreign key reference to P_PARTKEY |
| PS_SUPPKEY | identifier | Foreign key reference to S_SUPPKEY |
| PS_AVAILQTY | integer | |
| PS_SUPPLYCOST | decimal | |
| PS_COMMENT | variable text, size 199 | |

**Compound Primary Key**: PS_PARTKEY, PS_SUPPKEY

**CUSTOMER Table Layout**

| Column Name | Datatype Requirements | Comment |
|---|---|---|
| C_CUSTKEY | identifier | SF*150,000 are populated |
| C_NAME | variable text, size 25 | |
| C_ADDRESS | variable text, size 40 | |
| C_NATIONKEY | identifier | Foreign key reference to N_NATIONKEY |
| C_PHONE | fixed text, size 15 | |
| C_ACCTBAL | decimal | |
| C_MKTSEGMENT | fixed text, size 10 | |
| C_COMMENT | variable text, size 117 | |

**Primary Key**: C_CUSTKEY

**ORDERS Table Layout**

| Column Name | Datatype Requirements | Comment |
|---|---|---|
| O_ORDERKEY | identifier | SF*1,500,000 are sparsely populated |
| O_CUSTKEY | identifier | Foreign key reference to C_CUSTKEY |
| O_ORDERSTATUS | fixed text, size 1 | |
| O_TOTALPRICE | decimal | |
| O_ORDERDATE | date | |
| O_ORDERPRIORITY | fixed text, size 15 | |
| O_CLERK | fixed text, size 15 | |
| O_SHIPPRIORITY | integer | |
| O_COMMENT | variable text, size 79 | |

**Primary Key**: O_ORDERKEY

**Comment**: Orders are not present for all customers. In fact, one-third of the customers do not have any order in the database. The orders are assigned at random to two-thirds of the customers (see Clause 4). The purpose of this is to exercise the capabilities of the DBMS to handle "dead data" when joining two or more tables.

**LINEITEM Table Layout**

| Column Name | Datatype Requirements | Comment |
|---|---|---|
| L_ORDERKEY | identifier | Foreign key reference to O_ORDERKEY |
| L_PARTKEY | identifier | Foreign key reference to P_PARTKEY, Compound Foreign Key Reference to (PS_PARTKEY, PS_SUPPKEY) with L_SUPPKEY |
| L_SUPPKEY | identifier | Foreign key reference to S_SUPPKEY, Compound Foreign key reference to (PS_PARTKEY, PS_SUPPKEY) with L_PARTKEY |
| L_LINENUMBER | integer | |
| L_QUANTITY | decimal | |
| L_EXTENDEDPRICE | decimal | |
| L_DISCOUNT | decimal | |
| L_TAX | decimal | |
| L_RETURNFLAG | fixed text, size 1 | |
| L_LINESTATUS | fixed text, size 1 | |
| L_SHIPDATE | date | |
| L_COMMITDATE | date | |
| L_RECEIPTDATE | date | |
| L_SHIPINSTRUCT | fixed text, size 25 | |
| L_SHIPMODE | fixed text, size 10 | |
| L_COMMENT | variable text size 44 | |

**Compound Primary Key**: L_ORDERKEY, L_LINENUMBER

**NATION Table Layout**

| Column Name | Datatype Requirements | Comment |
|---|---|---|
| N_NATIONKEY | identifier | 25 nations are populated |
| N_NAME | fixed text, size 25 | |
| N_REGIONKEY | identifier | Foreign key reference to R_REGIONKEY |
| N_COMMENT | variable text, size 152 | |

**Primary Key**: N_NATIONKEY

**REGION Table Layout**

| Column Name | Datatype Requirements | Comment |
|---|---|---|
| R_REGIONKEY | identifier | 5 regions are populated |
| R_NAME | fixed text, size 25 | |
| R_COMMENT | variable text, size 152 | |

**Primary Key**: R_REGIONKEY

### 1.4.2 Constraints

The use of constraints is optional. There is no specific requirement to define primary keys, foreign keys or check constraints. However, if constraints are used, they must satisfy the following requirements:

- They must be specified using SQL. There is no specific implementation requirement. For example, CREATE TABLE, ALTER TABLE, and CREATE TRIGGER are all valid statements;
- Constraints must be enforced either at the statement level or at the transaction level;
- All defined constraints must be enforced and validated before the load test is complete (see Clause 5.1.1.2);
- Any subset of the constraints listed below may be specified. No additional constraints may be used.

1.4.2.1 Nulls: The NOT NULL attribute may be used for any column.

1.4.2.2 Primary keys: The following primary keys may be defined as primary key (using the PRIMARY KEY clause or other equivalent syntax):

- P_PARTKEY;
- S_SUPPKEY;
- PS_PARTKEY, PS_SUPPKEY;
- C_CUSTKEY;
- O_ORDERKEY;
- L_ORDERKEY, L_LINENUMBER;
- N_NATIONKEY;
- R_REGIONKEY.

Constraining a column (or set of columns) to contain unique values can only be implemented for the primary key(s) listed above.

1.4.2.3 Foreign Keys: Any of the foreign keys listed in the comments of Clause 1.4.1 may be defined. There is no specific requirement for delete/update actions (e.g., RESTRICT, CASCADE, NO ACTION, etc.). If any foreign key relationship is defined by an implementation, then all foreign key relationships must be defined by the implementation.

1.4.2.4 Check Constraints: Check constraints may be defined to restrict the database contents. In order to support evolutionary change, the check constraints must not rely on knowledge of the enumerated domains of each column. The following list of expressions defines permissible check constraints:

1. Positive Keys

    1.P_PARTKEY >= 0

    2.S_SUPPKEY >= 0

    3.C_CUSTKEY >= 0

    4.PS_PARTKEY >= 0

    5.R_REGIONKEY >= 0

    6.N_NATIONKEY >= 0

2. Open-interval constraints

    1.P_SIZE >= 0

    2.P_RETAILPRICE >= 0

    3.PS_AVAILQTY >= 0

    4.PS_SUPPLYCOST >= 0

    5.O_TOTALPRICE >= 0

    6.L_QUANTITY >= 0

    7.L_EXTENDEDPRICE >= 0

    8.L_TAX >= 0

3. Closed-interval constraints

    1.L_DISCOUNT between 0.00 and 1.00

4. Multi-column constraints

    1.L_SHIPDATE <= L_RECEIPTDATE

**Comment:** The constraints rely solely on the diagram provided in Clause 1.2 and the description in Clause 1.4. They are not derived from explicit knowledge of the data population specified in Clause 4.2.

## 1.5 Implementation Rules

1.5.1 The database shall be implemented using a commercially available database management system (DBMS).

1.5.2 The physical clustering of records within the database is allowed as long as this clustering does not alter the logical independence of each table.

**Comment**: The intent of this clause is to permit flexibility in the physical design of a database while preserving a strict logical view of all the tables.

1.5.3 At the end of the Load Test, all tables must have exactly the number of rows defined for the scale factor, SF, and the database population, both specified in Clause 4.

1.5.4 Horizontal partitioning of base tables or auxiliary structures created by database directives (see Clause 1.5.7) is allowed. Groups of rows from a table or auxiliary structure may be assigned to different files, disks, or areas. If this assignment is a function of data in the table or auxiliary structure, the assignment must be based on the value of a partitioning field. A partitioning field must be one and only one of the following:

- A primary key as defined in Clause 1.4.2.2
- A foreign key as defined in Clause 1.4.1 may be defined. There is no specific requirement for delete/update actions (e.g., RESTRICT, CASCADE, NO ACTION, etc.). If any foreign key relationship is defined by an implementation, then all foreign key relationships must be defined by the implementation.
- A single date column

Some partitioning schemes require the use of directives that specify explicit values for the partitioning field. If such directives are used they must satisfy the following conditions:

- They may not rely on any knowledge of the data stored in the table except the minimum and maximum values of columns used for the partitioning field. The minimum and maximum values of columns are specified in Clause 4.2.3
- Within the limitations of integer division, they must define each partition to accept an equal portion of the range between the minimum and maximum values of the partitioning column(s).
- The directives must allow the insertion of values of the partitioning column(s) outside the range covered by the minimum and maximum values, as required by The database must allow for insertion of arbitrary data values that conform to the datatype and optional constraint definitions from Clause 1.3 and Clause 1.4..

Multiple-level partitioning of base tables or auxiliary structures is allowed only if each level of partitioning satisfies the conditions stated above and each level references only one partitioning field as defined above. If implemented, the details of such partitioning must be disclosed.

1.5.5 Physical placement of data on durable media is not auditable. SQL DDL that explicitly partitions data vertically is prohibited. The row must be logically presented as an atomic set of columns.

**Comment:** This implies that vertical partitioning which does not rely upon explicit partitioning directives is allowed. Explicit partitioning directives are those that assign groups of columns of one row to files, disks or areas different from those storing the other columns in that row.

1.5.6 Except as provided in Clause 1.5.7, logical replication of database objects (i.e., tables, rows, or columns) is not allowed. The physical implementation of auxiliary data structures to the tables may involve data replication of selected data from the tables provided that:

- All replicated data are managed by the DBMS, the operating system, or the hardware;
- All replications are transparent to all data manipulation operations;
- Data modifications are reflected in all logical copies of the replicated data by the time the updating transaction is committed;
- All copies of replicated data maintain full ACID properties (see Clause 3) at all times.

1.5.7 Auxiliary data structures that constitute logical replications of data from one or more columns of a base table (e.g., indexes, materialized views, summary tables, structures used to enforce relational integrity constraints) must conform to the provisions of Clause 1.5.6. The directives defining and creating these structures are subject to the fol-

lowing limitations:

- They may reference no more than one base table, and may not reference other auxiliary structures.
- They must satisfy exactly one of the following two conditions:
    - They may reference no more than one base table column that is chosen from:
        - A column that is a primary key on its own, or is a component of a compound primary key as defined in Required Tables
        - A column that is a foreign key on its own, or is a component of a compund foreign key as defined in Required Tables.
        - A column having a date datatype as defined in Clause 1.3.
    - They may reference more than one column of a base table only if those columns exactly comprise a compound primary or foreign key of that table, as defined in Clause 1.4.1.
- They may contain functions or expressions on explicitly permitted columns

No directives (e.g. DDL, session options, global configuration parameters) are permitted in TPC-HTPC-H scripts whose effect is to cause the materialization of columns (or functions on columns) in auxiliary data structures other than those columns explicitly permitted by the above limitations. Further, no directives are permitted whose effect is to cause the materialization of columns in auxiliary data structures derived from more than one table.

**Comment:** Database implementations of auxiliary structures generated as a result of compliant directives usually contain embedded pointers or references to corresponding base table rows. Database implementations that transparently employ either "row IDs" or embedded base table primary key values for this purpose are equally acceptable. In particular, the generation of transparently embedded primary key values required by auxiliary structures is a permitted materialization of the primary key column(s). Primary and foreign key columns are defined in Required Tables.

1.5.8 Table names should match those provided in Clause 1.4. In cases where a table name conflicts with a reserved word in a given implementation, delimited identifiers or an alternate meaningful name may be chosen.

1.5.9 For each table, the set of columns must include all those defined in Clause 1.4. No column can be added to any of the tables. However, the order of the columns is not constrained.

1.5.10 Column names must match those provided in Clause 1.4.

1.5.11 Each column, as described in Clause 1.4, must be logically discrete and independently accessible by the data manager. For example, C_ADDRESS and C_PHONE cannot be implemented as two sub-parts of a single discrete column C_DATA.

1.5.12 Each column, as described in Clause 1.4, must be accessible by the data manager as a single column. For example, P_TYPE cannot be implemented as two discrete columns P_TYPE1 and P_TYPE2.

1.5.13 The database must allow for insertion of arbitrary data values that conform to the datatype and optional constraint definitions from Clause 1.3 and Clause 1.4.

**Comment 1**: Although the refresh functions (see Clause 2.26) do not insert arbitrary values and do not modify all tables, all tables must be modifiable throughout the performance test.

**Comment 2**: The intent of this Clause is to prevent the database schema definition from taking undue advantage of the limited data population of the database (see also Clause 0.2 and Clause 5.2.7).

## 1.6 Data Access Transparency Requirements

1.6.1 Data Access Transparency is the property of the system that removes from the query text any knowledge of the location and access mechanisms of partitioned data. No finite series of tests can prove that the system supports complete data access transparency. The requirements below describe the minimum capabilities needed to establish that the system provides transparent data access. An implementation that uses horizontal partitioning must meet the requirements for transparent data access described in Clause 1.6.2 and Clause 1.6.3.

**Comment**: The intent of this Clause is to require that access to physically and/or logically partitioned data be provided directly and transparently by services implemented by commercially available layers such as the interactive SQL interface, the database management system (DBMS), the operating system (OS), the hardware, or any combination of these.

1.6.2 Each of the tables described in Clause 1.4 must be identifiable by names that have no relationship to the partitioning of tables. All data manipulation operations in the executable query text (see Clause 2.1.1.2) must use only these names.

1.6.3 Using the names which satisfy Clause 1.6.2, any arbitrary non-TPC-H query must be able to reference any set of rows or columns:

- Identifiable by any arbitrary condition supported by the underlying DBMS;
- Using the names described in Clause 1.6.2 and using the same data manipulation semantics and syntax for all tables.

For example, the semantics and syntax used to query an arbitrary set of rows in any one table must also be usable when querying another arbitrary set of rows in any other table.

**Comment**: The intent of this clause is that each TPC-H query uses general purpose mechanisms to access data in the database.

# 2: QUERIES AND REFRESH FUNCTIONS

This Clause describes the twenty-two decision support queries and the two database refresh functions that must be executed as part of the TPC-H benchmark.

## 2.1 General Requirements and Definitions for Queries

### 2.1.1 Query Overview

2.1.1.1 Each query is defined by the following components:

- The **business question**, which illustrates the business context in which the query could be used;
- The **functional query definition,** which defines, using the SQL-92 language, the function to be performed by the query;
- The **substitution parameters,** which describe how to generate the values needed to complete the query syntax;
- The **query validation,** which describes how to validate the query against the qualification database.

2.1.1.2 For each query, the test sponsor must create an implementation of the functional query definition, referred to as the **executable query text**.

### 2.1.2 Functional Query Definitions

2.1.2.1 The functional query definitions are written in the **SQL-92** language (ISO/IEC 9075:1992), annotated where necessary to specify the number of rows to be returned. They define the function that each executable query text must perform against the test database (see Clause 4.1.1).

2.1.2.2 If an executable query text, with the exception of its substitution parameters, is not identical to the specified functional query definition it must satisfy the compliance requirements of Clause 2.2.

2.1.2.3 When a functional query definition includes the creation of a new entity (e.g., cursor, view, or table) some mechanism must be used to ensure that newly created entities do not interfere with other execution streams and are not shared between multiple execution streams (see Clause 5.1.2.3).

Functional query definitions in this document (as well as QGEN, see Clause 2.1.4) achieve this separation by appending a **text-token** to the new entity name. This text-token is expressed in upper case letters and enclosed in square brackets (i.e., [STREAM_ID]). This text-token, whenever found in the functional query definition, must be replaced by a unique stream identification number (starting with 0) to complete the executable query text.

**Comment**: Once an identification number has been generated and assigned to a given query stream, the same identification number must be used for that query stream for the duration of the test.

2.1.2.4 When a functional query definition includes the creation of a table, the datatype specification of the columns uses the <datatype> notation. The definition of <datatype> is obtained from Clause 1.3.1.

2.1.2.5 Any entity created within the scope of an executable query text must also be deleted within the scope of that same executable query text.

2.1.2.6 A logical tablespace is a named collection of physical storage devices referenced as a single, logically contiguous, non-divisible entity.

2.1.2.7 If CREATE TABLE statements are used during the execution of the queries, these CREATE TABLE statements may be extended only with a tablespace reference (e.g., IN <tablespacename>). A single tablespace must be used for all these tables.

**Comment:** The allowance for tablespace syntax applies only to variants containing CREATE TABLE statements.

2.1.2.8 All tables created during the execution of a query must meet the ACID properties defined in Clause 3.

2.1.2.9 Queries 2, 3, 10, 18 and 21 require that a given number of rows are to be returned (e.g., "Return the first 10 selected rows"). If N is the number of rows to be returned, the query must return exactly the first N rows unless fewer than N rows qualify, in which case all rows must be returned. There are three permissible ways of satisfying this requirement. A test sponsor must select any one of them and use it consistently for all the queries that require that a specified number of rows be returned.

1. Vendor-specific control statements supported by a test sponsor's interactive SQL interface may be used (e.g., SET ROWCOUNT n) to limit the number of rows returned.
2. Control statements recognized by the implementation specific layer (see Clause 6.2.4) and used to control a loop which fetches the rows may be used to limit the number of rows returned (e.g., while rowcount <= n).
3. Vendor-specific SQL syntax may be added to the SELECT statement to limit the number of rows returned (e.g., SELECT FIRST n). This syntax is not classified as a minor query modification since it completes the functional requirements of the functional query definition and there is no standardized syntax defined. In all other respects, the query must satisfy the requirements of Clause 2.2. The syntax must deal solely with the answer set, and must not make any additional explicit reference, for example to tables, indices, or access paths.

### 2.1.3 Substitution Parameters and Output Data

2.1.3.1 Each query has one or more **substitution parameters**. When generating executable query text a value must be supplied for each substitution parameter of that query. These values must be used to complete the executable query text. These substitution parameters are expressed as names in uppercase and enclosed in square brackets. For example, in the Pricing Summary Report Query (see Clause 2.4) the substitution parameter [DELTA], whenever found in the functional query definition, must be replaced by the value generated for DELTA to complete the executable query text.

**Comment 1**: When dates are part of the substitution parameters, they must be expressed in a format that includes the year, month and day in integer form, in that order (e.g., YYYY-MM-DD). The delimiter between the year, month and day is not specified. Other date representations, for example the number of days since 1970-01-01, are specifically not allowed.

**Comment 2**: When a substitution parameter appears more than once in a query, a single value is generated for that substitution parameter and each of its occurrences in the query must be replaced by that same value.

**Comment 3**: Generating executable query text may also involve additional text substitution (see Clause 2.1.2.3).

2.1.3.2 The term **randomly selected** when used in the definitions of substitution parameters means selected at random from a uniform distribution over the range or list of values specified.

2.1.3.3 Seeds to the random number generator used to generate substitution parameters must be selected using the following method:

An initial seed (seed0) is first selected as the time stamp of the end of the database load time expressed in the format mmddhhmmss where mm is the month, dd the day, hh the hour, mm the minutes and ss the seconds. This seed is used to seed the Power test of Run 1. Further seeds (for the Throughput test) are chosen as seed0 + 1, seed0 +

2,...,seed0 + n where s is the number of throughput streams selected by the vendor. This process leads to s + 1 seeds required for Run 1 of a benchmark with s streams. The seeds for Run 2 can be the same as those for Run 2 (see 5.3.2). However, should the test sponsor decide to use different seeds for Run 2 from those used for Run 1, the sponsor must use a selection process similar to that of Run 1. The seeds must again be of the form seed0, seed0 + 1, seed0 + 2,...., seed0 + s, where and seed0 is be the time stamp of the end of Run 1, expressed in the format defined above.

**Comment 1**: The intent of this Clause is to prevent performance advantage that could result from multiple streams beginning work with identical seeds or using seeds known in advance while providing a well-defined and unified method for seed selection.

**Comment 2**: QGEN is a utility provided by the TPC (see Clause 2.1.4) to generate executable query text. If a sponsor-created tool is used instead of QGEN, the behavior of its seeds must satisfy this Clause and its code must be disclosed. After execution, the query returns one or more rows. The rows returned are either rows from the database or rows built from data in the database and are called the **output data**.

2.1.3.4 Output data for each query should be expressed in a format easily readable by a non-sophisticated computer user. In particular, in order to be comparable with known output data for the purpose of query validation (see Clause 2.3), the format of the output data for each query must adhere to the following guidelines:

a) Columns appear in the order specified by the SELECT list of either the functional query definition or an approved variant. Column headings are optional.

b) Non-integer expressions including prices are expressed in decimal notation with at least two digits behind the decimal point.

c) Integer quantities contain no leading zeros.

d) Dates are expressed in a format that includes the year, month and day in integer form, in that order (e.g., YYYY-MM-DD). The delimiter between the year, month and day is not specified. Other date representations, for example the number of days since 1970-01-01, are specifically not allowed.

e) Strings are case-sensitive and must be displayed as such. Leading or trailing blanks are acceptable.

f) The amount of white space between columns is not specified.

2.1.3.5 The **precision** of all values contained in the query validation output data must adhere to the following rules:

a) For singleton column values and results from COUNT aggregates, the values must exactly match the query validation output data.

b) For ratios, results must be within 1% of the query validation output data when reported to the nearest 1/100th, rounded up.

c) For results from SUM aggregates, the resulting values must be within $100 of the query validation output data.

d) For results from AVG aggregates, the resulting values must be within 1% of the query validation output data when reported to the nearest 1/100th, rounded up.

### 2.1.4 The QGEN Program

2.1.4.1 Executable query text must be generated according to the requirements of Clause 2.1.2 and Clause 2.1.3. The **QGEN** source code provided in Appendix D is a sample implementation of an executable query text generator. It has been written in ANSI 'C' and has been ported to a large number of platforms. If QGEN is used, its version must match the version of the benchmark specification.

**Comment 1**: Use of QGEN is strongly recommended. Exact query answer set compliance is required. This may not be possible unless substitution parameters and text tokens are generated and integrated within the executable query text identically to QGEN's output.

**Comment 2**: The numbering used in this Clause for the definition of substitution parameters corresponds to the numbering used by QGEN to generate values for these substitution parameters.

## 2.2 Query Compliance

2.2.1 The queries must be expressed in a commercially available implementation of the SQL language. Since the latest ISO SQL standard (currently ISO/IEC 9075:1992) has not yet been fully implemented by most vendors, and since the ISO SQL language is continually evolving, the TPC-H benchmark specification includes a number of permissible deviations from the formal functional query definitions found in Clause 2. An on-going process is also defined to approve additional deviations that meet specific criteria.

2.2.2 There are two types of permissible deviations from the functional query definitions, as follows:

a) Minor query modifications;

b) Approved query variants.

### 2.2.3 Minor Query Modifications

2.2.3.1 It is recognized that implementations require specific adjustments for their operating environment and the syntactic variations of its dialect of the SQL language. Therefore, minor query modifications are allowed. Minor query modifications are those that fall within the bounds of what is described in Clause 2.2.3.3. They do not require approval. Modifications that do not fall within the bounds of what is described in Clause 2.2.3.3 are not minor and are not compliant unless they are an integral part of an approved query variant (see Clause 2.2.4).

**Comment 1**: The intent of this Clause is to allow the use of any number of minor query modifications. These query modifications are labeled minor based on the assumption that they do not significantly impact the performance of the queries.

**Comment 2:** The only exception is for the queries that require a given number of rows to be returned. The requirements governing this exception are given in Clause 2.1.2.9.

2.2.3.2 Minor query modifications can be used to produce executable query text by modifying either a functional query definition or an approved variant of that definition.

2.2.3.3 The following query modifications are minor:

a) Table names - The table and view names found in the CREATE TABLE, CREATE VIEW, DROP VIEW and in the FROM clause of each query may be modified to reflect the customary naming conventions of the system under test.

b) Select-list expression aliases - For queries that include the definition of an alias for a SELECT-list item (e.g., AS CLAUSE), vendor-specific syntax may be used instead of the specified SQL-92 syntax. Replacement syntax must have equivalent semantic behavior. Examples of acceptable implementations include "TITLE <string>", or "WITH HEADING <string>". Use of a select-list expression alias is optional.

c) Date expressions - For queries that include an expression involving manipulation of dates (e.g., adding/subtracting days/months/years, or extracting years from dates), vendor-specific syntax may be used instead of the specified SQL-92 syntax. Replacement syntax must have equivalent semantic behavior. Examples of acceptable implementations include "YEAR(<column>)" to extract the year from a date column or "DATE(<date>) + 3 MONTHS" to add 3 months to a date.

d) GROUP BY and ORDER BY - For queries that utilize a view, nested table-expression, or select-list alias solely for the purposes of grouping or ordering on an expression, vendors may replace the view, nested table-expression or select-list alias with a vendor-specific SQL extension to the GROUP BY or ORDER BY clause. Examples of acceptable implementations include "GROUP BY <ordinal>", "GROUP BY <expression>", "ORDER BY <ordinal>", and "ORDER BY <expression>".

e) Command delimiters - Additional syntax may be inserted at the end of the executable query text for the purpose of signaling the end of the query and requesting its execution. Examples of such command delimiters are a semicolon or the word "GO".

f) Output formatting functions - Scalar functions whose sole purpose is to affect output formatting or intermediate arithmetic result precision (such as CASTs) may be applied to items in the outermost SELECT list of the query.

g) Transaction control statements - A CREATE/DROP TABLE or CREATE/DROP VIEW statement may be followed by a COMMIT WORK statement or an equivalent vendor-specific transaction control statement.

h) Correlation names – Table-name aliases may be added to the executable query text. The keyword "AS" before the table-name alias may be omitted.

i) Explicit ASC - ASC may be explicitly appended to columns in the ORDER BY.

j) CREATE TABLE statements may be augmented with a tablespace reference conforming to the requirements of Clause 2.1.2.6.

k) In cases where identifier names conflict with SQL-92 reserved words in a given implementation, delimited identifiers may be used.

l) Relational operators - Relational operators used in queries such as "<", ">", "<>", "<=", and "=", may be replaced by equivalent vendor-specific operators, for example ".LT.", ".GT.", "!=" or "^=", ".LE.", and "==", respectively.

m) Nested table-expression aliasing - For queries involving nested table-expressions, the nested keyword "AS" before the table alias may be omitted.

n) If an implementation is using variants involving views and the implementation only supports “DROP RESTRICT” semantics (i.e., all dependent objects must be dropped first), then additional DROP statements for the dependent views may be added.

o) At large scale factors, the aggregates may exceed the range of the values supported by an integer. The aggregate functions AVG and COUNT may be replaced with equivalent vendor-specific functions to handle the expanded range of values (e.g., AVG_BIG and COUNT_BIG).

p) Substring Scalar Functions – For queries which use the SUBSTRING() scalar function, vendor-specific syntax may be used instead of the specified SQL 92 syntax. Replacement syntax must have equivalent semantic behavior. For example, “SUBSTRING(C_PHONE, 1, 2)”.

q) Outer Join – For outer join queries, vendor specific syntax may be used instead of the specified SQL 92 syntax. Replacement syntax must have equivalent semantic behavior. For example, the join expression “CUSTOMER LEFT OUTER JOIN ORDERS ON C_CUSTKEY = O_CUSTKEY” may be replaced by adding CUSTOMER and ORDERS to the from clause and adding a specially-marked join predicate (e.g., C_CUSTKEY *= O_CUSTKEY).

2.2.3.4 The application of minor query modifications to functional query definitions or approved variants must be consistent over the query set. For example, if a particular vendor-specific date expression or table name syntax is used in one query, it must be used in all other queries involving date expressions or table names.

2.2.3.5 The use of minor modifications to obtain executable query text must be disclosed and justified (see Clause 8.3.3.4).

### 2.2.4 Approved Query Variants

2.2.4.1 Approval of any new query variant is required prior to using such variant to produce compliant TPC-H results. The approval process is based on criteria defined in Clause 2.2.4.3.

2.2.4.2 Query variants that have already been approved are listed in Appendix B of this specification.

**Comment**: Since Appendix B is updated each time a new variant is approved, test sponsors should obtain the latest version of this appendix prior to implementing the benchmark.

2.2.4.3 The executable query text for each query in a compliant implementation must be taken from either the functional query definition (see Clause 2) or an approved query variant (see Appendix B). Except as specifically allowed in Clause 2.2.3.3, executable query text must be used in full exactly as written in the TPC-H specification. New query variants will be considered for approval if they meet one of the following criteria:

a) The vendor cannot successfully run the executable query text against the qualification database using the functional query definition or an approved variant even after applying appropriate minor query modifications as per Clause 2.2.3.

b) The variant contains new or enhanced SQL syntax, relevant to the benchmark, which is defined in an Approved Committee Draft of a new ISO SQL standard.

c) The variant contains syntax that brings the proposed variant closer to adherence to an ISO SQL standard.

d) The variant contains minor syntax differences that have a straightforward mapping to ISO SQL syntax used in the functional query definition and offers functionality substantially similar to the ISO SQL standard.

2.2.4.4 To be approved, a proposed variant should have the following properties. Not all of the following properties are specifically required. Rather, the cumulative weight of each property satisfied by the proposed variant will be the determining factor in approving it.

a) Variant is syntactical only, seeking functional compatibility and not performance gain.

b) Variant is minimal and restricted to correcting a missing functionality.

c) Variant is based on knowledge of the business question rather than on knowledge of the system under test (SUT) or knowledge of specific data values in the test database.

d) Variant has broad applicability among different vendors.

e) Variant is non procedural.

f) Variant is an SQL-92 standard [ISO/IEC 9075:1992] implementation of the functional query definition.

g) Variant is sponsored by a vendor who can implement it and who intends on using it in an upcoming implementation of the benchmark.

2.2.4.5 Query variants that are submitted for approval will be recorded, along with a rationale describing why they were or were not approved.

2.2.4.6 Query variants listed in Appendix B are defined using the conventions defined for functional query definitions (see Clause 2.1.2.3 through Clause 2.1.2.6).

### 2.2.5 Coding Style

Implementers may code the executable query text in any desired coding style, including:

a) additional line breaks, tabs or white space

b) choice of upper or lower case text

The coding style used must have no impact on the performance of the system under test, and must be consistently applied across the entire query set. Any coding style that differs from the functional query definitions in Clause 2 must be disclosed.

**Comment:** This does not preclude the auditor from verifying that the coding style does not affect performance.

## 2.3 Query Validation

2.3.1 To validate the compliance of the executable query text, the following validation test must be executed by the test sponsor and the results reported in the full disclosure report:

1. A qualification database must be built in a manner substantially the same as the test database (see Clause 4.1.2).
2. The query validation test must be run using a qualification database that has not been modified by any update activity (e.g., RF1, RF2, or ACID Transaction executions).
3. The query text used (see Clause 2.1.3) must be the same as that used in the performance test. The default substitution parameters provided for each query must be used. The refresh functions, RF1 and RF2, are not executed.
4. The same driver and implementation specific layer used to execute the queries against the test database must be used for the validation of the qualification database.
5. The resulting output must match the output data specified for the query validation (see Appendix C). A subset of this output can be found as part of the definition of each query.
6. Any difference between the output obtained and the query validation output must satisfy the requirements of Clause 2.1.3.5.

Any query whose output differs from the query validation output to a greater degree than allowed by Clause 2.1.3.5 when run against the qualification database as specified above is not compliant.

**Comment**: The validation test, above, provides a minimum level of assurance of compliance. The auditor may request additional assurance that the query texts execute in accordance with the benchmark requirements.

2.3.2 No aspect of the System Under Test (e.g., system parameters and conditional software features such as those listed in Clause 5.2.7, hardware configuration, software releases, etc.), may differ between this demonstration of compliance and the performance test.

**Comment**: While the intent of this validation test is that it be executed without any change to the hardware configuration, building the qualification database on additional disks (i.e., disks not included in the priced system) is allowed as long as this change has no impact on the results of the demonstration of compliance.

## 2.4 Pricing Summary Report Query (Q1)

This query reports the amount of business that was billed, shipped, and returned.

### 2.4.1 Business Question

The Pricing Summary Report Query provides a summary pricing report for all lineitems shipped as of a given date. The date is within 60 - 120 days of the greatest ship date contained in the database. The query lists totals for extended price, discounted extended price, discounted extended price plus tax, average quantity, average extended price, and average discount. These aggregates are grouped by RETURNFLAG and LINESTATUS, and listed in ascending order of RETURNFLAG and LINESTATUS. A count of the number of lineitems in each group is included.

### 2.4.2 Functional Query Definition

```
select
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
        sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
from
        lineitem
where
        l_shipdate <= date '1998-12-01' - interval '[DELTA]' day (3)
group by
        l_returnflag,
        l_linestatus
order by
        l_returnflag,
        l_linestatus;
```

### 2.4.3 Substitution Parameters

Values for the following substitution parameter must be generated and used to build the executable query text:

7. DELTA is randomly selected within [60. 120].

**Comment**: 1998-12-01 is the highest possible ship date as defined in the database population. (This is ENDDATE - 30). The query will include all lineitems shipped before this date minus DELTA days. The intent is to choose DELTA so that between 95% and 97% of the rows in the table are scanned.

### 2.4.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. DELTA = 90.

Query validation output data:

| L_RETURNFLAG | L_LINESTATUS | SUM_QTY | SUM_BASE_PRICE | SUM_DISC_PRICE |
|---|---|---|---|---|
| A | F | 37734107.00 | 56586554400.73 | 53758257134.87 |
| N | F | 991417.00 | 1487504710.38 | 1413082168.05 |
| N | O | 74476040.00 | 111701729697.74 | 106118230307.61 |
| R | F | 37719753.00 | 56568041380.90 | 53741292684.60 |

| SUM_CHARGE | AVG_QTY | AVG_PRICE | AVG_DISC | COUNT_ORDER |
|---|---|---|---|---|
| 55909065222.83 | 25.52 | 38273.13 | .05 | 1478493 |
| 1469649223.19 | 25.52 | 38284.47 | .05 | 38854 |
| 110367043872.50 | 25.50 | 38249.12 | .05 | 2920374 |
| 55889619119.83 | 25.51 | 38250.86 | .05 | 1478870 |

## 2.5 Minimum Cost Supplier Query (Q2)

This query finds which supplier should be selected to place an order for a given part in a given region.

### 2.5.1 Business Question

The Minimum Cost Supplier Query finds, in a given region, for each part of a certain type and size, the supplier who can supply it at minimum cost. If several suppliers in that region offer the desired part type and size at the same (minimum) cost, the query lists the parts from suppliers with the 100 highest account balances. For each supplier, the query lists the supplier's account balance, name and nation; the part's number and manufacturer; the supplier's address, phone number and comment information.

### 2.5.2 Functional Query Definition

Return the first 100 selected rows

```
select
        s_acctbal,
        s_name,
        n_name,
        p_partkey,
        p_mfgr,
        s_address,
        s_phone,
        s_comment
from
        part,
        supplier,
        partsupp,
        nation,
        region
where
        p_partkey = ps_partkey
        and s_suppkey = ps_suppkey
        and p_size = [SIZE]
        and p_type like '%[TYPE]'
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = '[REGION]'
        and ps_supplycost = (
                select
                        min(ps_supplycost)
                from
                        partsupp, supplier,
                        nation, region
                where
                        p_partkey = ps_partkey
                        and s_suppkey = ps_suppkey
                        and s_nationkey = n_nationkey
                        and n_regionkey = r_regionkey
                        and r_name = '[REGION]'
```

```
                    )
order by
        s_acctbal desc,
        n_name,
        s_name,
        p_partkey;
```

### 2.5.3 Substitution Parameters

Values for the following substitution parameter must be generated and used to build the executable query text:

1. SIZE is randomly selected within [1. 50];
2. TYPE is randomly selected within the list Syllable 3 defined for Types in Clause 4.2.2.12;
3. REGION is randomly selected within the list of values defined for R_NAME in Clause 4.2.3.

### 2.5.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. SIZE = 15;
2. TYPE = BRASS;
3. REGION = EUROPE.

Query validation output data:

| S_ACCTBAL | S_NAME | N_NAME | P_PARTKEY | P_MFGR |
|---|---|---|---|---|
| 9938.53 | Supplier#000005359 | UNITED KINGDOM | 185358 | Manufacturer#4 |
| 9937.84 | Supplier#000005969 | ROMANIA | 108438 | Manufacturer#1 |
| 9936.22 | Supplier#000005250 | UNITED KINGDOM | 249 | Manufacturer#4 |
| 9923.77 | Supplier#000002324 | GERMANY | 29821 | Manufacturer#4 |
| 9871.22 | Supplier#000006373 | GERMANY | 43868 | Manufacturer#5 |

[90 more rows]

| | | | | |
|---|---|---|---|---|
| 7887.08 | Supplier#000009792 | GERMANY | 164759 | Manufacturer#3 |
| 7871.50 | Supplier#000007206 | RUSSIA | 104695 | Manufacturer#1 |
| 7852.45 | Supplier#000005864 | RUSSIA | 8363 | Manufacturer#4 |

| | | | | |
|---|---|---|---|---|
| 7850.66 | Supplier#000001518 | UNITED KINGDOM | 86501 | Manufacturer#1 |
| 7843.52 | Supplier#000006683 | FRANCE | 11680 | Manufacturer#4 |

| S_ADDRESS | S_PHONE | S_COMMENT |
|---|---|---|
| QKu-HYh,vZGiwu2FWEJoLDx04 | 33-429-790-6131 | blithely silent pinto beans are furiously. slyly final deposits acros |
| ANDEN-SOSmk,miq23Xfb5RWt6dvUcvt6Qa | 29-520-692-3537 | carefully slow deposits use furiously. slyly ironic platelets above the ironic |
| B3rqp0xbSEim4Mpy 2RH J | 33-320-228-2957 | blithely special packages are. stealthily express deposits across the closely final instructi |
| y3OD9UywSTOk | 17-779-299-1839 | quickly express packages breach quiet pinto beans. requ |
| J8fcXWsTqM | 17-813-485-8637 | never silent deposits integrate furiously blit |

[90 More Rows]

| | | |
|---|---|---|
| Y28ITVeYriT3kIGdV2K8fSZV2UqT5H1Otz | 17-988-938-4296 | pending, ironic packages sleep among the carefully ironic accounts. quickly final accounts |
| 3w fNCnrVmvJjE95sgWZzvW | 32-432-452-7731 | furiously dogged pinto beans cajole. bold, express notornis until the slyly pending |
| WCNfBPZeSXh3h,c | 32-454-883-3821 | blithely regular deposits |
| ONda3YJiHKJOC | 33-730-383-3892 | furiously final accounts wake carefully idle requests. even dolphins wake acc |
| 2Z0JGkiv01Y00oCFwUGfviIbhzCdy | 16-464-517-8943 | carefully bold accounts doub |

## 2.6 Shipping Priority Query (Q3)

This query retrieves the 10 unshipped orders with the highest value.

### 2.6.1 Business Question

The Shipping Priority Query retrieves the shipping priority and potential revenue, defined as the sum of l_extendedprice * (1-l_discount), of the orders having the largest revenue among those that had not been shipped as of a given date. Orders are listed in decreasing order of revenue. If more than 10 unshipped orders exist, only the 10 orders with the largest revenue are listed.

### 2.6.2 Functional Query Definition

Return the first 10 selected rows

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < date '[DATE]'
        and l_shipdate > date '[DATE]'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate;
```

### 2.6.3 Substitution Parameters

Values for the following substitution parameters must be generated and used to build the executable query text:

1. SEGMENT is randomly selected within the list of values defined for Segments in Clause 4.2.2.12;
2. DATE is a randomly selected day within [1995-03-01 .. 1995-03-31].

### 2.6.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. SEGMENT = BUILDING;
2. DATE = 1995-03-15.

Query validation output data:

| L_ORDERKEY | REVENUE | O_ORDERDATE | O_SHIPPRIORITY |
|---|---|---|---|
| 2456423 | 406181.01 | 1995-03-05 | 0 |
| 3459808 | 405838.70 | 1995-03-04 | 0 |
| 492164 | 390324.06 | 1995-02-19 | 0 |
| 1188320 | 384537.94 | 1995-03-09 | 0 |
| 2435712 | 378673.06 | 1995-02-26 | 0 |
| 4878020 | 378376.80 | 1995-03-12 | 0 |
| 5521732 | 375153.92 | 1995-03-13 | 0 |
| 2628192 | 373133.31 | 1995-02-22 | 0 |
| 993600 | 371407.46 | 1995-03-05 | 0 |
| 2300070 | 367371.15 | 1995-03-13 | 0 |

## 2.7 Order Priority Checking Query (Q4)

This query determines how well the order priority system is working and gives an assessment of customer satisfaction.

### 2.7.1 Business Question

The Order Priority Checking Query counts the number of orders ordered in a given quarter of a given year in which at least one lineitem was received by the customer later than its committed date. The query lists the count of such orders for each order priority sorted in ascending priority order.

### 2.7.2 Functional Query Definition

```
select
        o_orderpriority,
        count(*) as order_count
from orders
where
        o_orderdate >= date '[DATE]'
        and o_orderdate < date '[DATE]' + interval '3' month
        and exists (
                select
                        *
                from
                        lineitem
                where
                        l_orderkey = o_orderkey
                        and l_commitdate < l_receiptdate
        )
group by
        o_orderpriority
order by
        o_orderpriority;
```

### 2.7.3 Substitution Parameters

Values for the following substitution parameter must be generated and used to build the executable query text:

1. DATE is the first day of a randomly selected month between the first month of 1993 and the 10th month of 1997.

### 2.7.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. DATE = 1993-07-01.

Query validation output data:

| O_ORDERPRIORITY | ORDER_COUNT |
|---|---|
| 1-URGENT | 10594 |
| 2-HIGH | 10476 |
| 3-MEDIUM | 10410 |
| 4-NOT SPECIFIED | 10556 |
| 5-LOW | 10487 |

## 2.8 Local Supplier Volume Query (Q5)

This query lists the revenue volume done through local suppliers.

### 2.8.1 Business Question

The Local Supplier Volume Query lists for each nation in a region the revenue volume that resulted from lineitem transactions in which the customer ordering parts and the supplier filling them were both within that nation. The query is run in order to determine whether to institute local distribution centers in a given region. The query considers only parts ordered in a given year. The query displays the nations and revenue volume in descending order by revenue. Revenue volume for all qualifying lineitems in a particular nation is defined as sum(l_extendedprice * (1 - l_discount)).

### 2.8.2 Functional Query Definition

```
select
        n_name,
        sum(l_extendedprice * (1 - l_discount)) as revenue
from
        customer,
        orders,
        lineitem,
        supplier,
        nation,
        region
where
        c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and l_suppkey = s_suppkey
        and c_nationkey = s_nationkey
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = '[REGION]'
        and o_orderdate >= date '[DATE]'
        and o_orderdate < date '[DATE]' + interval '1' year
group by
        n_name
order by
        revenue desc;
```

### 2.8.3 Substitution Parameters

Values for the following substitution parameters must be generated and used to build the executable query text:

1. REGION is randomly selected within the list of values defined for R_NAME in Clause 4.2.3;
2. DATE is the first of January of a randomly selected year within [1993 .. 1997].

### 2.8.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. REGION = ASIA;
2. DATE = 1994-01-01.

Query validation output data:

| N_NAME | REVENUE |
|---|---|
| INDONESIA | 55502041.17 |
| VIETNAM | 55295087.00 |
| CHINA | 53724494.26 |
| INDIA | 52035512.00 |
| JAPAN | 45410175.70 |

## 2.9 Forecasting Revenue Change Query (Q6)

This query quantifies the amount of revenue increase that would have resulted from eliminating certain company-wide discounts in a given percentage range in a given year. Asking this type of "what if" query can be used to look for ways to increase revenues.

### 2.9.1 Business Question

The Forecasting Revenue Change Query considers all the lineitems shipped in a given year with discounts between DISCOUNT-0.01 and DISCOUNT+0.01. The query lists the amount by which the total revenue would have increased if these discounts had been eliminated for lineitems with l_quantity less than quantity. Note that the potential revenue increase is equal to the sum of [l_extendedprice * l_discount] for all lineitems with discounts and quantities in the qualifying range.

### 2.9.2 Functional Query Definition

```
select
        sum(l_extendedprice*l_discount) as revenue
from
        lineitem
where
        l_shipdate >= date '[DATE]'
        and l_shipdate < date '[DATE]' + interval '1' year
        and l_discount between [DISCOUNT] - 0.01 and [DISCOUNT] + 0.01
        and l_quantity < [QUANTITY];
```

### 2.9.3 Substitution Parameters

Values for the following substitution parameters must be generated and used to build the executable query text:

1. DATE is the first of January of a randomly selected year within [1993 .. 1997];
2. DISCOUNT is randomly selected within [0.02 .. 0.09];
3. QUANTITY is randomly selected within [24 .. 25].

### 2.9.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. DATE = 1994-01-01;
2. DISCOUNT = 0.06;
3. QUANTITY = 24.

Query validation output data:

| REVENUE |
|---|

123141078.23

## 2.10 Volume Shipping Query (Q7)

This query determines the value of goods shipped between certain nations to help in the re-negotiation of shipping contracts.

### 2.10.1 Business Question

The Volume Shipping Query finds, for two given nations, the gross discounted revenues derived from lineitems in which parts were shipped from a supplier in either nation to a customer in the other nation during 1995 and 1996. The query lists the supplier nation, the customer nation, the year, and the revenue from shipments that took place in that year. The query orders the answer by Supplier nation, Customer nation, and year (all ascending).

### 2.10.2 Functional Query Definition

```
select
        supp_nation,
        cust_nation,
        l_year, sum(volume) as revenue
from (
        select
                n1.n_name as supp_nation,
                n2.n_name as cust_nation,
                extract(year from l_shipdate) as l_year,
                l_extendedprice * (1 - l_discount) as volume
        from
                supplier,
                lineitem,
                orders,
                customer,
                nation n1,
                nation n2
        where
                s_suppkey = l_suppkey
                and o_orderkey = l_orderkey
                and c_custkey = o_custkey
                and s_nationkey = n1.n_nationkey
                and c_nationkey = n2.n_nationkey
                and (
                        (n1.n_name = '[NATION1]' and n2.n_name = '[NATION2]')
                        or (n1.n_name = '[NATION2]' and n2.n_name = '[NATION1]')
                )
                and l_shipdate between date '1995-01-01' and date '1996-12-31'
```

```
        ) as shipping
group by
        supp_nation,
        cust_nation,
        l_year
order by
        supp_nation,
        cust_nation,
        l_year;
```

### 2.10.3 Substitution Parameters

Values for the following substitution parameters must be generated and used to build the executable query text:

1. NATION1 is randomly selected within the list of values defined for N_NAME in Clause 4.2.3;
2. NATION2 is randomly selected within the list of values defined for N_NAME in Clause 4.2.3 and must be different from the value selected for NATION1 in item 1 above.

### 2.10.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. NATION1 = FRANCE;
2. NATION2 = GERMANY.

Query validation output data:

| SUPP_NATION | CUST_NATION | YEAR | REVENUE |
|---|---|---|---|
| FRANCE | GERMANY | 1995 | 54639732.73 |
| FRANCE | GERMANY | 1996 | 54633083.31 |
| GERMANY | FRANCE | 1995 | 52531746.67 |
| GERMANY | FRANCE | 1996 | 52520549.02 |

## 2.11 National Market Share Query (Q8)

This query determines how the market share of a given nation within a given region has changed over two years for a given part type.

### 2.11.1 Business Question

The market share for a given nation within a given region is defined as the fraction of the revenue, the sum of [l_extendedprice * (1-l_discount)], from the products of a specified type in that region that was supplied by suppliers from the given nation. The query determines this for the years 1995 and 1996 presented in this order.

### 2.11.2 Functional Query Definition

```
select
        o_year,
        sum(case
                when nation = '[NATION]'
                then volume
                else 0
        end) / sum(volume) as mkt_share
from (
        select
                extract(year from o_orderdate) as o_year,
                l_extendedprice * (1-l_discount) as volume,
                n2.n_name as nation
        from
                part,
                supplier,
                lineitem,
                orders,
                customer,
                nation n1,
                nation n2,
                region
        where
                p_partkey = l_partkey
                and s_suppkey = l_suppkey
                and l_orderkey = o_orderkey
                and o_custkey = c_custkey
                and c_nationkey = n1.n_nationkey
                and n1.n_regionkey = r_regionkey
                and r_name = '[REGION]'
                and s_nationkey = n2.n_nationkey
                and o_orderdate between date '1995-01-01' and date '1996-12-31'
                and p_type = '[TYPE]'
```

```
        ) as all_nations
group by
        o_year
order by
        o_year;
```

### 2.11.3 Substitution Parameters

Values for the following substitution parameters must be generated and used to build the executable query text:

1. NATION is randomly selected within the list of values defined for N_NAME in Clause 4.2.3;
2. REGION is the value defined in Clause 4.2.3 for R_NAME where R_REGIONKEY corresponds to N_REGIONKEY for the selected NATION in item 1 above;
3. TYPE is randomly selected within the list of 3-syllable strings defined for Types in Clause 4.2.2.12.

### 2.11.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. NATION = BRAZIL;
2. REGION = AMERICA;
3. TYPE = ECONOMY ANODIZED STEEL.

Query validation output data:

| YEAR | MKT_SHARE |
|---|---|
| 1995 | .03 |
| 1996 | .04 |

## 2.12 Product Type Profit Measure Query (Q9)

This query determines how much profit is made on a given line of parts, broken out by supplier nation and year.

### 2.12.1 Business Question

The Product Type Profit Measure Query finds, for each nation and each year, the profit for all parts ordered in that year that contain a specified substring in their names and that were filled by a supplier in that nation. The profit is defined as the sum of [(l_extendedprice*(1-l_discount)) - (ps_supplycost * l_quantity)] for all lineitems describing parts in the specified line. The query lists the nations in ascending alphabetical order and, for each nation, the year and profit in descending order by year (most recent first).

### 2.12.2 Functional Query Definition

```
select
        nation,
        o_year,
        sum(amount) as sum_profit
from (
        select
                n_name as nation,
                extract(year from o_orderdate) as o_year,
                l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
        from
                part,
                supplier,
                lineitem,
                partsupp,
                orders,
                nation
        where
                s_suppkey = l_suppkey
                and ps_suppkey = l_suppkey
                and ps_partkey = l_partkey
                and p_partkey = l_partkey
                and o_orderkey = l_orderkey
                and s_nationkey = n_nationkey
                and p_name like '%[COLOR]%'
        ) as profit
group by
        nation,
        o_year
order by
        nation,
        o_year desc;
```

### 2.12.3 Substitution Parameters

Values for the following substitution parameter must be generated and used to build the executable query text:

1. COLOR is randomly selected within the list of values defined for the generation of P_NAME in Clause 4.2.3.

### 2.12.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. COLOR = green.

Query validation output data:

| NATION | YEAR | SUM_PROFIT |
|---|---|---|
| ALGERIA | 1998 | 31342867.24 |
| ALGERIA | 1997 | 57138193.03 |
| ALGERIA | 1996 | 56140140.13 |
| ALGERIA | 1995 | 53051469.66 |
| ALGERIA | 1994 | 53867582.12 |
| [165 more rows] | | |
| VIETNAM | 1996 | 50488161.42 |
| VIETNAM | 1995 | 49658284.61 |
| VIETNAM | 1994 | 50596057.26 |
| VIETNAM | 1993 | 50953919.14 |
| VIETNAM | 1992 | 49613838.33 |

## 2.13 Returned Item Reporting Query (Q10)

The query identifies customers who might be having problems with the parts that are shipped to them.

### 2.13.1 Business question

The Returned Item Reporting Query finds the top 20 customers, in terms of their effect on lost revenue for a given quarter, who have returned parts. The query considers only parts that were ordered in the specified quarter. The query lists the customer's name, address, nation, phone number, account balance, comment information and revenue lost. The customers are listed in descending order of lost revenue. Revenue lost is defined as sum(l_extendedprice*(1-l_discount)) for all qualifying lineitems.

### 2.13.2 Functional Query Definition

Return the first 20 selected rows

```
select
        c_custkey,
        c_name,
        sum(l_extendedprice * (1 - l_discount)) as revenue,
        c_acctbal,
        n_name,
        c_address,
        c_phone,
        c_comment
from
        customer,
        orders,
        lineitem,
        nation
where
        c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate >= date '[DATE]'
        and o_orderdate < date '[DATE]' + interval '3' month
        and l_returnflag = 'R'
        and c_nationkey = n_nationkey
group by
        c_custkey,
        c_name,
        c_acctbal,
        c_phone,
        n_name,
        c_address,
        c_comment
order by
        revenue desc;
```

### 2.13.3 Substitution Parameters

Values for the following substitution parameter must be generated and used to build the executable query text:

1. DATE is the first day of a randomly selected month between the first month of 1993 and the 12th month of 1994.

### 2.13.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. DATE = 1993-10-01.

Query validation output data:

| C_CUSTKEY | C_NAME | REVENUE | C_ACCTBAL | N_NAME |
|---|---|---|---|---|
| 57040 | Customer#000057040 | 734235.24 | 632.87 | JAPAN |
| 143347 | Customer#000143347 | 721002.70 | 2557.47 | EGYPT |
| 60838 | Customer#000060838 | 679127.31 | 2454.77 | BRAZIL |
| 101998 | Customer#000101998 | 637029.57 | 3790.89 | UNITED KINGDOM |
| 125341 | Customer#000125341 | 633508.09 | 4983.51 | GERMANY |

[10 more rows]

| | | | | |
|---|---|---|---|---|
| 110246 | Customer#000110246 | 566842.98 | 7763.35 | VIETNAM |
| 142549 | Customer#000142549 | 563537.24 | 5085.99 | INDONESIA |
| 146149 | Customer#000146149 | 557254.99 | 1791.55 | ROMANIA |
| 52528 | Customer#000052528 | 556397.35 | 551.79 | ARGENTINA |
| 23431 | Customer#000023431 | 554269.54 | 3381.86 | ROMANIA |

| C_ADDRESS | C_PHONE | C_COMMENT |
|---|---|---|
| Eioyzjf4pp | 22-895-641-3466 | requests sleep blithely about the furiously i |
| 1aReFYv,Kw4 | 14-742-935-3718 | fluffily bold excuses haggle finally after the u |
| 64EaJ5vMAHWJlBOxJklpNc2RJiWE | 12-913-494-9813 | furiously even pinto beans integrate under the ruthless foxes; ironic, even dolphins across the slyl |
| 01c9CILnNtfOQYmZj | 33-593-865-6378 | accounts doze blithely! enticing, final deposits sleep blithely special accounts. slyly express accounts pla |

| | | |
|---|---|---|
| S29ODD6bceU8QSuuE JznkNaK | 17-582-695-5962 | quickly express requests wake quickly blithely |
| [10 more rows] | | |
| 7KzflgX MDOq7sOkI | 31-943-426-9837 | dolphins sleep blithely among the slyly final |
| ChqEoK43OysjdHbtK Cp6dKqjNyvvi9 | 19-955-562-2398 | regular, unusual dependencies boost slyly; ironic attainments nag fluffily into the unusual packages? |
| s87fvzFQpU | 29-744-164-6487 | silent, unusual requests detect quickly slyly regul |
| NFztyTOR10UOJ | 11-208-192-3205 | unusual requests detect. slyly dogged theodo-lites use slyly. deposit |
| HgiV0phqhaIa9aydN oIlb | 29-915-458-2654 | instructions nag quickly. furiously bold accounts cajol |

## 2.14 Important Stock Identification Query (Q11)

This query finds the most important subset of suppliers' stock in a given nation.

### 2.14.1 Business Question

The Important Stock Identification Query finds, from scanning the available stock of suppliers in a given nation, all the parts that represent a significant percentage of the total value of all available parts. The query displays the part number and the value of those parts in descending order of value.

### 2.14.2 Functional Query Definition

```
select
        ps_partkey,
        sum(ps_supplycost * ps_availqty) as value
from
        partsupp,
        supplier,
        nation
where
        ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = '[NATION]'
group by
        ps_partkey having
                sum(ps_supplycost * ps_availqty) > (
                        select
                                sum(ps_supplycost * ps_availqty) * [FRACTION]
                        from
                                partsupp,
                                supplier,
                                nation
                        where
                                ps_suppkey = s_suppkey
                                and s_nationkey = n_nationkey
                                and n_name = '[NATION]'
                )
order by
        value desc;
```

### 2.14.3 Substitution Parameters

Values for the following substitution parameter must be generated and used to build the executable query text:

1. NATION is randomly selected within the list of values defined for N_NAME in Clause 4.2.3;
2. FRACTION is chosen as 0.0001 / SF.

### 2.14.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. NATION = GERMANY;
2. FRACTION = 0.0001.

Query validation output data:

| PS_PARTKEY | VALUE |
|---|---|
| 129760 | 17538456.86 |
| 166726 | 16503353.92 |
| 191287 | 16474801.97 |
| 161758 | 16101755.54 |
| 34452 | 15983844.72 |

[1038 More Rows]

| | |
|---|---|
| 154731 | 7888301.33 |
| 101674 | 7879324.60 |
| 51968 | 7879102.21 |
| 72073 | 7877736.11 |
| 5182 | 7874521.73 |

## 2.15 Shipping Modes and Order Priority Query (Q12)

This query determines whether selecting less expensive modes of shipping is negatively affecting the critical-priority orders by causing more parts to be received by customers after the committed date.

### 2.15.1 Business Question

The Shipping Modes and Order Priority Query counts, by ship mode, for lineitems actually received by customers in a given year, the number of lineitems belonging to orders for which the l_receiptdate exceeds the l_commitdate for two different specified ship modes. Only lineitems that were actually shipped before the l_commitdate are considered. The late lineitems are partitioned into two groups, those with priority URGENT or HIGH, and those with a priority other than URGENT or HIGH.

### 2.15.2 Functional Query Definition

```
select
        l_shipmode,
        sum(case
                when o_orderpriority ='1-URGENT'
                        or o_orderpriority ='2-HIGH'
                then 1
                else 0
        end) as high_line_count,
        sum(case
                when o_orderpriority <> '1-URGENT'
                        and o_orderpriority <> '2-HIGH'
                then 1
                else 0
        end) as low_line_count
from
        orders,
        lineitem
where
        o_orderkey = l_orderkey
        and l_shipmode in ('[SHIPMODE1]', '[SHIPMODE2]')
        and l_commitdate < l_receiptdate
        and l_shipdate < l_commitdate
        and l_receiptdate >= date '[DATE]'
        and l_receiptdate < date '[DATE]' + interval '1' year
group by
        l_shipmode
order by
        l_shipmode;
```

### 2.15.3 Substitution Parameters

Values for the following substitution parameters must be generated and used to build the executable query text:

1. SHIPMODE1 is randomly selected within the list of values defined for Modes in Clause 4.2.2.12;
2. SHIPMODE2 is randomly selected within the list of values defined for Modes in Clause 4.2.2.12 and must be different from the value selected for SHIPMODE1 in item 1;

3. DATE is the first of January of a randomly selected year within [1993 .. 1997].

### 2.15.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. SHIPMODE1 = MAIL;
2. SHIPMODE2 = SHIP;
3. DATE = 1994-01-01.

Query validation output data:

| L_SHIPMODE | HIGH_LINE_COUNT | LOW_LINE_COUNT |
|---|---|---|
| MAIL | 6202 | 9324 |
| SHIP | 6200 | 9262 |

## 2.16 Customer Distribution Query (Q13)

This query seeks relationships between customers and the size of their orders.

### 2.16.1 Business Question

This query determines the distribution of customers by the number of orders they have made, including customers who have no record of orders, past or present. It counts and reports how many customers have no orders, how many have 1, 2, 3, etc. A check is made to ensure that the orders counted do not fall into one of several special categories of orders. Special categories are identified in the order comment column by looking for a particular pattern.

### 2.16.2 Functional Query Definition

```
select
        c_count, count(*) as custdist
from (
        select
                c_custkey,
                count(o_orderkey)
        from
                customer left outer join orders on
                        c_custkey = o_custkey
                        and o_comment not like '%[WORD1]%[WORD2]%'
        group by
                c_custkey
        )as c_orders (c_custkey, c_count)
group by
        c_count
order by
        custdist desc,
        c_count desc;
```

### 2.16.3 Substitution Parameters

1. WORD1 is randomly selected from 4 possible values: special, pending, unusual, express.
2. WORD2 is randomly selected from 4 possible values: packages, requests, accounts, deposits.

### 2.16.4 Query Validation

For validation against the qualification database the query must be executed using the following substitution parameters and must produce the following output data:

Values for substitution parameters:

1. WORD1 = special.
2. WORD2 = requests.

Query validation output data:

| C_COUNT | CUSTDIST |
|---|---|

| | |
|---|---|
| 0 | 50004 |
| 9 | 6641 |
| 10 | 6566 |
| 11 | 6058 |
| 8 | 5949 |

[32 more rows]

| | |
|---|---|
| 37 | 7 |
| 40 | 4 |
| 38 | 4 |
| 39 | 2 |
| 41 | 1 |

## 2.17 Promotion Effect Query (Q14)

This query monitors the market response to a promotion such as TV advertisements or a special campaign.

### 2.17.1 Business Question

The Promotion Effect Query determines what percentage of the revenue in a given year and month was derived from promotional parts. The query considers only parts actually shipped in that month and gives the percentage. Revenue is defined as (l_extendedprice * (1-l_discount)).

### 2.17.2 Functional Query Definition

```
select
        100.00 * sum(case
                when p_type like 'PROMO%'
                then l_extendedprice*(1-l_discount)
                else 0
        end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
        lineitem,
        part
where
        l_partkey = p_partkey
        and l_shipdate >= date '[DATE]'
        and l_shipdate < date '[DATE]' + interval '1' month;
```

### 2.17.3 Substitution Parameters

Values for the following substitution parameter must be generated and used to build the executable query text:

1. DATE is the first day of a month randomly selected from a random year within [1993 .. 1997].

### 2.17.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. DATE = 1995-09-01.

Query validation output data:

| PROMO_REVENUE |
|---|
| 16.38 |

## 2.18 Top Supplier Query (Q15)

This query determines the top supplier so it can be rewarded, given more business, or identified for special recognition.

### 2.18.1 Business Question

The Top Supplier Query finds the supplier who contributed the most to the overall revenue for parts shipped during a given quarter of a given year. In case of a tie, the query lists all suppliers whose contribution was equal to the maximum, presented in supplier number order.

### 2.18.2 Functional Query Definition

```
create view revenue[STREAM_ID] (supplier_no, total_revenue) as
        select
                l_suppkey,
                sum(l_extendedprice * (1 - l_discount))
        from
                lineitem
        where
                l_shipdate >= date '[DATE]'
                and l_shipdate < date '[DATE]' + interval '3' month
        group by
                l_suppkey;

select
        s_suppkey,
        s_name,
        s_address,
        s_phone,
        total_revenue
from
        supplier,
        revenue[STREAM_ID]
where
        s_suppkey = supplier_no
        and total_revenue = (
                select
                        max(total_revenue)
                from
                        revenue[STREAM_ID]
        )
order by
        s_suppkey;

drop view revenue[STREAM_ID];
```

### 2.18.3 Substitution Parameters

Values for the following substitution parameter must be generated and used to build the executable query text:

1. DATE is the first day of a randomly selected month between the first month of 1993 and the 10th month of

1997.

### 2.18.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. DATE = 1996-01-01.

Query validation output data:

| S_SUPPKEY | S_NAME | S_ADDRESS | S_PHONE | TOTAL_REVENUE |
|---|---|---|---|---|
| 8449 | Supplier#000008449 | Wp34zim9qYFbVctdW | 20-469-856-8873 | 1772627.21 |

## 2.19 Parts/Supplier Relationship Query (Q16)

This query finds out how many suppliers can supply parts with given attributes. It might be used, for example, to determine whether there is a sufficient number of suppliers for heavily ordered parts.

### 2.19.1 Business Question

The Parts/Supplier Relationship Query counts the number of suppliers who can supply parts that satisfy a particular customer's requirements. The customer is interested in parts of eight different sizes as long as they are not of a given type, not of a given brand, and not from a supplier who has had complaints registered at the Better Business Bureau. Results must be presented in descending count and ascending brand, type, and size.

### 2.19.2 Functional Query Definition

```
select
        p_brand,
        p_type,
        p_size,
        count(distinct ps_suppkey) as supplier_cnt
from
        partsupp,
        part
where
        p_partkey = ps_partkey
        and p_brand <> '[BRAND]'
        and p_type not like '[TYPE]%'
        and p_size in ([SIZE1], [SIZE2], [SIZE3], [SIZE4], [SIZE5], [SIZE6], [SIZE7], [SIZE8])
        and ps_suppkey not in (
                select
                        s_suppkey
                from
                        supplier
                where
                        s_comment like '%Customer%Complaints%'
        )
group by
        p_brand,
        p_type,
        p_size
order by
        supplier_cnt desc,
        p_brand,
        p_type,
        p_size;
```

### 2.19.3 Substitution Parameters

Values for the following substitution parameters must be generated and used to build the executable query text:

1. BRAND = Brand#MN where M and N are two single character strings representing two numbers randomly and independently selected within [1 .. 5];

2. TYPE is made of the first 2 syllables of a string randomly selected within the list of 3-syllable strings defined for Types in Clause 4.2.2.12;
3. SIZE1 is randomly selected as a set of eight different values within [1 .. 50];
4. SIZE2 is randomly selected as a set of eight different values within [1 .. 50];
5. SIZE3 is randomly selected as a set of eight different values within [1 .. 50];
6. SIZE4 is randomly selected as a set of eight different values within [1 .. 50];
7. SIZE5 is randomly selected as a set of eight different values within [1 .. 50];
8. SIZE6 is randomly selected as a set of eight different values within [1 .. 50];
9. SIZE7 is randomly selected as a set of eight different values within [1 .. 50];
10. SIZE8 is randomly selected as a set of eight different values within [1 .. 50].

### 2.19.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. BRAND = Brand#45.
2. TYPE = MEDIUM POLISHED .
3. SIZE1 = 49
4. SIZE2 = 14
5. SIZE3 = 23
6. SIZE4 = 45
7. SIZE5 = 19
8. SIZE6 = 3
9. SIZE7 = 36
10. SIZE8 = 9.

Query validation output data:

| P_BRAND | P_TYPE | P_SIZE | SUPPLIER_CNT |
|---|---|---|---|
| Brand#41 | MEDIUM BRUSHED TIN | 3 | 28 |
| Brand#54 | STANDARD BRUSHED COPPER | 14 | 27 |
| Brand#11 | STANDARD BRUSHED TIN | 23 | 24 |
| Brand#11 | STANDARD BURNISHED BRASS | 36 | 24 |
| Brand#15 | MEDIUM ANODIZED NICKEL | 3 | 24 |

[18,304 more rows]

| | | | |
|---|---|---|---|
| Brand#52 | MEDIUM BRUSHED BRASS | 45 | 3 |
| Brand#53 | MEDIUM BRUSHED TIN | 45 | 3 |
| Brand#54 | ECONOMY POLISHED BRASS | 9 | 3 |
| Brand#55 | PROMO PLATED BRASS | 19 | 3 |
| Brand#55 | STANDARD PLATED TIN | 49 | 3 |

## 2.20 Small-Quantity-Order Revenue Query (Q17)

This query determines how much average yearly revenue would be lost if orders were no longer filled for small quantities of certain parts. This may reduce overhead expenses by concentrating sales on larger shipments.

### 2.20.1 Business Question

The Small-Quantity-Order Revenue Query considers parts of a given brand and with a given container type and determines the average lineitem quantity of such parts ordered for all orders (past and pending) in the 7-year database. What would be the average yearly gross (undiscounted) loss in revenue if orders for these parts with a quantity of less than 20% of this average were no longer taken?

### 2.20.2 Functional Query Definition

```
select
        sum(l_extendedprice) / 7.0 as avg_yearly
from
        lineitem,
        part
where
        p_partkey = l_partkey
        and p_brand = '[BRAND]'
        and p_container = '[CONTAINER]'
        and l_quantity < (
                select
                        0.2 * avg(l_quantity)
                from
                        lineitem
                where
                        l_partkey = p_partkey
        );
```

### 2.20.3 Substitution Parameters

Values for the following substitution parameter must be generated and used to build the executable query text:

1. BRAND = 'Brand#MN' where MN is a two character string representing two numbers randomly and independently selected within [1 .. 5];
2. CONTAINER is randomly selected within the list of 2-syllable strings defined for Containers in Clause 4.2.2.12.

### 2.20.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. BRAND = Brand#23;
2. CONTAINER = MED BOX.

Query validation output data:

| AVG_YEARLY |
|---|
| 348406.05 |

## 2.21 Large Volume Customer Query (Q18)

The Large Volume Customer Query ranks customers based on their having placed a large quantity order. Large quantity orders are defined as those orders whose total quantity is above a certain level.

### 2.21.1 Business Question

The Large Volume Customer Query finds a list of the top 100 customers who have ever placed large quantity orders. The query lists the customer name, customer key, the order key, date and total price and the quantity for the order.

### 2.21.2 Functional Query Definition

Return the first 100 selected rows

```
select
        c_name,
        c_custkey,
        o_orderkey,
        o_orderdate,
        o_totalprice,
        sum(l_quantity)
from
        customer,
        orders,
        lineitem
where
        o_orderkey in (
                select
                        l_orderkey
                from
                        lineitem
                group by
                        l_orderkey having
                                sum(l_quantity) > [QUANTITY]
        )
        and c_custkey = o_custkey
        and o_orderkey = l_orderkey
group by
        c_name,
        c_custkey,
        o_orderkey,
        o_orderdate,
        o_totalprice
order by
        o_totalprice desc,
        o_orderdate;
```

### 2.21.3 Substitution Parameters

Values for the following substitution parameter must be generated and used to build the executable query text:

1. QUANTITY is randomly selected within [312..315].

### 2.21.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. QUANTITY = 300

Query validation output data:

| C_NAME | C_CUSTKEY | O_ORDERKEY | O_ORDERDATE | O_TOTALPRICE | Sum(L_QUANTITY) |
|---|---|---|---|---|---|
| Customer#000128120 | 128120 | 4722021 | 1994-04-07 | 544089.09 | 323.00 |
| Customer#000144617 | 144617 | 3043270 | 1997-02-12 | 530604.44 | 317.00 |
| Customer#000013940 | 13940 | 2232932 | 1997-04-13 | 522720.61 | 304.00 |
| Customer#000066790 | 66790 | 2199712 | 1996-09-30 | 515531.82 | 327.00 |
| Customer#000046435 | 46435 | 4745607 | 1997-07-03 | 508047.99 | 309.00 |
| [47 more rows] | | | | | |
| Customer#000069904 | 69904 | 1742403 | 1996-10-19 | 408513.00 | 305.00 |
| Customer#000017746 | 17746 | 6882 | 1997-04-09 | 408446.93 | 303.00 |
| Customer#000013072 | 13072 | 1481925 | 1998-03-15 | 399195.47 | 301.00 |
| Customer#000082441 | 82441 | 857959 | 1994-02-07 | 382579.74 | 305.00 |
| Customer#000088703 | 88703 | 2995076 | 1994-01-30 | 363812.12 | 302.00 |

## 2.22 Discounted Revenue Query (Q19)

The Discounted Revenue Query reports the gross discounted revenue attributed to the sale of selected parts handled in a particular manner. This query is an example of code such as might be produced programmatically by a data mining tool.

### 2.22.1 Business Question

The Discounted Revenue query finds the gross discounted revenue for all orders for three different types of parts that were shipped by air or delivered in person . Parts are selected based on the combination of specific brands, a list of containers, and a range of sizes.

### 2.22.2 Functional Query Definition

```
select
        sum(l_extendedprice * (1 - l_discount) ) as revenue
from
        lineitem,
        part
where
        (
                p_partkey = l_partkey
                and p_brand = '[BRAND1]'
                and p_container in ( 'SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
                and l_quantity >= [QUANTITY1] and l_quantity <= [QUANTITY1] + 10
                and p_size between 1 and 5
                and l_shipmode in ('AIR', 'AIR REG')
                and l_shipinstruct = 'DELIVER IN PERSON'
        )
        or
        (
                p_partkey = l_partkey
                and p_brand = '[BRAND2]'
                and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
                and l_quantity >= [QUANTITY2] and l_quantity <= [QUANTITY2] + 10
                and p_size between 1 and 10
                and l_shipmode in ('AIR', 'AIR REG')
                and l_shipinstruct = 'DELIVER IN PERSON'
        )
        or
                (
                p_partkey = l_partkey
                and p_brand = '[BRAND3]'
                and p_container in ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
                and l_quantity >= [QUANTITY3] and l_quantity <= [QUANTITY3] + 10
                and p_size between 1 and 15
                and l_shipmode in ('AIR', 'AIR REG')
                and l_shipinstruct = 'DELIVER IN PERSON'
```

);

### 2.22.3 Substitution Parameters

1. QUANTITY1 is randomly selected within [1..10].
2. QUANTITY2 is randomly selected within [10..20].
3. QUANTITY3 is randomly selected within [20..30].
4. BRAND1, BRAND2, BRAND3 = 'Brand#MN' where each MN is a two character string representing two numbers randomly and independently selected within [1 .. 5]

### 2.22.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. QUANTITY1 = 1.
2. QUANTITY2 = 10.
3. QUANTITY3 = 20.
4. BRAND1 = Brand#12.
5. BRAND2 = Brand#23.
6. BRAND3 = Brand#34.

Query validation output data:

| REVENUE |
|---|
| 3083843.0578 |

## 2.23 Potential Part Promotion Query (Q20)

The Potential Part Promotion Query identifies suppliers in a particular nation having selected parts that may be candidates for a promotional offer.

### 2.23.1 Business Question

The Potential Part Promotion query identifies suppliers who have an excess of a given part available; an excess is defined to be more than 50% of the parts like the given part that the supplier shipped in a given year for a given nation. Only parts whose names share a certain naming convention are considered.

### 2.23.2 Functional Query Definition

```
select
        s_name,
        s_address
from
        supplier, nation
where
        s_suppkey in (
                select
                        ps_suppkey
                from
                        partsupp
                where
                        ps_partkey in (
                                select
                                        p_partkey
                                from
                                        part
                                where
                                        p_name like '[COLOR]%'
                        )
                and ps_availqty > (
                        select
                                0.5 * sum(l_quantity)
                        from
                                lineitem
                        where
                                l_partkey = ps_partkey
                                and l_suppkey = ps_suppkey
                                and l_shipdate >= date('[DATE]')
                                and l_shipdate < date('[DATE]') + interval '1' year
                )
```

```
        )
        and s_nationkey = n_nationkey
        and n_name = '[NATION]'
order by
        s_name;
```

### 2.23.3 Substitution Parameters

1. COLOR is randomly selected within the list of values defined for the generation of P_NAME.
2. DATE is the first of January of a randomly selected year within 1993..1997.
3. NATION is randomly selected within the list of values defined for N_NAME in Clause 4.2.3.

### 2.23.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. COLOR = forest.
2. DATE = 1994-01-01.
3. NATION = CANADA.

Query validation output data:

| S_NAME | S_ADDRESS |
|---|---|
| Supplier#000000020 | iybAE,RmTymrZVYaFZva2SH,j |
| Supplier#000000091 | YV45D7TkfdQanOOZ7q9QxkyGUapU1oOWU6q3 |
| Supplier#000000197 | YC2Acon6kjY3zj3Fbxs2k4Vdf7X0cd2F |
| Supplier#000000226 | 83qOdU2EYRdPQAQhEtn GRZEd |
| Supplier#000000285 | Br7e1nnt1yxrw6ImgpJ7YdhFDjuBf |
| [194 more rows] | |
| Supplier#000009862 | rJzweWeN58 |
| Supplier#000009868 | ROjGgx5gvtkmnUUoeyy7v |
| Supplier#000009869 | ucLqxzrpBTRMewGSM29t0rNTM30g1Tu3Xgg3mKag |
| Supplier#000009899 | 7XdpAHrzr1t,UQFZE |
| Supplier#000009974 | 7wJ,J5DKcxSU4Kp1cQLpbcAvB5AsvKT |

## 2.24 Suppliers Who Kept Orders Waiting Query (Q21)

This query identifies certain suppliers who were not able to ship required parts in a timely manner.

### 2.24.1 Business Question

The Suppliers Who Kept Orders Waiting query identifies suppliers, for a given nation, whose product was part of a multi-supplier order (with current status of 'F') where they were the only supplier who failed to meet the committed delivery date.

### 2.24.2 Functional Query Definition

Return the first 100 selected rows.

```
select
        s_name,
        count(*) as numwait
from
        supplier,
        lineitem l1,
        orders,
        nation
where
        s_suppkey = l1.l_suppkey
        and o_orderkey = l1.l_orderkey
        and o_orderstatus = 'F'
        and l1.l_receiptdate > l1.l_commitdate
        and exists (
                select
                        *
                from
                        lineitem l2
                where
                        l2.l_orderkey = l1.l_orderkey
                        and l2.l_suppkey <> l1.l_suppkey
        )
        and not exists (
                select
                        *
                from
                        lineitem l3
                where
                        l3.l_orderkey = l1.l_orderkey
                        and l3.l_suppkey <> l1.l_suppkey
                        and l3.l_receiptdate > l3.l_commitdate
```

```
        )
        and s_nationkey = n_nationkey
        and n_name = '[NATION]'
group by
        s_name
order by
        numwait desc,
        s_name;
```

### 2.24.3 Substitution Parameters

1. NATION is randomly selected within the list of values defined for N_NAME in Clause 4.2.3.

### 2.24.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:

1. NATION = SAUDI ARABIA.

Query validation output data:

| S_NAME | NUMWAIT |
|---|---|
| Supplier#000002829 | 20 |
| Supplier#000005808 | 18 |
| Supplier#000000262 | 17 |
| Supplier#000000496 | 17 |
| Supplier#000002160 | 17 |
| [90 more rows] | |
| Supplier#000001916 | 12 |
| Supplier#000001925 | 12 |
| Supplier#000002039 | 12 |
| Supplier#000002357 | 12 |
| Supplier#000002483 | 12 |

## 2.25 Global Sales Opportunity Query (Q22)

The Global Sales Opportunity Query identifies geographies where there are customers who may be likely to make a purchase.

### 2.25.1 Business Question

This query counts how many customers within a specific range of country codes have not placed orders for 7 years but who have a greater than average “positive” account balance. It also reflects the magnitude of that balance. Country code is defined as the first two characters of c_phone.

### 2.25.2 Functional Query Definition

```
select
        cntrycode,
        count(*) as numcust,
        sum(c_acctbal) as totacctbal
from (
        select
                substring(c_phone from 1 for 2) as cntrycode,
                c_acctbal
        from
                customer
        where
                substring(c_phone from 1 for 2) in
                        ('[I1]','[I2]’,'[I3]','[I4]','[I5]','[I6]','[I7]')
                and c_acctbal > (
                        select
                                avg(c_acctbal)
                        from
                                customer
                        where
                                c_acctbal > 0.00
                                and substr (c_phone from 1 for 2) in
                                        ('[I1]','[I2]','[I3]','[I4]','[I5]','[I6]','[I7]')
                )
                and not exists (
                        select
                                *
                        from
                                orders
                        where
                                o_custkey = c_custkey
                )
```

```
        ) as custsale
group by
        cntrycode
order by
        cntrycode;
```

### 2.25.3 Substitution Parameters

1. I1 … I7 are randomly selected without repetition from the possible values for Country code as defined in Clause 4.2.2.9.

### 2.25.4 Query Validation

For validation against the qualification database the query must be executed using the following substitution parameters and must produce the following output data:

1. I1 = 13.
2. I2 = 31.
3. I3 = 23.
4. I4 = 29.
5. I5 = 30.
6. I6 = 18.
7. I7 = 17.

Query validation output data:

| CNTRYCODE | NUMCUST | TOTACCTBAL |
|---|---|---|
| 13 | 888 | 6737713.99 |
| 17 | 861 | 6460573.72 |
| 18 | 964 | 7236687.40 |
| 23 | 892 | 6701457.95 |
| 29 | 948 | 7158866.63 |
| 30 | 909 | 6808436.13 |
| 31 | 922 | 6806670.18 |

## 2.26 General Requirements for Refresh functions

### 2.26.1 Refresh Function Overview

Each refresh function is defined by the following components:

- The **business rationale,** which illustrates the business context in which the refresh functions could be used;
- The **refresh function definition,** which defines in pseudo-code the function to be performed by the refresh function;
- The **refresh data set,** which defines the set of rows to be inserted or deleted by each execution of the refresh function into or from the ORDERS and LINEITEM tables. This set of rows represents 0.1% of the initial population of these two tables (see Table 4: LINEITEM Cardinality).

### 2.26.2 Transaction Requirements for Refresh functions

The execution of each refresh function (RF1 or RF2) can be decomposed into any number of database transactions as long as the following conditions are met:

- All ACID properties are met;
- Each atomic transaction includes a sufficient number of data modifications to maintain the logical database consistency. For example, when adding or deleting a new order, the LINEITEM and the ORDERS tables are both modified within the same transaction;
- An output message is sent when the last transaction of the refresh function has completed successfully.

### 2.26.3 Refresh Function Compliance

2.26.3.1 The benchmark specification does not place any requirements on the implementation of the refresh functions other than their functional equivalence to the refresh function definition and compliance with Clause 2.26.2. For RF1 and RF2 only, the implementation is permitted to:

- Use any language to write the code for the refresh functions;
- Pre-process, compile and link the executable code on the SUT at any time prior to or during the measurement interval;
- Provide the SUT with the data to be inserted by RF1 or the set of keys for the rows to be deleted by RF2 prior to the execution of the benchmark (this specifically does not allow pre-execution of the refresh functions).

**Comment**: The intent is to separate the resources required to generate the data to be inserted (or the set of key for the rows to be deleted) from the resources required to execute insert and delete operations against the database.

- Group the individual refresh functions into transactions and organize their execution serially or in parallel. This grouping may be different in the power test and in the throughput test.

2.26.3.2 The refresh functions do not produce any output other than a message of successful completion.

2.26.3.3 The proper implementation of the refresh functions must be validated by the independent auditor who may request additional tests to ascertain that the refresh functions execute in accordance with the benchmark requirements.

## 2.27 New Sales Refresh Function (RF1)

This refresh function adds new sales information to the database.

### 2.27.1 Business Rationale

The New Sales refresh function inserts new rows into the ORDERS and LINEITEM tables in the database following the scaling and data generation methods used to populate the database.

### 2.27.2 Refresh Function Definition

```
   LOOP (SF * 1500) TIMES
   INSERT a new row into the ORDERS table
   LOOP RANDOM(1, 7) TIMES
            INSERT a new row into the LINEITEM table
   END LOOP
END LOOP
```

**Comment:** The refresh functions can be implemented with much greater flexibility than the queries (see Clause 2.26.3). The definition provided here is an example only. Test sponsors may wish to explore other implementations.

### 2.27.3 Refresh Data Set

The set of rows to be inserted must be produced by DBGEN (or a modified version of DBGEN, see Clause 4.2.1) using the -U option. This option will produce as many sets of rows as required for use in multi-stream tests.

## 2.28 Old Sales Refresh Function (RF2)

This refresh function removes old sales information from the database.

### 2.28.1 Business Rationale

The Old Sales refresh function removes rows from the ORDERS and LINEITEM tables in the database to emulate the removal of stale or obsolete information.

### 2.28.2 Refresh Function Definition

```
   LOOP (SF * 1500) TIMES
   DELETE FROM ORDERS WHERE O_ORDERKEY = [value]
   DELETE FROM LINEITEM WHERE L_ORDERKEY = [value]
END LOOP
```

**Comment:** The refresh functions can be implemented with much greater flexibility than the queries (see Clause 2.26.3). The definition provided here is an example only. Test sponsors may wish to explore other implementation

### 2.28.3 Refresh Data Set

The primary key values for the set of rows to be deleted must be produced by DBGEN (or a modified version of DBGEN, see Clause 4.2.1) using the -U option. This option will produce as many sets of primary keys as required for use in multi-stream throughput tests. The rows being deleted begin with the first row of each of the two targeted tables.

## 2.29 Database Evolution Process

The test sponsor must assure the correctness of the database for each run within the performance test.

This is accomplished by ”evolving” the test database, keeping track of which set of inserted and deleted rows should be used by RF1 and RF2 for each run (see Clause 5.1.1.4).

**Comment**: It is explicitly not permitted to rebuild or reload the test database during the performance test (see Clause 5.1.1.3).

2.29.1 The test database may be endlessly reused if the test sponsor keeps careful track of how many pairs of refresh functions RF1/RF2 have been executed and completed successfully. For example, a test sponsor running five streams would execute one RF1/RF2 pair during the power test using the first set of insert/delete rows produced by DBGEN (see Clause 4.2.1). The throughput test would then execute the next five RF1/RF2 pairs using the second through the sixth sets of inset/delete rows produced by DBGEN. The next run would use the sets of insert/delete rows produced by DBGEN for the seventh RF1/RF2 pair, and continue from there.

# 3: DATABASE SYSTEM PROPERTIES

## 3.1 The ACID Properties

3.1.1 The ACID (Atomicity, Consistency, Isolation, and Durability) properties of transaction processing systems must be supported by the system under test during the timed portion of this benchmark. Since TPC-H is not a transaction processing benchmark, the ACID properties must be evaluated outside the timed portion of the test. It is the intent of this section to informally define the ACID properties and to specify a series of tests that can be performed to demonstrate that these properties are met.

3.1.2 While it is required for the system under test (SUT) to support the ACID properties defined in this Clause, the execution of the corresponding ACID tests is only required in lieu of supplying other sufficient evidence of the SUT's support for these ACID properties. The existence of another published TPC-H benchmark for which support for the ACID properties have been demonstrated using the tests defined in this Clause may be sufficient evidence that the new SUT supports some or all of the required ACID properties. The determination of whether previously published TPC-H test results are sufficient evidence of the above is left to the discretion of the auditor.

**Comment 1**: No finite series of tests can prove that the ACID properties are fully supported. Being able to pass the specified tests is a necessary, but not sufficient, condition for meeting the ACID requirements.

**Comment 2**: The ACID tests are intended to demonstrate that the ACID properties are supported by the SUT and enabled during the performance measurements. They are not intended to be an exhaustive quality assurance test.

3.1.3 The ACID tests must be performed against the qualification database. The same set of mechanisms used to ensure full ACID properties of the qualification database during the ACID tests must be used/enabled for the test database during the performance test. This applies both to attributes of the database and to attributes of the database session(s) used to execute the ACID and performance tests. The attributes of the session executing the ACID Query (see Clause 3.1.6.3) must be the same as those used in the performance test query stream(s) (see Clause 5.1.2.3), and the attributes of the session executing the ACID transaction (see Clause 3.1.6.2) must be the same as those used in the performance test refresh stream (see Clause 5.1.2.4).

3.1.4 The mechanisms used to ensure durability of the qualification database must be enabled for the test database. For example:

a) If the qualification database relies on undo logs to ensure atomicity, then such logging must also be enabled for the test database during the performance test, even though no transactions are aborted.

b) If the qualification database relies on a database backup to meet the durability requirement (see Clause 3.5), a backup must be taken of the test database.

c) If the qualification database relies on RAID or mirroring to meet the durability requirement (see Clause 3.5), these mechanisms must be active during the execution of the performance test.

3.1.5 The test sponsor must attest that the reported configuration would also pass the ACID tests with the test database.

### 3.1.6 The ACID Transaction and The ACID Query

Since this benchmark does not contain any OLTP transaction, a special **ACID Transaction** is defined for use in some of the ACID tests. In addition, to simplify the demonstration that ACID properties are enabled while read-only queries are executing concurrently with other activities, a special **ACID Query** is defined.

3.1.6.1 Both the ACID transaction and the ACID Query utilize a **truncation function** to guarantee arithmetic function portability and consistency of results. Define trunc(n,p) as

$$\text{trunc}(n, p) = \lfloor n * 10^{p} \rfloor / 10^{p}$$

which truncates n to the $p^{th}$ decimal place (e.g., trunc(1.357,2) = 1.35).

**Comment:** The intent of this clause is to specify the required functionality without dictating a particular implementation.

3.1.6.2 The ACID Transaction must be implemented to conform to the following transaction profile:

Given the set of input data (O_KEY, L_KEY, [delta]), with

- O_KEY selected at random from the same distribution as that used to populate L_ORDERKEY in the qualification database (see Clause 4.2.3),
- L_KEY selected at random from [1 .. M] where
  M = SELECT MAX(L_LINENUMBER) FROM LINEITEM WHERE L_ORDERKEY = O_KEY
  using the qualification database, and [delta] selected at random within [1 .. 100]:

  BEGIN TRANSACTION

  Read O_TOTALPRICE from ORDERS into [ototal] where O_ORDERKEY = [o_key]

  Read L_QUANTITY, L_EXTENDEDPRICE, L_PARTKEY, L_SUPPKEY, L_TAX, L_DISCOUNT into
  [quantity], [extprice], [pkey], [skey], [tax], [disc]
  where L_ORDERKEY = [o_key] and L_LINENUMBER = [l_key]

  Set [ototal] = [ototal] -
  trunc( trunc([extprice] * (1 - [disc]), 2) * (1 + [tax]), 2)

  Set [rprice] = trunc([extprice]/[quantity], 2)

  Set [cost] = trunc([rprice] * [delta], 2)

  Set [new_extprice] = [extprice] + [cost]

  Set [new_ototal] = trunc([new_extprice] * (1.0 - [disc]), 2)

  Set [new_ototal] = trunc([new_ototal] * (1.0 + [tax]), 2)

  Set [new_ototal] = [ototal] + [new_ototal]

  Update LINEITEM
  where L_ORDERKEY = [o_key] and L_LINENUMBER = [l_key]

  Set L_EXTENDEDPRICE = [new_extprice]

  Set L_QUANTITY = [quantity] + [delta]

  Write L_EXTENDEDPRICE, L_QUANTITY to LINEITEM

  Update ORDERS where O_ORDERKEY = [o_key]

  Set O_TOTALPRICE = [new_ototal]

  Write O_TOTALPRICE to ORDERS

  Insert Into HISTORY

Values ([pkey], [skey], [o_key], [l_key], [delta], [current_date_time])

COMMIT TRANSACTION

Return [rprice], [quantity], [tax], [disc], [extprice], [ototal] to driver

Where HISTORY is a table required only for the ACID tests and defined as follows:

| Column Name | Datatype Requirements | |
|---|---|---|
| H_P_KEY | identifier | Foreign reference to P_PARTKEY |
| H_S_KEY | identifier | Foreign reference to S_SUPPKEY |
| H_O_KEY | identifier | Foreign reference to O_ORDERKEY |
| H_L_KEY | integer | |
| H_DELTA | integer | |
| H_DATE_T | date and time to second | |

**Comment**: The values returned by the ACID Transaction are the old values, as read before the updates.

3.1.6.3 The ACID Query must be implemented to conform to the following functional query definition:

Given the input data:

- O_KEY, selected within the same distributions as those used for the population of L_ORDERKEY in the qualification database:

```
SELECT SUM(trunc(
    trunc(L_EXTENDEDPRICE * (1 - L_DISCOUNT),2) * (1 + L_TAX),2))
    FROM LINEITEM
    WHERE L_ORDERKEY = [o_key]
```

3.1.6.4 The ACID Transaction and the ACID Query must be used to demonstrate that the ACID properties are fully supported by the system under test.

3.1.6.5 Although the ACID Transaction and the ACID Query do not involve all the tables of the TPC-H database, the ACID properties must be supported for all tables of the TPC-H database.

## 3.2 Atomicity Requirements

### 3.2.1 Atomicity Property Definition

The system under test must guarantee that transactions are atomic; the system will either perform all individual operations on the data, or will assure that no partially-completed operations leave any effects on the data.

### 3.2.2 Atomicity Tests

3.2.2.1 Perform the ACID Transaction (see Clause 3.1.5) for a randomly selected set of input data and verify that the appropriate rows have been changed in the ORDERS, LINEITEM, and HISTORY tables.

3.2.2.2 Perform the ACID Transaction for a randomly selected set of input data, substituting a ROLLBACK of the transaction for the COMMIT of the transaction. Verify that the appropriate rows have not been changed in the ORDERS, LINEITEM, and HISTORY tables.

## 3.3 Consistency Requirements

### 3.3.1 Consistency Property Definition

Consistency is the property of the application that requires any execution of transactions to take the database from one consistent state to another.

### 3.3.2 Consistency Condition

3.3.2.1 A consistent state for the TPC-H database is defined to exist when:

O_TOTALPRICE =
SUM(trunc(trunc(L_EXTENDEDPRICE *(1 - L_DISCOUNT),2) * (1+L_TAX),2))

for each ORDERS and LINEITEM defined by (O_ORDERKEY = L_ORDERKEY)

3.3.2.2 A TPC-H database, when populated as defined in Clause 4.2, must meet the consistency condition defined in Clause 3.3.2.1.

3.3.2.3 If data is replicated, as permitted under Clause 1.5.10, each copy must meet the consistency condition defined in Clause 3.3.2.1.

### 3.3.3 Consistency Tests

To verify the consistency between the ORDERS, and LINEITEM tables, perform the following steps:

1. Verify that the ORDERS, and LINEITEM tables are initially consistent as defined in Clause 3.3.2.1, based on a random sample of at least 10 distinct values of O_ORDERKEY.
2. Submit at least 100 ACID Transactions from each of at least the number of execution streams ( # query streams + 1 refresh stream) used in the reported throughput test (see Clause 5.3.4). Each transaction must use values of (O_KEY, L_KEY, DELTA) randomly generated within the ranges defined in Clause 3.1.6.2. Ensure that all the values of O_ORDERKEY chosen in Step 1 are used by some transaction in Step 2.
3. Re-verify the consistency of the ORDERS, and LINEITEM tables as defined in Clause 3.3.2.1 based on the same sample values of O_ORDERKEY selected in Step 1.

## 3.4 Isolation Requirements

### 3.4.1 Isolation Property Definition

Isolation can be defined in terms of the following phenomena that may occur during the execution of concurrent database transactions (i.e., read-write transactions or read-only queries):

P0 (“Dirty Write”): Database transaction T1 reads a data element and modifies it. Database transaction T2 then modifies or deletes that data element, and performs a COMMIT. If T1 were to attempt to re-read the data element, it may receive the modified value from T2 or discover that the data element has been deleted.

P1 (“Dirty Read”): Database transaction T1 modifies a data element. Database transaction T2 then reads that data element before T1 performs a COMMIT. If T1 were to perform a ROLLBACK, T2 will have read a value that was never committed and that may thus be considered to have never existed.

P2 (“Non-repeatable Read”): Database transaction T1 reads a data element. Database transaction T2 then modifies or deletes that data element, and performs a COMMIT. If T1 were to attempt to re-read

the data element, it may receive the modified value or discover that the data element has been deleted.

P3 ("Phantom"): Database transaction T1 reads a set of values N that satisfy some <search condition>. Database transaction T2 then executes statements that generate one or more data elements that satisfy the <search condition> used by database transaction T1. If database transaction T1 were to repeat the initial read with the same <search condition>, it obtains a different set of values.

Each database transaction T1 and T2 above must be executed completely or not at all.

The following table defines four isolation levels with respect to the phenomena P0, P1, P2, and P3.

| | Phenomena P0 | Phenomena P1 | Phenomena P2 | Phenomena P3 |
|---|---|---|---|---|
| Level 0 | Not Possible | Possible | Possible | Possible |
| Level 1 | Not Possible | Not Possible | Possible | Possible |
| Level 2 | Not Possible | Not Possible | Not Possible | Possible |
| Level 3 | Not Possible | Not Possible | Not Possible | Not Possible |

**Table 1: Isolation Levels**

The following terms are defined:

$T_1$ = An instance of the ACID Transaction;

$T_2$ = An instance of the ACID Transaction;

$T_3$ = Any of the TPC-H queries 1 to 22 or an instance of the ACID query;

$T_n$ = Any arbitrary transaction.

Although arbitrary, the transaction $T_n$ shall not do dirty writes.

The following table defines the isolation requirements that must be met by TPC-H implementations.

| Req. # | For transactions in this set: | these phenomena: | must NOT be seen by this transaction: | Textual Description: |
|---|---|---|---|---|
| 1. | $\{T_i, T_j\}$ $1 \le i,j \le 2$ | P0, P1, P2, P3 | $T_i$ | Level 3 isolation between any two ACID Transactions. |
| 2. | $\{T_i, T_n\} 1 \le i \le 2$ | P0, P1, P2 | $T_i$ | Level 2 isolation for any ACID Transaction relative to any arbitrary transaction. |
| 3. | $\{T_i, T_3\} 1 \le i \le n$ | P0, P1 | $T_3$ | Level 1 isolation for any of TPC-H queries 1 to 22 relative to any ACID Transaction and any arbitrary transaction. |

**Table 2: Isolation Requirements**

Sufficient conditions must be enabled at either the system or application level to ensure the required isolation defined above is obtained.

However, the required isolation levels must not be obtained by the use of configurations or explicit session-level options that give a particular session or transaction *a priori* exclusive access to the database.

The intent is not to preclude automatic mechanisms such as lock escalation, but to disallow configurations and options that would a priori preclude queries and update transactions against the same database from making progress concurrently.

In addition, the configuration of the database or session-level options must be such that the continuous submission of arbitrary (read-only) queries against one or more tables could not indefinitely delay update transactions affecting those tables from making progress.

### 3.4.2 Isolation Tests

For conventional locking schemes, isolation shall be tested as described below. Systems that implement other isolation schemes may require different validation techniques. It is the responsibility of the test sponsor to disclose those techniques and the tests for them. If isolation schemes other than conventional locking are used, it is permissible to implement these tests differently provided full details are disclosed.

The six tests described here are designed to verify that the system under test is configured to support the required isolation levels, as defined in Clause 3.4.1. All Isolation Tests are performed using a randomly selected set of values (P_KEY, S_KEY, O_KEY, L_KEY, DELTA).

**Comment**: In the isolation tests, the values returned by the ACID Transaction are the old values, as read before the updates.

#### 3.4.2.1 Isolation Test 1

This test demonstrates isolation for the read-write conflict of a read-write transaction and a read-only transaction when the read-write transaction is committed. Perform the following steps:

1. Start an ACID Transaction Txn1 for a randomly selected O_KEY, L_KEY, and DELTA.
2. Suspend Txn1 immediately prior to COMMIT.
3. Start an ACID Query Txn2 for the same O_KEY as in Step 1. (Txn2 attempts to read the data that has just been updated by Txn1.)
4. Verify that Txn2 does not see Txn1's updates.
5. Allow Txn1 to complete.
6. Txn2 should now have completed.

#### 3.4.2.2 Isolation Test 2

This test demonstrates isolation for the read-write conflict of a read-write transaction and a read-only transaction when the read-write transaction is rolled back. Perform the following steps:

1. Start an ACID Transaction Txn1 for a randomly selected O_KEY, L_KEY, and DELTA.
2. Suspend Txn1 immediately prior to COMMIT.
3. Start an ACID Query Txn2 for the same O_KEY as in Step 1. (Txn2 attempts to read the data that has just been updated by Txn1.)
4. Verify that Txn2 does not see Txn1's updates.

5. Force Txn1 to rollback.
6. Txn2 should now have completed.

#### 3.4.2.3 Isolation Test 3

This test demonstrates isolation for the write-write conflict of two update transactions when the first transaction is committed. Perform the following steps:

1. Start an ACID Transaction Txn1 for a randomly selected O_KEY, L_KEY, and DELTA1.
2. Stop Txn1 immediately prior to COMMIT.
3. Start another ACID Transaction Txn2 for the same O_KEY, L_KEY and for a randomly selected DELTA2. (Txn2 attempts to read and update the data that has just been updated by Txn1.)
4. Verify that Txn2 waits.
5. Allow Txn1 to complete. Txn2 should now complete.
6. Verify that
   Txn2.L_EXTENDEDPRICE = Txn1.L_EXTENDEDPRICE+
   (DELTA1 * (Txn1.L_EXTENDEDPRICE / Txn1.L_QUANTITY))

#### 3.4.2.4 Isolation Test 4

This test demonstrates isolation for the write-write conflict of two update transactions when the first transaction is rolled back. Perform the following steps:

1. Start an ACID Transaction Txn1 for a randomly selected O_KEY, L_KEY, and DELTA1.
2. Stop Txn1 immediately prior to COMMIT.
3. Start another ACID Transaction Txn2 for the same O_KEY, L_KEY and for a randomly selected DELTA2. (Txn2 attempts to read and update the data that has just been updated by Txn1.)
4. Verify that Txn2 waits.
5. Force Txn1 to rollback. Txn2 should now complete.
6. Verify that
   Txn2.L_EXTENDEDPRICE = Txn1.L_EXTENDEDPRICE

#### 3.4.2.5 Isolation Test 5

This test demonstrates the ability of read and write transactions affecting different database tables to make progress concurrently.

1. Start an ACID Transaction Txn1 with randomly selected values of O_KEY, L_KEY and DELTA.
2. Suspend Txn1 immediately prior to COMMIT.
3. Start a transaction Txn2 that does the following:
4. Select random values of PS_PARTKEY and PS_SUPPKEY. Return all columns of the PARTSUPP table for which PS_PARTKEY and PS_SUPPKEY are equal to the selected values.
5. Verify that Txn2 completes.
6. Allow Txn1 to complete. Verify that the appropriate rows in the ORDERS, LINEITEM and HISTORY tables have been changed.

#### 3.4.2.6 Isolation Test 6

This test demonstrates that the continuous submission of arbitrary (read-only) queries against one or more tables of the database does not indefinitely delay update transactions affecting those tables from making progress.

1. Start a transaction Txn1. Txn1 executes Q1 (from Clause 2.4) against the qualification database where the substitution parameter [delta] is chosen from the interval [0 .. 2159] so that the query runs for a sufficient length of time.

**Comment**: Choosing [delta] = 0 will maximize the run time of Txn1.

2. Before Txn1 completes, submit an ACID Transaction Txn2 with randomly selected values of O_KEY, L_KEY and DELTA.

If Txn2 completes before Txn1 completes, verify that the appropriate rows in the ORDERS, LINEITEM and HISTORY tables have been changed. In this case, the test is complete with only Steps 1 and 2. If Txn2 will not complete before Txn1 completes, perform Steps 3 and 4:

3. Ensure that Txn1 is still active. Submit a third transaction Txn3, which executes Q1 against the qualification database with a test-sponsor selected value of the substitution parameter [delta] that is not equal to the one used in Step 1.

4. Verify that Txn2 completes before Txn3, and that the appropriate rows in the ORDERS, LINEITEM and HISTORY tables have been changed.

**Comment**: In some implementations Txn2 will not queue behind Txn1. If Txn2 completes prior to Txn1 completion, it is not necessary to run Txn3 in order to demonstrate that updates will be processed in a timely manner as required by Isolation Tests.

## 3.5 Durability Requirements

The SUT must guarantee durability: the ability to preserve the effects of committed transactions and ensure database consistency after recovery from any one of the failures listed in Clause 3.5.3.

**Comment**: No system provides complete durability (i.e., durability under all possible types of failures). The specific set of single failures addressed in Clause 3.5.3 is deemed sufficiently significant to justify demonstration of durability across such failures.

### 3.5.1 Durable Medium Definition

A durable medium is a data storage medium that is either:

a) An inherently non-volatile medium (e.g., magnetic disk, magnetic tape, optical disk, etc.) or;

b) A volatile medium with its own self-contained power supply that will retain and permit the transfer of data, before any data is lost, to an inherently non-volatile medium after the failure of external power.

A configured and priced Uninterruptible Power Supply (UPS) is not considered external power.

**Comment**: A durable medium can fail; this is usually protected against by replication on a second durable medium (e.g., mirroring) or logging to another durable medium. Memory can be considered a durable medium if it can preserve data long enough to satisfy the requirement (b) above, for example, if it is accompanied by an Uninterruptible Power Supply, and the contents of memory can be transferred to an inherently non-volatile medium during the failure. Note that no distinction is made between main memory and memory performing similar permanent or temporary data storage in other parts of the system (e.g., disk controller caches).

### 3.5.2 Committed Property Definition

3.5.2.1 A transaction is considered committed when the transaction manager component of the system has either written the log or written the data for the committed updates associated with the transaction to a durable medium.

**Comment 1**: Transactions can be committed without the user subsequently receiving notification of that fact, since message integrity is not required for TPC-H.

**Comment 2:** Although the order of operations in the ACID Transaction is immaterial, the actual return of data cannot begin until the commit operation has successfully completed.

3.5.2.2 To facilitate the execution of the durability tests the driver must maintain a durable success file that records the details of each transaction which has successfully completed and whose message has been returned to the driver. At the time of an induced failure this success file must contain a record of all transactions which have been committed, except for transactions whose commit notification message to the driver was interrupted by the failure.

The durability success file is required only for the durability tests and must contain the following fields:

| Fields | Datatype Definition |
|---|---|
| P_KEY | identifierForeign reference to P_PARTKEY |
| S_KEY | identifierForeign reference to S_SUPPKEY |
| O_KEY | identifierForeign reference to O_ORDERKEY |
| L_KEY | integer |
| DELTA | integer |
| DATE_T | date and time to second |

**Comment**: If the driver resides on the SUT, the success file must be isolated from the TPC-H database. For example, the success file must be written outside of the ACID Transaction, and if the durability of the success file is provided by the same data manager as the TPC-H database, it must use a different log file.

### 3.5.3 Durability Across Single Failures

The test sponsor is required to guarantee that the test system will preserve the database and the effects of committed updates after recovery from any of the failures listed below:

- Permanent irrecoverable failure of any single durable medium containing TPC-H database tables or recovery log data. The media to be failed is to be chosen at random by the auditor, and cannot be specially prepared.

**Comment**: If main memory is used as a durable medium, then it must be considered as a potential single point of failure. Sample mechanisms to survive single durable medium failures are database archiving in conjunction with a redo (after image) log, and mirrored durable media. If memory is the durable medium and mirroring is the mechanism used to ensure durability, then the mirrored memories must be independently powered.

- Instantaneous interruption (system crash/system hang) in processing which requires system re-boot to recover.

**Comment**: This implies abnormal system shutdown, which requires loading of a fresh copy of the operating system from the boot device. It does not necessarily imply loss of volatile memory. When the recovery mechanism relies on the pre-failure contents of volatile memory, the means used to avoid the loss of volatile memory (e.g., an Uninterruptible Power Supply) must be included in the system cost calculation. A sample mechanism to survive an instantaneous interruption in processing is an undo/redo log.

- Failure of all or part of memory (loss of contents).

**Comment**: This implies that all or part of memory has failed. This may be caused by a loss of external power or the permanent failure of a memory board.

- SUT Power Failure: Loss of all external power to the SUT for an indefinite time period.

**Comment:** To demonstrate durability in a cluster during a power failure, the largest subset of the SUT maintained by a single UPS must be failed. For example, if a system has one UPS per node or set of nodes, it is sufficient to fail one node or that set of nodes. If there is only one UPS for the entire system, then the entire system must be failed. In either case, all UPSs must be priced.

Regardless of UPS configuration, at least one node of each subset of the nodes in the cluster providing a distinct function must be failed.

### 3.5.4 Durability Tests

The intent of these tests is to demonstrate that all transactions whose output messages have been received by the driver have in fact been committed in spite of any single failure from the list in Clause 3.5.3 and that all consistency conditions are still met after the database is recovered.

For each of the failure types defined in Clause 3.5.3 perform the following steps:

1. Verify that the ORDERS, and LINEITEM tables are initially consistent as defined in Clause 3.3.2.1, based on a random sample of at least 10 distinct values of O_ORDERKEY.
2. Asynchronously submit a minimum of 200 ACID Transactions from each of at least the number of the execution streams (# query streams + 1 refresh stream) used in the reported throughput test. Each transaction must use values of (O_KEY, L_KEY, DELTA) randomly generated within the ranges defined in Clause 3.1.6.2. Ensure that all the values of O_ORDERKEY chosen in Step 1 are used by some transaction in Step 2. It must be demonstrated that transactions are in progress at the time of the failure.
3. Wait until at least 100 of the ACID transactions from each stream submitted in Step 2 have completed. Cause the failure selected from the list in Clause 3.5.3. At the time of the failure, it must be demonstrated that:

- At least one transaction is in flight.
- No stream has completed submission of the required minimum number of ACID transactions as defined in Step 2.

**Comment**: The intent is that the failure is induced while all streams are continuously submitting and executing transactions. If the number of in-flight transactions at the point of failure is less than the number of streams, this is assumed to be a random consequence of interrupting some streams during the very small interval between committing one transaction and submitting the next.

4. Restart the system under test using normal recovery procedures.
5. Compare the contents of the durability success file and the HISTORY table to verify that records in the success file for a committed ACID Transaction have a corresponding record in the HISTORY table and that no success record exists for uncommitted transactions. Count the number of entries in the success file and in the HISTORY table and report any difference.

**Comment**: This difference can only be due to transactions that were committed on the system under test, but for which the data was not written in the success file before the failure.

6. Re-verify the consistency of the ORDERS, and LINEITEM tables as defined in Clause 3.3.2.1.

# 4: SCALING AND DATABASE POPULATION

## 4.1 Database Definition and Scaling

### 4.1.1 Test Database

4.1.1.1 The test database is the database used to execute the load test and the performance test (see Clause 5.1.1.4).

4.1.1.2 The test database must be scaled as defined in Clause 4.1.3.

4.1.1.3 The test database must be populated according to Clause 4.2.

### 4.1.2 Qualification Database

4.1.2.1 A qualification database must be created and populated for use in the query validation test described in Clause 2.3. The intent is that the functionality exercised by running the validation queries against the qualification database be the same as that exercised against the test database during the performance test. To this end, the qualification database must be identical to the test database in virtually every regard except size, including but not limited to:

- Column definitions;
- Method of data generation and loading;
- Statistics gathering method;
- ACID property implementation;
- Type of partitioning (but not degree of partitioning);
- Replication
- Table type (if there is a choice);
- Auxiliary data structures (e.g., indices).

The qualification database may differ from the test database only if the difference is directly related to the difference in sizes. For example, if the test database employs horizontal partitioning (see Clause 1.5.3), then the qualification database must also employ horizontal partitioning, though the number of partitions may differ in each case. As another example, the qualification database could be configured such that it uses a representative sub-set of the processors/cores/threads, memory and disks used by the test database configuration. If the qualification database configuration differs from the test database configuration in any way, the differences must be disclosed (see Clause 8.3.5.8).

4.1.2.2 The population of the qualification database must be exactly equal to a scale factor, SF, of 1 (see Clause 4.1.3 for a definition of SF).

### 4.1.3 Database Scaling Requirements

4.1.3.1 Scale factors used for the test database must be chosen from the set of fixed scale factors defined as follows:

1, 10, 30, 100, 300, 1000, 3000, 10000, 30000, 100000

The database size is defined with reference to scale factor 1 (i.e., SF = 1; approximately 1GB as per Clause 4.2.5), the minimum required size for a test database. Therefore, the following series of database sizes corresponds to the series of scale factors and must be used in the metric names QphH@Size and Price-per-QphH@Size (see Clause 5.4), as well as in the executive summary statement (see Appendix E):

1GB, 10GB, 30GB, 100GB, 300GB, 1000GB, 3000GB, 10000GB, 30000GB, 100000GB

Where GB stands for gigabyte, defined to be $2^{30}$ bytes.

**Comment 1**: Although the minimum size of the test database for a valid performance test is 1GB (i.e., SF = 1), a test database of 3GB (i.e., SF = 3) is not permitted. This requirement is intended to encourage comparability of results at the low end and to ensure a substantial actual difference in test database sizes.

**Comment 2**: The maximum size of the test database for a valid performance test is currently set at 100000 (i.e., SF = 100,000). The TPC recognizes that additional benchmark development work is necessary to allow TPC-H to scale beyond that limit.

4.1.3.2 Test sponsors must choose the database size they want to execute against by selecting a size and corresponding scale factor from the defined series.

4.1.3.3 The ratio of total data storage to database size must be computed by dividing the total durable data storage of the priced configuration (expressed in GB) by the size chosen for the test database as defined in Scale factors used for the test database must be chosen from the set of fixed scale factors defined as follows:. The ratio must be reported to the nearest 1/100th, rounded up. The ratio must be included in both the Full Disclosure Report and the Executive Summary.

## 4.2 DBGEN and Database Population

### 4.2.1 The DBGEN Program

4.2.1.1 The test database and the qualification database must be populated with data that meets the requirements of Clause 4.2.2 and Clause 4.2.3. The **DBGEN** source code provided in Appendix D is a sample implementation of a database population generator. It has been written in ANSI 'C' and has been ported to a large number of platforms. If DBGEN is used, its version must match the version of the benchmark specification.

**Comment**: Use of DBGEN is strongly recommended. Exact query answer set compliance is required and this may not be possible unless data is identical to DBGEN's output. This, in turn, may not be possible without duplicating the exact sequence of random numbers generated by DBGEN.

4.2.1.2 The data generated by DBGEN are meant to be compliant with Clause 4.2.2 and Clause 4.2.3. In case of differences between the content of these two Clauses and the data generated by DBGEN, the latter prevails.

4.2.1.3 If a program other than DBGEN is used to populate the database, the resultant data must meet the following requirements in order to be considered correct:

- The content of individual columns must match that produced by DBGEN;
- The data format of individual columns must be identical to that produced by DBGEN;
- The number of rows generated for a given scale factor must match that produced by DBGEN.

**Comment 1**: The intent of this clause is to allow for modification of the DBGEN code required for portability or speed, while precluding any change that effects the resulting data.

**Comment 2**: If a program other than DBGEN is used to populate the database, its complete source code must be included in the full disclosure report. Alternatively, if a modified version of DBGEN is used, the modified source files must be disclosed in full.

### 4.2.2 Definition Of Terms

4.2.2.1 The term **random** means independently selected and uniformly distributed over the specified range of values.

4.2.2.2 The term **unique within [x]** represents any one value within a set of x values between 1 and x, unique within the scope of rows being populated.

4.2.2.3 The notation **random value [x .. y]** represents a random value between x and y inclusively, with a mean of (x+y)/2, and with the same number of digits of precision as shown. For example, [0.01 .. 100.00] has 10,000 unique values, whereas [1..100] has only 100 unique values.

4.2.2.4 The notation **random string [list_name]** represents a string selected at random within the list of strings list_name as defined in Clause 4.2.2.12. Each string must be selected with equal probability.

4.2.2.5 The notation **text appended with digit [text, x]** represents a string generated by concatenating the sub-string text, the character "# ", and the sub-string representation of the number x.

4.2.2.6 The notation **random a-string [x]** represents a string of length x comprised of randomly generated alphanumeric characters within a character set of at least 64 symbols.

4.2.2.7 The notation **random v-string [x]** represents a string comprised of randomly generated alphanumeric characters within a character set of at least 64 symbols. The length of the string is a random value [0.4 x .. 1.6 x] rounded up to the next integer.

4.2.2.8 The term **date** represents a string of numeric characters separated by hyphens and comprised of a 4 digit year, 2 digit month and 2 digit day of the month.

4.2.2.9 The term **phone number** represents a string of numeric characters separated by hyphens and generated as follows:

Let i be an index into the list of strings Nations (i.e., ALGERIA is 0, ARGENTINA is 1, etc., see Clause 4.2.3),
Let country_code be the sub-string representation of the number (i + 10),
Let local_number1 be random [100 .. 999],
Let local_number2 be random [100 .. 999],
Let local_number3 be random [1000 .. 9999],

The phone number string is obtained by concatenating the following sub-strings:

country_code, "-", local_number1, "-", local_number2, "-", local_number3

4.2.2.10 The term **text string[x]** represents a string generated by the pseudo text grammar defined in Clause 4.2.2.13. The length of the string is a random value [0.4 x .. 1.6 x] rounded up to the next integer. The output of the pseudo text grammar is truncated at the selected length.

4.2.2.11 All dates must be computed using the following values:

STARTDATE = 1992-01-01 CURRENTDATE = 1995-06-17 ENDDATE = 1998-12-31

4.2.2.12 The following list of strings must be used to populate the database:

List name:**Types**

Each string is generated by the concatenation of a variable length syllable selected at random from each of the three following

lists and separated by a single space (for a total of 150 combinations).

| Syllable 1 | Syllable 2 | Syllable 3 |
|---|---|---|
| STANDARD | ANODIZED | TIN |
| SMALL | BURNISHED | NICKEL |
| MEDIUM | PLATED | BRASS |
| LARGE | POLISHED | STEEL |
| ECONOMY | BRUSHED | COPPER |
| PROMO | | |

List name:**Containers**

Each string is generated by the concatenation of a variable length syllable selected at random from each of the two following lists and separated by a single space (for a total of 40 combinations).

| Syllable 1 | Syllable 2 |
|---|---|
| SM | CASE |
| LG | BOX |
| MED | BAG |
| JUMBO | JAR |
| WRAP | PKG |
| | PACK |
| | CAN |
| | DRUM |

List name:**Segments**

| | | | |
|---|---|---|---|
| AUTOMOBILE | BUILDING | FURNITURE | MACHINERY |
| HOUSEHOLD | | | |

List name:**Priorities**

| | | | |
|---|---|---|---|
| 1-URGENT | 2-HIGH | 3-MEDIUM | 4-NOT SPECIFIED |
| 5-LOW | | | |

List name:**Instructions**

| | | | |
|---|---|---|---|
| DELIVER IN PERSON | COLLECT COD | NONE | TAKE BACK RETURN |

List name:**Modes**

| | | | |
|---|---|---|---|
| REG AIR | AIR | RAIL | SHIP |
| TRUCK | MAIL | FOB | |

List name:**Nouns**

| | | | |
|---|---|---|---|
| foxes | ideas | theodolites | pinto beans |
| instructions | dependencies | excuses | platelets |
| asymptotes | courts | dolphins | multipliers |
| sauternes | warthogs | frets | dinos |
| attainments | somas | Tiresias' | patterns |
| forges | braids | hockey players | frays |
| warhorses | dugouts | notornis | epitaphs |
| pearls | tithes | waters | orbits |
| gifts | sheaves | depths | sentiments |
| decoys | realms | pains | grouches |
| escapades | | | |

List name:**Verbs**

| | | | |
|---|---|---|---|
| sleep | wake | are | cajole |
| haggle | nag | use | boost |
| affix | detect | integrate | maintain |
| nod | was | lose | sublate |
| solve | thrash | promise | engage |
| hinder | print | x-ray | breach |
| eat | grow | impress | mold |
| poach | serve | run | dazzle |
| snooze | doze | unwind | kindle |
| play | hang | believe | doubt |

List name:**Adjectives**

| | | | |
|---|---|---|---|
| furious | sly | careful | blithe |
| quick | fluffy | slow | quiet |
| ruthless | thin | close | dogged |
| daring | brave | stealthy | permanent |
| enticing | idle | busy | regular |
| final | ironic | even | bold |
| silent | | | |

List name:**Adverbs**

| | | | |
|---|---|---|---|
| sometimes | always | never | furiously |
| slyly | carefully | blithely | quickly |
| fluffily | slowly | quietly | ruthlessly |
| thinly | closely | doggedly | daringly |
| bravely | stealthily | permanently | enticingly |
| idly | busily | regularly | finally |
| ironically | evenly | boldly | silently |

List name:**Prepositions**

| | | | |
|---|---|---|---|
| about | above | according to | across |
| after | against | along | alongside of |
| among | around | at | atop |
| before | behind | beneath | beside |
| besides | between | beyond | by |
| despite | during | except | for |
| from | in place of | inside | instead of |
| into | near | of | on |
| outside | over | past | since |
| through | throughout | to | toward |
| under | until | up | upon |
| without | with | within | |

List name:**Auxiliaries**

| | | | |
|---|---|---|---|
| do | may | might | shall |
| will | would | can | could |
| should | ought to | must | will have to |
| shall have to | could have to | should have to | must have to |
| need to | try to | | |

List name:**Terminators**

| | | | |
|---|---|---|---|
| . | ; | : | ? |
| ! | -- | | |

4.2.2.13 Pseudo text used in the data population (see Clause 4.2.2.10) must conform to the following grammar:

text:<sentence>
|<text> <sentence>
;

sentence:<noun phrase> <verb phrase> <terminator>
|<noun phrase> <verb phrase> <prepositional phrase> <terminator>
|<noun phrase> <verb phrase> <noun phrase> <terminator>
|<noun phrase> <prepositional phrase> <verb phrase>

<noun phrase> <terminator>
|<noun phrase> <prepositional phrase> <verb phrase>
<prepositional phrase> <terminator>
;

noun phrase:<noun>
|<adjective> <noun>
|<adjective>, <adjective> <noun>
|<adverb> <adjective> <noun>
;

verb phrase:<verb>
|<auxiliary> <verb>
|<verb> <adverb>
|<auxiliary> <verb> <adverb>
;

prepositional phrase: <preposition> the <noun phrase>
;

noun:selected from Nouns (as defined in Clause 4.2.2.12)

verb: selected from Verbs (as defined in Clause 4.2.2.12)

adjective: selected from Adjectives (as defined in Clause 4.2.2.12)

adverb: selected from Adverbs (as defined in Clause 4.2.2.12)

preposition: selected from Prepositions (as defined in Clause 4.2.2.12)

terminator: selected from Terminators (as defined in Clause 4.2.2.12)

auxiliary: selected from Auxiliary (as defined in Clause 4.2.2.12)

4.2.2.14 The grammar defined in Clause 4.2.2.13 relies on the weighted, non-uniform distribution of its constituent distributions (nouns, verbs, auxiliaries, etc.). A sponsor wishing to use a means of data generation other than DBGEN must assure that the distributions defined in the DBGEN source code are reproduced exactly.

### 4.2.3 Test Database Data Generation

The data generated by DBGEN (see Clause 4.2.1) must be used to populate the database as follows (where SF is the scale factor, see Clause 4.1.3.1):

- SF * 10,000 rows in the SUPPLIER table with:
  S_SUPPKEY unique within [SF * 10,000].
  S_NAME text appended with digit ["Supplier", S_SUPPKEY].
  S_ADDRESS random v-string[25].
  S_NATIONKEY random value [0 .. 24].
  S_PHONE generated according to Clause 4.2.2.9.
  S_ACCTBAL random value [-999.99 .. 9,999.99].
  S_COMMENT text string [63].
  SF * 5 rows are randomly selected to hold at a random position a string matching "Customer%Complaints". Another SF * 5 rows are randomly selected to hold at a random position a string matching "Customer%Recommends", where % is a wildcard that denotes zero or more characters.
- SF * 200,000 rows in the PART table with:

P_PARTKEY unique within [SF * 200,000].
P_NAME generated by concatenating five unique randomly selected strings from the following list, separated by a single space:

{"almond", "antique", "aquamarine", "azure", "beige", "bisque", "black", "blanched", "blue", "blush", "brown", "burlywood", "burnished", "chartreuse", "chiffon", "chocolate", "coral", "cornflower", "cornsilk", "cream", "cyan", "dark", "deep", "dim", "dodger", "drab", "firebrick", "floral", "forest", "frosted", "gainsboro", "ghost", "goldenrod", "green", "grey", "honeydew", "hot", "indian", "ivory", "khaki", "lace", "lavender", "lawn", "lemon", "light", "lime", "linen", "magenta", "maroon", "medium", "metallic", "midnight", "mint", "misty", "moccasin", "navajo", "navy", "olive", "orange", "orchid", "pale", "papaya", "peach", "peru", "pink", "plum", "powder", "puff", "purple", "red", "rose", "rosy", "royal", "saddle", "salmon", "sandy", "seashell", "sienna", "sky", "slate", "smoke", "snow", "spring", "steel", "tan", "thistle", "tomato", "turquoise", "violet", "wheat", "white", "yellow"}.

P_MFGR text appended with digit ["Manufacturer",M], where M = random value [1,5].
P_BRAND text appended with digit ["Brand",MN], where N = random value [1,5] and M is defined while generating P_MFGR.
P_TYPE random string [Types].
P_SIZE random value [1 .. 50].
P_CONTAINER random string [Containers].
P_RETAILPRICE =
(90000 + ((P_PARTKEY/10) modulo 20001 ) + 100 * (P_PARTKEY modulo 1000))/100
P_COMMENT text string [14].
For each row in the PART table, four rows in PARTSUPP table with:
PS_PARTKEY = P_PARTKEY.
PS_SUPPKEY = (PS_PARTKEY + (i * (( S/4 ) + (int)(PS_PARTKEY-1 )/S)))) modulo S + 1 where i is the $i^{th}$ supplier within [0 .. 3] and S = SF * 10,000.
PS_AVAILQTY random value [1 .. 9,999].
PS_SUPPLYCOST random value [1.00 .. 1,000.00].
PS_COMMENT text string [124].

- SF * 150,000 rows in CUSTOMER table with:
  C_CUSTKEY unique within [SF * 150,000].
  C_NAME text appended with digit ["Customer", C_CUSTKEY].
  C_ADDRESS random v-string [25].
  C_NATIONKEY random value [0 .. 24].
  C_PHONE generated according to Clause 4.2.2.9.
  C_ACCTBAL random value [-999.99 .. 9,999.99].
  C_MKTSEGMENT random string [Segments].
  C_COMMENT text string [73].

- For each row in the CUSTOMER table, ten rows in the ORDERS table with:
  O_ORDERKEY unique within [SF * 1,500,000 * 4].
  **Comment**: The ORDERS and LINEITEM tables are sparsely populated by generating a key value that causes the first 8 keys of each 32 to be populated, yielding a 25% use of the key range. Test sponsors must not take advantage of this aspect of the benchmark. For example, horizontally partitioning the test database onto different devices in order to place unused areas onto separate peripherals is prohibited.

  O_CUSTKEY = random value [1 .. (SF * 150,000)].
  The generation of this random value must be such that O_CUSTKEY modulo 3 is not zero.
  **Comment:** Orders are not present for all customers. Every third customer (in C_CUSTKEY order) is not assigned any order.

  O_ORDERSTATUS set to the following value:
  "F" if all lineitems of this order have L_LINESTATUS set to "F".
  "O" if all lineitems of this order have L_LINESTATUS set to "O".
  "P" otherwise.

O_TOTALPRICE computed as:
sum (L_EXTENDEDPRICE * (1+L_TAX) * (1-L_DISCOUNT)) for all LINEITEM of this order.
O_ORDERDATE uniformly distributed between STARTDATE and (ENDDATE - 151 days).
O_ORDERPRIORITY random string [Priorities].
O_CLERK text appended with digit ["Clerk", C] where C = random value [000000001 .. (SF * 1000)].
O_SHIPPRIORITY set to 0.
O_COMMENT text string [49].

- For each row in the ORDERS table, a random number of rows within [1 .. 7] in the LINEITEM table with:
L_ORDERKEY = O_ORDERKEY.
L_PARTKEY random value [1 .. (SF * 200,000)].
L_SUPPKEY = (L_PARTKEY + (i * (( S/4 ) + (int)(L_PARTKEY-1 )/S))) modulo S + 1
where i is the corresponding supplier within [0 .. 3] and S = SF * 10,000.
L_LINENUMBER unique within [7].
L_QUANTITY random value [1 .. 50].
L_EXTENDEDPRICE = l_quantity * p_retailprice
where p_retailprice is from the part with P_PARTKEY = L_PARTKEY.
L_DISCOUNT random value [0.00 .. 0.10].
L_TAX random value [0.00 .. 0.08].
L_RETURNFLAG set to a value selected as follows:
If L_RECEIPTDATE <= CURRENTDATE
then either "R" or "A" is selected at random
else "N" is selected.
L_LINESTATUS set the following value:
"O" if L_SHIPDATE > CURRENTDATE
"F" otherwise.
L_SHIPDATE = O_ORDERDATE + random value [1 .. 121].
L_COMMITDATE = O_ORDERDATE + random value [30 .. 90].
L_RECEIPTDATE = L_SHIPDATE + random value [1 .. 30].
L_SHIPINSTRUCT random string [Instructions].
L_SHIPMODE random string [Modes].
L_COMMENT text string [27].

- 25 rows in the NATION table with:
N_NATIONKEY unique value between 0 and 24.
N_NAME string from the following series of (N_NATIONKEY, N_NAME, N_REGIONKEY).
(0, ALGERIA, 0);(1, ARGENTINA, 1);(2, BRAZIL, 1);
(3, CANADA, 1);(4, EGYPT, 4);(5, ETHIOPIA, 0);
(6, FRANCE, 3);(7, GERMANY, 3);(8, INDIA, 2);
(9, INDONESIA, 2);(10, IRAN, 4);(11, IRAQ, 4);
(12, JAPAN, 2);(13, JORDAN, 4);(14, KENYA, 0);
(15, MOROCCO, 0);(16, MOZAMBIQUE, 0);(17, PERU, 1);
(18, CHINA, 2);(19, ROMANIA, 3);(20, SAUDI ARABIA, 4);
(21, VIETNAM, 2);(22, RUSSIA, 3);(23, UNITED KINGDOM, 3);
(24, UNITED STATES, 1)
N_REGIONKEY is taken from the series above.
N_COMMENT text string [95].

- 5 rows in the REGION table with:
R_REGIONKEY unique value between 0 and 4.
R_NAME string from the following series of (R_REGIONKEY, R_NAME).
(0, AFRICA);(1, AMERICA); (2, ASIA);
(3, EUROPE);(4, MIDDLE EAST)
R_COMMENT text string [95].

### 4.2.4 Refresh Function Data Generation

4.2.4.1 The test database is initially populated with 75% sparse primary keys for the ORDERS and LINEITEM tables (see Clause 4.2.3) where only the first eight key values of each group of 32 keys are used. Subsequently, the refresh function RF1 uses the 'holes' in the key ranges for inserting new rows.

4.2.4.2 DBGEN generates refresh data sets for the refresh functions such that:

- For the first through the 1,000th execution of RF1 data sets are generated for inserting 0.1% new rows with a primary key within the second 8 key values of each group of 32 keys;
- For the first through the 1,000th execution of RF2 data sets are generated for deleting 0.1% existing rows with a primary key within the original first 8 key values of each group of 32 keys.

**Comment**: As a result, after 1,000 executions of RF1/RF2 pairs the test database is still populated with 75% sparse primary keys, but the second 8 key values of each group of 32 keys are now used.

4.2.4.3 The refresh function data set generation scheme can be repeated until 4000 RF1/RF2 pairs have been executed, at which point the population of the test database is once again in its initial state.

### 4.2.5 Database Size

4.2.5.1 Table 3: Estimated Database Size shows the test database size for a scale factor, SF, of 1.

**Table 3: Estimated Database Size**

| Table Name | Cardinality (in rows) | Length (in bytes) of Typical[2] Row | Typical[2] Table Size (in 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[3] | 6,001,215 | 112 | 641 |
| NATION[1] | 25 | 128 | < 1 |
| REGION[1] | 5 | 124 | < 1 |
| Total | 8,661,245 | | 956 |

1 Fixed cardinality: does not scale with SF.

2 Typical lengths and sizes given here are examples, not requirements, of what could result from an implementation (sizes do not include storage/access overheads).

3 The cardinality of the LINEITEM table is not a strict multiple of SF since the number of lineitems in an order

is chosen at random with an average of four (see Clause 4.2.5.2).

**Note**: 1 MB is defined to be $2^{20}$ bytes.

Data types are sized as follows: 4-byte integers, 8-byte decimals, 4-byte dates.

4.2.5.2 Table 4: LINEITEM Cardinality shows the cardinality of the LINEITEM table at all authorized scale factors.

**Table 4: LINEITEM Cardinality**

| Scale Factor (SF) | Cardinality of LINEITEM Table |
|---|---|
| 1 | 6001215 |
| 10 | 59986052 |
| 30 | 179998372 |
| 100 | 600037902 |
| 300 | 1799989091 |
| 1000 | 5999989709 |
| 3000 | 18000048306 |
| 10000 | 59999994267 |
| 30000 | 179999978268 |
| 100000 | 599999969200 |

## 4.3 Database Load Time

4.3.1 The process of building the test database is known as database load. Database load consists of timed and untimed components. However, all components must be fully disclosed (see Clause 8.3.5.7).

4.3.2 The total elapsed time to prepare the test database for the execution of the performance test is called the database load time, and must be reported. This includes all of the elapsed time to create the tables defined in Clause 1.4, load data, create indices, define and validate constraints, gather statistics for the test database, configure the system under test as it will be during the performance test, and ensure that the test database meets the ACID requirements including syncing loaded data on RAID devices and the taking of a backup of the database, when necessary.

4.3.3 The population of the test database, as defined in Clause 4.1, consists of two logical phases:

1. **Generation**: the process of using DBGEN or a similar program to create data in a format suitable for presentation to the DBMS load facility. The generated data may be stored in memory, or in flat files on tape or disk.
2. **Loading**: the process of storing the generated data into the database tables.

Generation and loading of the data can be accomplished in two ways:

1. DBGEN (or a similar program) is used to generate flat files stored on disk or tape. The records in these files may optionally be permuted and relocated to the SUT. After table creation on the SUT, data is loaded from the flat files into the database. In this case, called **load from flat files**, only the loading phase contributes to the database load time.

2. DBGEN (or a similar program) is used to generate data that is directly loaded into the database tables using an "in-line" load facility. In this case, called **in-line load**, generation and loading occur concurrently and both contribute to the database load time.

4.3.4 The database load time must be measured on the system under test (SUT).

4.3.5 The timing of the database load time begins with the creation of the tables defined in Clause 1.4.

4.3.6 There are five classes of operations which may be excluded from database load time:

- Any operation that does not affect the state of the DBMS (e.g., data generation into flat files, relocation of flat files to the SUT, permutation of data in flat files, operating-system-level disk partitioning or configuration);
- Any modification to the state of the DBMS that is not specific to the TPC-H workload (e.g. logical tablespace creation or database block formatting);
- The time required to install or remove physical resources (e.g. processors/cores/threads, memory or disk drives) on the SUT that are not priced (see Clause 4.3.9);
- An optional backup of the test database performed at the test sponsor's discretion. However, if a backup is required to ensure that the ACID properties can be met it must be included in the load time;
- Operations that create RAID devices.

**Comment**: The time required to perform any necessary software reconfiguration (such as DBMS or operating system parameters) must be included in the database load time.

4.3.7 The timing of the database load ends when the database is fully populated and the SUT is configured as it will be during the performance test.

**Comment**: The intent of this Clause is that when the timing ends the system under test be capable of executing the performance test without any further change. The database load may be decomposed into several phases. Database load time is the sum of the elapsed times of all phases during which activity other than that detailed in Clause 4.3.6 occurred on the SUT. The timing of a load phase completes only when any change to the test database has been written to durable media (see Clause 3.5.1).

**Comment 2:** Since the time of the end of the database load is used to seed the random number generator for the substitution parameter, that time cannot be delayed in any way that would make it predictable to the test sponsor.

4.3.8 The resources used to generate, permute, relocate to the SUT or hold DBGEN data may optionally be distinct from those used to run the actual benchmark. For example:

- For load from flat files, a separate system or a distinct storage subsystem may be used to generate, store and permute DBGEN data into the flat files used for the database load.
- For in-line load, separate and distinct processing elements may be used to generate and permute data and to deliver it to the DBMS.

4.3.9 Resources used only in the generation phase of the population of the test database must be treated as follows:

For load from flat files,

- Any processing element (e.g., processor/core/thread or memory) used exclusively to generate and hold DBGEN data or relocate it to the SUT prior to the load phase shall not be included in the total priced system (see Clause 7.1) and must be physically removed from or made inaccessible to the SUT prior to the start of the load phase;

- Any storage facility (e.g., disk drive, tape drive or peripheral controller) used exclusively to generate and deliver data to the SUT during the load phase shall not be included in the total priced system. The test sponsor must demonstrate to the satisfaction of the auditor that this facility is not being used in the performance tests.

For in-line load,

- Any processing element (e.g., processor/core/thread or memory) or storage facility (e.g., disk drive, tape drive or peripheral controller) used exclusively to generate and deliver DBGEN data to the SUT during the load phase shall not be included in the total priced system and must be physically removed from or made inaccessible to the SUT prior to the start of the measurement tests.

**Comment**: The intent is to isolate the cost of resources required to generate data from those required to load data into the database tables.

4.3.10 An implementation may require additional programs to transfer DBGEN data into the database tables (from either flat file or in-line load). If non-commercial programs are used for this purpose, their source code must be disclosed. If commercially available programs are used for this purpose, their invocation and configuration must be disclosed. Whether or not the software is commercially available, use of the software's functionality's must be limited to:

1. Permutation of the data generated by DBGEN;
2. Delivery of the data generated by DBGEN to the DBMS.

4.3.11 The database load must be implemented using commercially available utilities (invoked at the command level or through an API) or an SQL programming interface (such as embedded SQL or ODBC).

# 5: PERFORMANCE METRICS AND EXECUTION RULES

## 5.1 Definition of Terms

### 5.1.1 Components of the Benchmark

5.1.1.1 The **benchmark** is defined as the execution of the load test followed by the performance test.

5.1.1.2 The **load test** begins with the creation of the database tables and includes all activity required to bring the system under test to the configuration that immediately precedes the beginning of the performance test (see Clause 4.1.3). The load test may not include the execution of any of the queries in the performance test (see Clause 5.1.2) or any similar query.

5.1.1.3 The **performance test** consists of two runs.

5.1.1.4 A **run** consists of one execution of the Power test described in Clause 5.3.3 followed by one execution of the Throughput test described in Clause 5.3.4.

5.1.1.5 Run 1 is the first run following the load test (see Clause 5.3.1.4). Run 2 is the run following Run 1.

5.1.1.6 A failed run is defined as a run that did not complete successfully due to unforeseen system failures.

### 5.1.2 Components of a Run

5.1.2.1 A **query** is defined as any one of the 22 TPC-H queries specified in Clause 2.

- The symbol "$Q_i$ ", with i in lowercase and from 1 to 22, represents a given query.

5.1.2.2 A **query set** is defined as the sequential execution of each and every one of the queries.

5.1.2.3 A **query stream** is defined as the sequential execution of a single query set submitted by a single emulated user.

- The symbol "S", in uppercase, is used to represent the number of query streams used during the throughput test;
- The symbol "s", in lowercase and from 1 to S, is used to represent a given query stream.

5.1.2.4 A **refresh stream** is defined as the sequential execution of an integral number of pairs of refresh functions submitted from within a batch program.

5.1.2.5 A **pair of refresh functions** is defined as one of each of the two TPC-H refresh functions specified in Clause 2.

- The symbol "$RF_j$ ", with j in lowercase and from 1 to 2, represents a given refresh function.

5.1.2.6 A **session** is defined as the process context capable of supporting the execution of either a query stream or a refresh stream.

## 5.2 Configuration Rules

5.2.1 The mechanism used to submit queries and refresh functions to the system under test (SUT) and measure their execution time is called a driver. The driver is a logical entity that can be implemented using one or more physical pro-

grams, processes, or systems (see Clause 6.3).

5.2.2 The communication between the driver and the SUT must be limited to one session per query stream or per refresh stream. These sessions are prohibited from communicating with one another except for the purpose of scheduling refresh functions (see Clause 5.3.7.8).

5.2.3 All sessions supporting the execution of a query stream must be initialized in exactly the same way. The initialization of the session supporting the execution of the refresh stream may be different than that of the query streams. All session initialization parameters, settings and commands must be disclosed.

**Comment 1**: The attributes of the session used in the query stream(s) (see Clause 5.1.2.3) must be the same as the attributes of the session used by the ACID Query (see Clause 3.1.6.3). Similarly, the attributes of the session used in the refresh stream (see Clause 5.1.2.4) must be the same as the attributes of the session used by the ACID Transaction (see Clause 3.1.6.3)

**Comment 2**: The intent of this Clause is to provide the information needed to precisely recreate the execution environment of any given stream prior to the submission of the first query or refresh function.

5.2.4 The driver submits each TPC-H query for execution by the SUT via the session associated with the corresponding query stream.

5.2.5 In the case of the two refresh functions (RF1 and RF2), the driver is only required to submit the commands necessary to cause the execution of each refresh function.

5.2.6 The driver's submittal to the SUT of the queries in the performance test (see Clause 5.1.2.1) is constrained by the following restrictions:

- It must comply with the query compliance requirements of Clause 2.2;
- No part of the interaction between the driver and the SUT can have the purpose of indicating to the DBMS or operating system an execution strategy or priority that is time dependent or query specific;

**Comment**: Automatic priority adjustment performed by the operating system is not prohibited, but specifying a varying priority to the operating system on a query by query basis is prohibited.

- The settings of the SUT's components, such as DBMS (including tables and tablespaces) and operating system, are not to be modified on a query by query basis. These parameters have to be set once before any query or refresh function is run and left in that setting for the duration of the performance test.

5.2.7 The configuration and initialization of the SUT, the database, or the session, including any relevant parameter, switch or option settings, must be based only on externally documented capabilities of the system that can be reasonably interpreted as useful for an ad-hoc decision support workload. This workload is characterized by:

- Sequential scans of large amounts of data;
- Aggregation of large amounts of data;
- Multi-table joins;
- Possibly extensive sorting.

While the configuration and initialization can reflect the general nature of this expected workload, it shall not take special advantage of the limited functions actually exercised by the benchmark. The queries actually chosen in the benchmark are merely examples of the types of queries that might be used in such an environment, not necessarily the actual user queries. Due to this limit in the number and scope of the queries and test environment, TPC-H has chosen to restrict the use of some database technologies (see Clause 1.5 and Clause 5.2.8). In general, the effect of

the configuration on benchmark performance should be representative of its expected effect on the performance of the class of applications modeled by the benchmark.

Furthermore, the features, switches or parameter settings that comprise the configuration of the operating system, the DBMS or the session must be such that it would be reasonable to expect a database administrator with the following characteristics be able to decide to use them:

- Knowledge of the general characteristics of the workload as defined above;
- Knowledge of the logical and physical database layout;
- Access to operating system and database documentation;
- No knowledge of product internals beyond what is documented externally.

Each feature, switch or parameter setting used in the configuration and initialization of the operating system, the DBMS or the session must meet the following criteria:

- It shall remain in effect without change throughout the performance test;
- It shall not make reference to specific tables, indices or queries for the purpose of providing hints to the query optimizer.

5.2.8 The gathering of statistics is part of the database load (see Clause 4.3) but it also serves as an important configuration vehicle, particularly for the query optimizer. In order to satisfy the requirements of Clause 5.2.7, it is desirable to collect the same quality of statistics for every column of every table. However, in order to reduce processing requirements, it is permissible to segment columns into distinct classes and base the level of statistics collection for a particular column on class membership. Class definitions must rely solely on schema-related attributes of a column and must be applied consistently across all tables. For example:

- membership in an index;
- leading or other position in an index;
- use in a constraint (including a primary or foreign key relationships).

Statistics that operate in sets, such as distribution statistics, should employ a fixed set appropriate to the scale factor used. Knowledge of the cardinality, values or distribution of a non-key column as specified in Clause 4 cannot be used to tailor statistics gathering.

5.2.9 Special rules apply to the use of so-called profile-directed optimization (PDO), in which binary executables are reordered or otherwise optimized to best suit the needs of a particular workload. These rules do not apply to the routine use of PDO by a database vendor in the course of building commercially available and supported database products; such use is not restricted. Rather, the rules apply to the use of PDO by a test sponsor to optimize executables of a database product for a particular workload. Such optimization is permissible if all of the following conditions are satisfied:

1. The use of PDO or similar procedures by the test sponsor must be disclosed.
2. The procedure and any scripts used to perform the optimization must be disclosed.
3. The procedure used by the test sponsor could reasonably be used by a customer on a shipped database executable.
4. The optimized database executables resulting from the application of the procedure must be supported by the database software vendor.
5. The workload used to drive the optimization is as described in Clause 5.2.10.
6. The same set of DBMS executables must be used for all phases of the benchmark.

5.2.10 If profile-directed optimization is used under the circumstances described in Clause 5.2.9, the workload used to drive it must be the (possibly repeated) execution of Queries 1,2,4 and 5 in any order, against a TPC-H database of any desired Scale Factor with default substitution parameters applied.

## 5.3 Execution Rules

### 5.3.1 General Rules

5.3.1.1 The driver must submit queries through one or more sessions on the SUT. Each session corresponds to one, and only one, query stream on the SUT.

5.3.1.2 Parallel activity within the SUT directed toward the execution of a single query (i.e., intra-query parallelism) is not restricted.

5.3.1.3 To measure the performance of a system using the TPC Benchmark™ H, the test sponsor will execute runs composed of:

- A **power test**, to measure the raw query execution power of the system when connected with a single active user. In this test, a single pair of refresh functions are executed exclusively by a separate refresh stream and scheduled before and after the execution of the queries (see Clause 5.3.3);
- A **throughput test**, to measure the ability of the system to process the most queries in the least amount of time. In this test, several pairs of refresh functions are executed exclusively by a separate refresh stream. and scheduled as defined by the test sponsor.
- **Comment**: The throughput test is where test sponsors can demonstrate the performance of their systems against a multi-user workload.

5.3.1.4 The performance test follows the load test. However, any system activity that takes place between the completion of the load test (see Clause 5.1.1.2) and the beginning of the performance test is limited to that which is not likely to improve the results of the subsequent performance test. All such activity must be disclosed (see Clause 8.3.6.1). Examples of acceptable activity include but are not limited to:

- Execution of scripts or queries requested by the auditor;
- Processing or archiving of files or timing data gathered during the load test;
- Configuration of performance monitoring tools;
- Execution of simple queries to verify that the database is correctly loaded;
- Taking database backups (if not needed to meet the ACID requirements);
- Rebooting the SUT or restarting the RDBMS.

5.3.1.5 The power test and the throughput test must both be executed under the same conditions, using the same hardware and software configuration and the same data manager and operating system parameters. All such parameters must be reported.

**Comment**: The intent of this Clause is to require that both tests (i.e., the power and throughput tests) be run in identical conditions except for the number of query streams and the scheduling of the refresh functions within the refresh stream.

5.3.1.6 For each query, at least one atomic transaction must be started and completed.

**Comment**: The intent of this Clause is to specifically prohibit the execution of an entire query stream as a single transaction.

5.3.1.7 Each refresh function must consist of at least one atomic transaction. However, logically consistent portions of the refresh functions may be implemented as separate transactions as defined in Clause 2.26.

**Comment**: This intent of this Clause is to specifically prohibit the execution of multiple refresh functions as a single transaction. The splitting of each refresh function into multiple transactions is permitted to encourage "trickle" updates performed concurrently with one or more query streams in the throughput test.

### 5.3.2 Run Sequencing

The performance test consists of two runs. If Run 1 is a failed run (see Clause 5.1.1.6) the benchmark must be restarted with a new load test. If Run 2 is a failed run, it may be restarted without a reload. The reported performance metric must be for the run with the lower TPC-H Composite Query-Per-Hour Performance Metric. The same set of seed values may be used in the consecutive runs.

The TPC-H metrics reported for a given system must represent a conservative evaluation of the system's level of performance. Therefore, the reported performance metrics must be for the run with the lower Composite Query-per-Hour metric

### 5.3.3 Power Test

5.3.3.1 The power test must be driven by queries submitted by the driver through a single session on the SUT. The session executes queries one after another. This test is used to measure the raw query execution power of the SUT with a single query stream. The power test must be executed in parallel with a single refresh stream (see Clause 5.1.2.4).

5.3.3.2 The power test must follow these steps in order:

1. The refresh function RF1 is executed by the refresh stream.
2. The full query set is executed once by the query stream.
3. The refresh function RF2 is executed by the same refresh stream.

5.3.3.3 The timing intervals (see Clause 5.3.7) for each query and for both refresh functions are collected and reported.

### 5.3.4 Throughput Test

**Table 11: Minimum Required Stream Count**

| SF | S(Streams) |
|---|---|
| 1 | 2 |
| 10 | 3 |
| 30 | 4 |
| 100 | 5 |
| 300 | 6 |
| 1000 | 7 |
| 3000 | 8 |
| 10000 | 9 |
| 30000 | 10 |

**Table 11: Minimum Required Stream Count**

| SF | S(Streams) |
|---|---|
| 100000 | 11 |

5.3.4.1 The throughput test must be driven by queries submitted by the driver through two or more sessions on the SUT. There must be one session per query stream on the SUT and each stream must execute queries serially (i.e., one after another). The value of S, the minimum number of query streams, is given in Table 11. The throughput test must be executed in parallel with a single refresh stream (see Clause 5.1.2.4).

The throughput test must immediately follow one, and only one, power test. No changes to the configuration of the SUT can be made between the power test and the throughput test (see 5.2.7). Any operations performed on the SUT between the power and throughput tests must have the following characteristics:

- · They are related to data collection required for the benchmark or requested by the auditor
- · They are not likely to improve the performance of the throughput test

5.3.4.2 When measuring and reporting a throughput test, the number, S, of query streams must remain constant during the whole measurement interval. When results are reported with S query streams, these S streams must be the only ones executing during the measurement interval (i.e., it is not allowed to execute more than S query streams and report only the S best ones).

5.3.4.3 For query sequencing purposes (see Clause 5.3.5), each query stream within the throughput test must be assigned a unique stream identification number ranging from 1 to S, the number of query streams in the test.

5.3.4.4 When measuring and reporting a throughput test, a single refresh stream (see Clause 5.1.2.4) must be executed in parallel with the S query streams.

### 5.3.5 Query Sequencing Rules

5.3.5.1 The query sequencing rules apply to each and every query stream, whether part of the power test or part of the throughput test.

5.3.5.2 Each query set has an ordering number, O(s), based on the identification number, s, of the query stream executing the set. For example:

- The query set within the unique query stream of the power test has the ordering number O(00);
- The query set within the first query stream of the throughput test has the ordering number O(01);
- The query set within the last of S query streams of the throughput test has the ordering number O(S).

5.3.5.3 The sequencing of query executions is done within a query set. The ordering number, O(s), of a query set determines the order in which queries must be submitted (i.e., sequenced for execution) within that set and is independent of any other query set.

5.3.5.4 The query submission order of an ordering number, O(s), is given in Appendix A by the ordered set with reference s.

**Comment**: For tests where the list of ordered sets in Appendix A is exhausted, the last reference in the list must be followed by the first reference in the list (i.e., wrapping around to s = 00).

### 5.3.6 Measurement Interval

The measurement interval, $\mathbf{T_s}$, for the throughput test is measured in seconds as follows:

- It starts either when the first character of the executable query text of the first query of the first query stream is submitted to the SUT by the driver, or when the first character requesting the execution of the first refresh function is submitted to the SUT by the driver, whichever happens first;

**Comment**: In this clause a query stream is said to be first if it starts submitting queries before any other query streams.

- It ends either when the last character of output data from the last query of the last query stream is received by the driver from the SUT, or when the last transaction of the last refresh function has been completely and successfully committed at the SUT and a success message has been received by the driver from the SUT, whichever happens last.

**Comment**: In this clause the last query stream is defined to be that query stream whose output data are received last by the driver.

### 5.3.7 Timing Intervals

5.3.7.1 Each of the TPC-H queries and refresh functions must be executed in an atomic fashion and timed in seconds.

5.3.7.2 The timing interval, **QI(i,s)**, for the execution of the query, $Q_i$, within the query stream, s, must be measured between:

- The time when the first character of the executable query text is submitted to the SUT by the driver;
- The time when the first character of the next executable query text is submitted to the SUT by the driver, except for the last query of the set for which it is the time when the last character of the query's output data is received by the driver from the SUT.

**Comment**: All the operations that are part of the execution of a query (e.g., creation and deletion of a temporary table or a view) must be included in the timing interval of that query.

5.3.7.3 The timing interval, **RI(j,s)**, for the execution of the refresh function, RFj, within the query stream (for the power test) or the refresh stream (for the throughput test), where s is 0 for the power test and s is the position of the pair of refresh functions for the throughput test, must be measured between:

- The time when the first character requesting the execution of the refresh function is submitted to the SUT by the driver;
- The last transaction of the refresh function has been completely and successfully committed at the SUT and a success message has been received by the driver from the SUT.

5.3.7.4 The real-time clock used by the driver to compute the timing intervals must be capable of a resolution of at least 0.01 second.

5.3.7.5 The timing interval of each query and refresh function executed during both tests (i.e., during the power test and the throughput test) must be individually reported, rounded to the nearest 0.1 second. For example, 23.74 is rounded to 23.7, and 23.75 is rounded to 23.8. Values of less than 0.05 second must be rounded up to 0.1 second to avoid zero values.

5.3.7.6 The throughput test must include the execution of a single refresh stream. This refresh stream must be used exclusively for the execution of the New Sales refresh function (RF1) and the Old Sales refresh function (RF2).

**Comment**: The purpose of the refresh stream is to simulate a sequence of batched data modifications executing against the database to bring it up to date with its operational data source.

5.3.7.7 The refresh stream must execute a number of pairs of refresh functions serially (i.e., one RF1 followed by one RF2) equal to the number of query streams used for the throughput test.

**Comment**: The purpose of this requirement is to maintain a consistent read/write ratio across a wide range of number of query streams.

5.3.7.8 The scheduling of each refresh function within the refresh stream is left to the test sponsor with the only requirement that a given pair must complete before the next pair can be initiated and that within a pair RF1 must complete before RF2 can be initiated.

**Comment**: The intent of this Clause is to allow implementations that execute the refresh functions in parallel with the ad-hoc queries as well as systems that segregate query executions from database refreshes.

5.3.7.9 The scheduling of individual refresh functions within an instance of RF1 or RF2 is left to the test sponsor as long as they meet the requirements of Clause 2.26.2 and Clause 2.26.3.

**Comment**: The intent of this Clause is to allow test sponsors to "trickle" the scheduling of refresh functions to maintain a more even refresh load throughout the throughput test.

5.3.7.10 Prior to the execution of the refresh stream the DBGEN data used for RF1 and RF2 may only be generated, permuted and relocated to the SUT. Any other operations on these data, such as data formatting or database activity, must be included in the execution and the timing of the refresh functions.

## 5.4 Metrics

TPC-H defines three primary metrics:

- The TPC-H Composite Query-per-Hour Metric (QphH@Size) is the performance metric, defined in Clause 5.4.3;
- The price-performance metric is the TPC-H Price/Performance ($/QphH) and is defined in Clause 5.4.4;
- The Systems Availability Date, defined in Clause 7.2.2.1.

No other TPC-H metric exists. However, numerical quantities such as TPC-H Power and TPC-H Throughput (defined in Clause 5.4.1 and Clause 5.4.2 respectively) and S, the number of query streams in the throughput test, must be disclosed in the numerical quantities summary (see Clause 8.4.4).

### 5.4.1 TPC-H Power

5.4.1.1 The results of the power test are used to compute the TPC-H query processing **power** at the chosen database size. It is defined as the inverse of the geometric mean of the timing intervals, and must be computed as:

$$\text{TPC-H Power@Size} = \frac{3600 * SF}{\sqrt[24]{\prod_{i=1}^{i=22} QI(i,0) * \prod_{j=1}^{j=2} RI(j,0)}}$$

Where:
QI(i,0) is the timing interval, in seconds, of query $Q_i$ within the single query stream of the power test (see

Clause 5.3.7)

RI(j,0) is the timing interval, in seconds, of refresh function $RF_j$ within the single query stream of the power test (see Clause 5.3.7)

Size is the database size chosen for the measurement and SF the corresponding scale factor, as defined in Clause 4.1.3.

**Comment**: the power numerical quantity is based on a query per hour rate (i.e., factored by 3600).

5.4.1.2 The units of TPC-H Power@Size are Queries per hour * Scale-Factor, reported to one digit after the decimal point, rounded to the nearest 0.1.

5.4.1.3 The TPC-H Power can also be computed as:

$$\text{TPC-H Power@Size} = 3600 * \exp\left\{-\frac{1}{24}\left[\sum_{i=1}^{i=22}\ln(QI(i,0)) + \sum_{j=1}^{j=2}\ln(RI(j,0))\right]\right\} * SF$$

Where:

- ln(x) is the natural logarithm of x

5.4.1.4 If the ratio between the longest query timing interval and the shortest query timing interval in the power test is greater than 1000 (i.e., max[QI(i,0)]/min[QI(i,0)] > 1000), then all query timing intervals which are smaller than max[QI(i,0)]/1000 must be increased to max[QI(i,0)]/1000. The quantity max[QI(i,0)]/1000 must be treated as a timing interval as specified in Clause 5.3.7.5 for the purposes of computing the TPC-H Power@Size.

**Comment:** The adjusted query timings affect only TPC-H Power@Size and no other component of the FDR.

### 5.4.2 TPC-H **Throughput Numerical Quantity**

5.4.2.1 The results of the throughput test are used to compute TPC-H **Throughput** at the chosen database size. It is defined as the ratio of the total number of queries executed over the length of the measurement interval, and must be computed as:

TPC-H Throughput@Size = $(S*22*3600)/T_s * SF$

Where:

$T_s$ is the measurement interval defined in Clause 5.3.6

S is the number of query streams used in the throughput test.

Size is the database size chosen for the measurement and SF the corresponding scale factor, as defined in Clause 4.1.3.

5.4.2.2 The units of TPC-H Throughput@Size are Queries per hour * Scale-Factor, reported to one digit after the decimal point, rounded to the nearest 0.1.

### 5.4.3 **The** TPC-H **Composite Query-Per-Hour Performance Metric**

5.4.3.1 The numerical quantities TPC-H Power and TPC-H Throughput are combined to form the TPC-H **composite query-per-hour** performance metric which must be computed as:

$$\text{QphH@Size} = \sqrt{Power\ @\ Size * Throughput\ @\ Size}$$

5.4.3.2 The units of QphH@Size are Queries per hour * Scale-Factor, reported to one digit after the decimal point, rounded to the nearest 0.1.

### 5.4.4 The TPC-H Price/Performance Metric

5.4.4.1 The TPC-H Price/Performance metric at the chosen database size, TPC-H **Price-per-QphH@Size** , must be computed using the performance metric QphH@Size as follows:

TPC-H Price-per-QphH@Size = $/QphH@Size

Where:

$ is the total system price in the reported currency. The list of components to be priced is described in Clause 7.1 of this specification. How to price the components and how to express the total system price are defined in Clause 7 of the latest revision of the TPC Pricing Specification Version 1.

QphH@Size is the composite query-per-hour performance metric defined in Clause 5.4.3.

Size is the database size chosen for the measurement, as defined in Clause 4.1.3.

5.4.4.2 The units of Price-per-QphH@Size are expressed as in Clause 7 of the latest revision of TPC Pricing Specification Version 1. In the United States the price performance is expressed as USD per QphH@Size rounded to the highest cent (e.g., $12.123 must be shown as $12.13USD for price/performance).

### 5.4.5 Fair Metric Comparison

5.4.5.1 Comparisons of TPC-H benchmark results measured against databases of different sizes are believed to be misleading because database performance and capabilities may not scale up proportionally with an increase in database size and, similarly, the system price/performance ratio may not scale down with a decrease in database size.

If results measured against different database sizes (i.e., with different scale factors) appear in a printed or electronic communication, then each reference to a result or metric must clearly indicate the database size against which it was obtained. In particular, all textual references to TPC-H metrics (performance or price/performance) appearing must be expressed in the form that includes the size of the test database as an integral part of the metric's name; i.e. including the "@size" suffix. This applies to metrics quoted in text or tables as well as those used to annotate charts or graphs. If metrics are presented in graphical form, then the test database size on which metric is based must be immediately discernible either by appropriate axis labeling or data point labeling.

In addition, the results must be accompanied by a disclaimer stating:

"The TPC believes that comparisons of TPC-H results measured against different database sizes are misleading and discourages such comparisons".

5.4.5.2 Any TPC-H result is comparable to other TPC-H results regardless of the number of query streams used during the test (as long as the scale factors chosen for their respective test databases were the same).

### 5.4.6 Required Reporting Components

To be compliant with the TPC-H standard and the TPC's fair use policies, all public references to TPC-H results for a given configuration must include the following components:

- The size of the test database, expressed separately or as part of the metric's names (e.g., QphH@10GB);
- The TPC-H Performance Metric, QphH@Size;
- The TPC-H Price/Performance metric, $/QphH@Size;
- The availability date of the complete configuration (see Clause 8.3.8.2).

Following are two examples of compliant reporting of TPC-H results:

**Example 1**: At 10GB the RALF/3000 Server has a TPC-H Composite Query-per-Hour metric of 3010 when run against a 10GB database yielding a TPC-H Price/Performance of $1,202 per query-per-hour and will be available 1-Apr-99.

**Example 2**: The RALF/3000 Server, which will start shipping on 1-Apr-99, is rated 3,010 QphH@10GB and 1202 $/QphH@10GB.

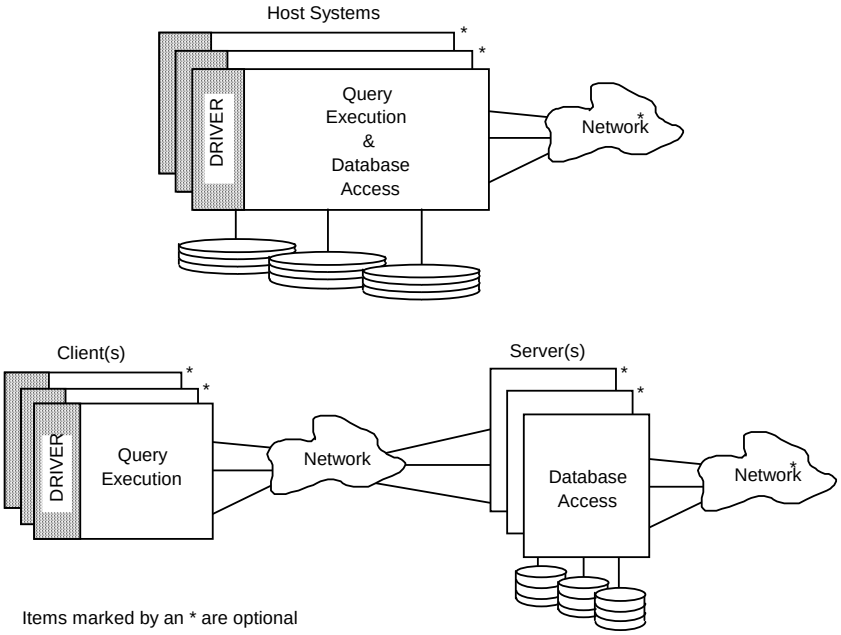
# 6: SUT AND DRIVER IMPLEMENTATION

## 6.1 Models of Tested Configurations

6.1.1 The tested and reported configuration(s) is composed of a driver that submits queries to a system under test (SUT). The SUT executes these queries and replies to the driver. The driver resides on the SUT hardware and software.

6.1.2 Figure 3: Two driver/SUT configurations, a "host-based" and a "client/server" configuration illustrates examples of driver/SUT configurations. The driver is the shaded area. The diagram also depicts the driver/SUT boundary (see Clause 5.2 and Clause 5.3) where timing intervals are measured.

**Figure 3: Two driver/SUT configurations, a "host-based" and a "client/server" configuration**




## 6.2 System Under Test (SUT) Definition

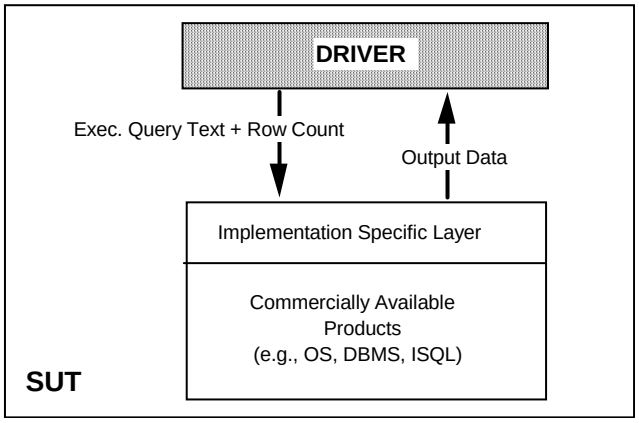
6.2.1 The SUT consists of:

- The host system(s) or server(s) including hardware and software supporting access to the database employed in the performance test and whose cost and performance are described by the benchmark metrics;
- One or more client processing units (e.g., front-end processors/cores/threads, workstations, etc.) that will execute the queries (if used);
- The hardware and software components needed to communicate with user interface devices;
- The hardware and software components of all networks required to connect and support the SUT components;
- Data storage media sufficient to satisfy both the scaling rules in Clause 4 and the ACID properties of Clause 3. The data storage media must hold all the data described in Clause 4 and be attached to the processing units(s).

6.2.2 All SUT components, as described in Clause 6.2.1, must be commercially available software or hardware products.

6.2.3 An implementation specific layer can be implemented on the SUT. This layer must be logically located between the driver and the SUT, as depicted by Figure 4: Implementation Specific Layer.

**Figure 4: Implementation Specific Layer**




6.2.4 An implementation specific layer, if present on the SUT, must be minimal, general purpose (i.e., not limited to the TPC-H queries) and its source code must be disclosed. Furthermore, the functions performed by an implementation specific layer must be strictly limited to the following:

- Database transaction control operations before and after each query execution;
- Cursor control and manipulation operations around the executable query text;
- Definition of procedures and data structures required to process dynamic SQL, including the communication of the executable query text to the commercially available layers of the SUT and the reception of the query output data;
- Communication with the commercially available layers of the SUT;
- Buffering of the query output data;
- Communication with the driver.

The following are examples of functions that the implementation specific layer shall not perform:

- Any modification of the executable query text;
- Any use of stored procedures to execute the queries;
- Any sorting or translation of the query output data;
- Any function prohibited by the requirements of Clause 5.2.7.

## 6.3 Driver Definition

6.3.1 The driver presents the workload to the SUT.

6.3.2 The driver is a logical entity that can be implemented using one or more programs, processes, or systems and performs the functions defined in Clause 6.3.3.

6.3.3 The driver can perform only the following functions:

- Generate a unique stream ID, starting with 1 (or 0 for the power test), for each query stream;
- Sequence queries for execution by the query streams (see Clause 5.3.5);
- Activate, schedule, and/or synchronize the execution of refresh functions in the refresh stream (see Clause 5.3.7.8);

- Generate the executable query text for each query;
- Generate values for the substitution parameters of each query;
- Complete the executable query text by replacing the substitution parameters by the values generated for them and, if needed, replacing the text-tokens by the query stream ID;
- Submit each complete executable query text to the SUT for execution, including the number of rows to be returned when specified by the functional query definition;
- Submit each executable refresh function to the SUT for execution;
- Receive the output data resulting from each query execution from the SUT;
- Measure the execution times of the queries and the refresh functions and compute measurement statistics;
- Maintain an audit log of query text and query execution output.

6.3.4 The generation of executable query text used by the driver to submit queries to the SUT does not need to occur on the SUT and does not have to be included in any timing interval.

6.3.5 The driver shall not perform any function other than those described in Clause 6.3.3. Specifically, the driver shall not perform any of the following functions:

- Performing, activating, or synchronizing any operation other than those mentioned in Clause 6.3.3;
- Delaying the execution of any query after the execution of the previous query other than for delays necessary to process the functions described in Clause 6.3.3. This delay must be reported and can not exceed half a second between any two consecutive queries of the same query stream;
- Modifying the compliant executable query text prior to its submission to the SUT;
- Embedding the executable query text within a stored procedure definition or an application program;
- Submitting to the SUT the values generated for the substitution parameters of a query other than as part of the executable query text submitted;
- Submitting to the SUT any data other than the instructions to execute the refresh functions, the compliant executable query text and, when specified by the functional query definition, the number of rows to be returned;
- Artificially extending the execution time of any query.

6.3.6 The driver is not required to be priced.

# 7: PRICING

This section defines the components, functional requirements of what is priced, and what substitutions are allowed. Rules for pricing the **Priced Configuration** and associated software and maintenance are included in the current revision of the TPC Pricing Specification Version 1 located at www.tpc.org.

## 7.1 Priced System

The system to be priced shall include the hardware and software components present in the System Under Test (SUT), a communication interface that can support user interface devices, additional operational components configured on the test system, and maintenance on all of the above

### 7.1.1 System Under Test

Calculation of the priced system consists of:

- Price of the SUT as tested and defined in Clause 6;
- Price of a communication interface capable of supporting the required number of user interface devices defined in Clause 7.1.2.1;
- Price of on-line storage for the database as described in Clause 7.1.3 and storage for all software included in the priced configuration;
- Price of additional products (software or hardware) required for customary operation, administration and maintenance of the SUT for a period of 3 years
- Price of all products required to create, execute, administer, and maintain the executable query texts or necessary to create and populate the test database.

Specifically excluded from the priced system calculation are:

- End-user communication devices and related cables, connectors, and concentrators;
- Equipment and tools used exclusively in the production of the full disclosure report;
- Equipment and tools used exclusively for the execution of the DBGEN or QGEN (see Clause 2.1.4 and Clause 4.2.1) programs.

### 7.1.2 User Interface Devices and Communications

7.1.2.1 The priced system must include the hardware and software components of a communication interface capable of supporting a number of user interface devices (e.g., terminals, workstations, PCs, etc.) at least equal to 10 times the number of query streams used for the throughput test (see Clause 5.3.4).

**Comment**: Test sponsors are encouraged to configure the SUT with a general-purpose communication interface capable of supporting a large number of user interface devices.

7.1.2.2 Only the interface is to be priced. Not to be included in the priced system are the user interface devices themselves and the cables, connectors and concentrators used to connect the user interface devices to the SUT. For example, in a configuration that includes an Ethernet interface to communicate with PCs, the Ethernet card and supporting software must be priced, but not the Ethernet cables and the PCs.

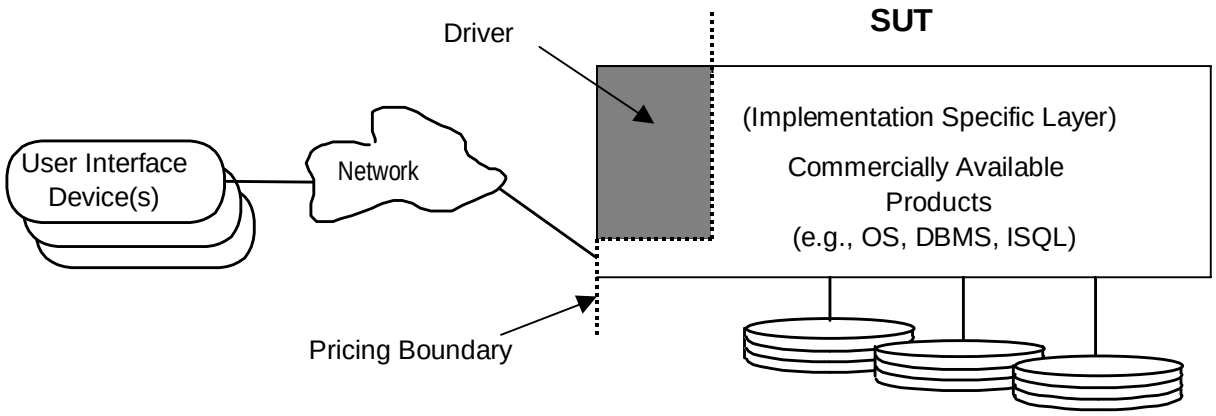
**Comment**: Active components (e.g., workstations, PCs, concentrators, etc.) can only be excluded from the priced system under the assumption that their role is strictly limited to submitting executable query text and receiving output data and that they do not participate in the query execution. All query processing performed by the tested con-

figuration is considered part of the performance test and can only be done by components that are included in the priced system.

7.1.2.3 The communication interface used must be an industry standard interface, such as Ethernet, Token Ring, or RS232.

7.1.2.4 The following diagram illustrates the boundary between what is priced (on the right) and what is not (on the left):

**Figure 5: The Pricing Boundary**




### 7.1.3 Database Storage and Recovery Log

7.1.3.1 Recovery data must be maintained for at least the duration of the run used to compute the published performance metrics.(see Clause 5.1.1.4).

Roll-back recovery data must be either in memory or in on-line storage at least until all transactions dependent on it are committed. Roll-forward recovery data may be stored on an off-line device provided that:

- The process that stores the roll-forward data is active during the measurement interval;
- The roll-forward data that is stored off-line during the measurement interval must be at least as great as the roll-forward recovery data that is generated during the period (i.e., the data may be first created in on-line storage and then moved to off-line storage, but the creation and the movement of the data must be in steady state);
- All ACID properties must be retained.

**Comment**: Storage is considered on-line if any record can be accessed randomly and updated within 1 second even if this access time requires the creation of a logical access path not present in the tested database. For example, a disk-based sequential file might require the creation of an index to satisfy the access time requirement. On-line storage may include magnetic disks, optical disks, or any combination of these, provided that the above mentioned access criteria are met.

7.1.3.2 While the benchmark requires the configuration of storage sufficient to hold the requisite recovery data as specified in Clause 7.1.3.1, it does not explicitly require the demonstration of rollforward recovery except as required by the ACID tests (See Clause 3.5).

7.1.3.3 The requirement to support at least eight hours of recovery log data can be met with storage on any durable media (see Clause 3.5.1) if all data required for recovery from failures listed in Clause 3.5.3 items 2 and 3 are on-line.

7.1.3.4 The storage that is required to be priced includes:

- storage required to execute the benchmark;

- storage to hold recovery data (see Clause 7.1.3);
- storage and media needed to assure that the test database meets the ACID requirements defined in Clause 3.

7.1.3.5 All storage required for the priced system must be present on the tested system.

### 7.1.4 Additional Operational Components

7.1.4.1 Additional products that might be included on a customer installed configuration, such as operator consoles and magnetic tape drives, are also to be included in the priced system if explicitly required for the operation, administration, or maintenance, of the priced system.

7.1.4.2 Copies of the software, on appropriate media, and a software load device, if required for initial load or maintenance updates, must be included.

7.1.4.3 The price of an Uninterruptible Power Supply, if specifically contributing to a durability solution, must be included (see Clause 3.5).

7.1.4.4 The price of all cables used to connect components of the system (except as noted in Clause 7.1.2.2) must be included.

## 7.2 Allowable Substitutions

Hardware product substitutions within the SUT, with the exceptions noted below require the benchmark to be re-run with the new components in order to re-establish compliance. The exceptions are individual disks and network interface cards (andassociated drivers) can be substituted.

# 8: FULL DISCLOSURE

Rules for reporting **Pricing** information are included in the current revision of the TPC Pricing Specification Version 1 located at www.tpc.org.

## 8.1 Reporting Requirements

8.1.1 A Full Disclosure Report (FDR) and Executive Summary are required.

8.1.2 The intent of this disclosure is to simplify comparison between results and for a customer to be able to replicate the results of this benchmark given appropriate documentation and products.

## 8.2 Format Guidelines

8.2.1 While established practice or practical limitations may cause a particular benchmark disclosure to differ from the examples provided in various small ways, every effort should be made to conform to the format guidelines. The intent is to make it as easy as possible for a reviewer to read, compare and evaluate material in different benchmark disclosures.

8.2.2 All sections of the report, including appendices, must be printed using font sizes of a minimum of 8 points.

8.2.3 The Executive Summary must be included near the beginning of the full disclosure report.

## 8.3 Full Disclosure Report Contents

The FDR should be sufficient to allow an interested reader to evaluate and, if necessary, recreate an implementation of TPC-H. If any sections in the FDR refer to another section of the report (e.g., an appendix), the names of the referenced scripts/programs must be clearly labeled in each section.

**Comment**: Since the building of a database may consist of a set of scripts and corresponding input files, it is important to disclose and clearly identify, by name, scripts and input files in the FDR.

The order and titles of sections in the test sponsor's full disclosure report must correspond with the order and titles of sections from the TPC-H standard specification (i.e., this document).

8.3.1 General Items

8.3.1.1 A statement identifying the benchmark sponsor(s) and other participating companies must be provided.

8.3.1.2 Settings must be provided for all customer-tunable parameters and options that have been changed from the defaults found in actual products, including but not limited to:

- Database tuning options;
- Optimizer/Query execution options;
- Query processing tool/language configuration parameters;
- Recovery/commit options;
- Consistency/locking options;
- Operating system and configuration parameters;
- Configuration parameters and options for any other software component incorporated into the pricing struc-

ture;

- Compiler optimization options.

**Comment 1**: In the event that some parameters and options are set multiple times, it must be easily discernible by an interested reader when the parameter or option was modified and what new value it received each time.

**Comment 2**: This requirement can be satisfied by providing a full list of all parameters and options, as long as all those that have been modified from their default values have been clearly identified and these parameters and options are only set once.

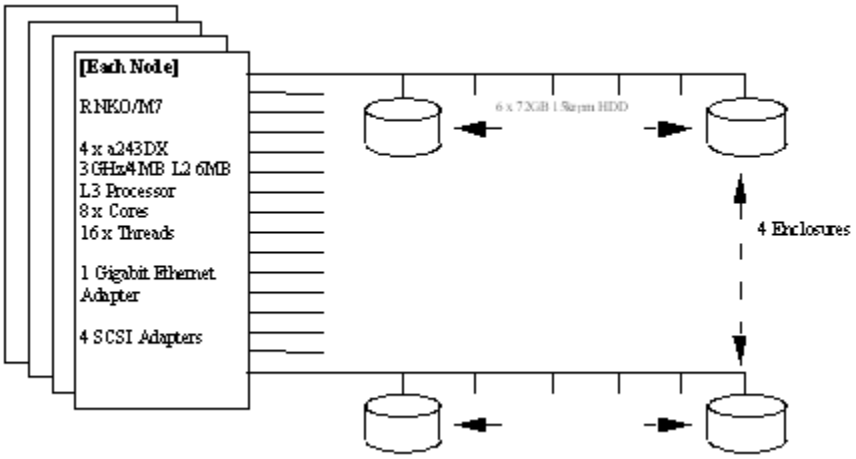
8.3.1.3 Explicit response to individual disclosure requirements specified in the body of earlier sections of this document must be provided.

8.3.1.4 Diagrams of both measured and priced configurations must be provided, accompanied by a description of the differences. This includes, but is not limited to:

- Total number of nodes used, total number and type of processors used/total number of cores used/total number of threads used (including sizes of L2 and L3 caches);
- Size of allocated memory, and any specific mapping/partitioning of memory unique to the test;
- Number and type of disk units (and controllers, if applicable);
- Number of channels or bus connections to disk units, including their protocol type;
- Number of LAN (e.g., Ethernet) connections, including routers, workstations, terminals, etc., that were physically used in the test or are incorporated into the pricing structure;
- Type and the run-time execution location of software components (e.g., DBMS, query processing tools/languages, middleware components, software drivers, etc.).

The following sample diagram illustrates a measured benchmark configuration using Ethernet, an external driver, and four processors each with two cores and four threads per node in the SUT. Note that this diagram does not depict or imply any optimal configuration for the TPC-H benchmark measurement.

**Figure 1: Sample Configuration Diagram (the front system box describes one node)**



LAN: Ethernet using NETplus routers
Total number of nodes used/total number of processors used/total number of cores used/total number of threads used: 4/16/32/64 x a243DX 3GHz with 4 MByte Second Level Cache
4 gigabyte of main memory
16 x SCSI-2 Fast Controllers
Disk: 96 x 2.1 gigabyte SCSI-2 drives

**Comment**: Detailed diagrams for system configurations and architectures can vary widely, and it is impossible to provide exact guidelines suitable for all implementations. The intent here is to describe the system components and connections in sufficient detail to allow independent reconstruction of the measurement environment. This example diagram shows homogeneous nodes. This does not preclude tests sponsors from using heterogeneous nodes as long as the system diagram reflects the correct system configuration.

### 8.3.2 Clause 1 - Logical Database Design Related Items

8.3.2.1 Listings must be provided for all table definition statements and all other statements used to set-up the test and qualification databases.

8.3.2.2 The physical organization of tables and indices within the test and qualification databases must be disclosed. If the column ordering of any table is different from that specified in Clause 1.4, it must be noted.

**Comment**: The concept of physical organization includes, but is not limited to: record clustering (i.e., rows from different logical tables are co-located on the same physical data page), index clustering (i.e., rows and leaf nodes of an index to these rows are co-located on the same physical data page), and partial fill-factors (i.e., physical data pages are left partially empty even though additional rows are available to fill them).

8.3.2.3 Horizontal partitioning of tables and rows in the test and qualification databases (see Clause ) must be disclosed.

8.3.2.4 Any replication of physical objects must be disclosed and must conform to the requirements of Clause 1.5.10.

### 8.3.3 Clause 2 - Query and Refresh function-Related Items

8.3.3.1 The query language used to implement the queries must be identified (e.g., “RALF/SQL-Plus”).

8.3.3.2 The method of verification for the random number generation must be described unless the supplied DBGEN and QGEN were used.

8.3.3.3 The method used to generate values for substitution parameters must be disclosed. If QGEN is not used for this purpose, then the source code of any non-commercial tool used must be disclosed. If QGEN is used, the version number, release number, modification number and patch level of QGEN must be disclosed.

8.3.3.4 The executable query text used for query validation must be disclosed along with the corresponding output data generated during the execution of the query text against the qualification database. If minor modifications (see Clause 2.2.3) have been applied to any functional query definitions or approved variants in order to obtain executable query text, these modifications must be disclosed and justified. The justification for a particular minor query modification can apply collectively to all queries for which it has been used. The output data for the power and throughput tests must be made available electronically upon request.

**Comment**: For query output of more than 10 rows, only the first 10 need to be disclosed in the FDR. The remaining rows must be made available upon request.

8.3.3.5 All the query substitution parameters used during the performance test must be disclosed in tabular format, along with the seeds used to generate these parameters.

8.3.3.6 The isolation level used to run the queries must be disclosed. If the isolation level does not map closely to one of the isolation levels defined in Clause 3.4, additional descriptive detail must be provided.

8.3.3.7 The details of how the refresh functions were implemented must be disclosed (including source code of any non-commercial program used).

### 8.3.4 Clause 3 - Database System Properties Related Items

8.3.4.1 The results of the ACID tests must be disclosed along with a description of how the ACID requirements were met. This includes disclosure of the code written to implement the ACID Transaction and Query.

### 8.3.5 Clause 4 - Scaling and Database Population Related Items

8.3.5.1 The cardinality (e.g., the number of rows) of each table of the test database, as it existed at the completion of the database load (see Clause 4.2.5), must be disclosed.

8.3.5.2 The distribution of tables and logs across all media must be explicitly described using a format similar to that shown in the following example for both the tested and priced systems.

**Comment**: Detailed diagrams for layout of database tables on disks can widely vary, and it is difficult to provide exact guidelines suitable for all implementations. The intent is to provide sufficient detail to allow independent reconstruction of the test database. The table below is an example of database layout descriptions and is not intended to describe any optimal layout for the TPC-H database.

**Table 12: Sample Database Layout Description**

| Controller | Disk Drive | Description of Content |
|---|---|---|
| 40A | 0 | Operating system, root |
| | 1 | System page and swap |
| | 2 | Physical log |
| | 3 | 100% of PART and SUPPLIER tables |
| 40B | 0 | 33% of CUSTOMER, ORDERS and LINEITEM tables |
| | 1 | 33% of CUSTOMER, ORDERS and LINEITEM tables |
| | 2 | 34% of CUSTOMER, ORDERS and LINEITEM tables |
| | 3 | 100% of PARTSUPP, NATION and REGION tables |

8.3.5.3 The mapping of database partitions/replications must be explicitly described.

**Comment**: The intent is to provide sufficient detail about partitioning and replication to allow independent reconstruction of the test database.

8.3.5.4 Implementations may use some form of RAID. The RAID level used must be disclosed for each device. If RAID is used in an implementation, the logical intent of its use must be disclosed. Three levels of usage are defined:

- Base tables only: In this case only the Base Tables (see Clause 1.2) are protected by any form of RAID;
- Base tables and auxiliary data structures: in addition to the protection of the base tables, implementations in this class must also employ RAID to protect all auxiliary data structures;

- Everything: implementations in this usage category must employ RAID to protect all database storage, including temporary or scratch space in addition to the base tables and auxiliary data structures.

8.3.5.5 The version number, release number, modification number, and patch level of DBGEN must be disclosed. Any modifications to the DBGEN (see Clause 4.2.1) source code (see Appendix B) must be disclosed. In the event that a program other than DBGEN was used to populate the database, it must be disclosed in its entirety.

8.3.5.6 The database load time for the test database (see Clause 4.3) must be disclosed.

8.3.5.7 The data storage ratio must be disclosed. It is computed by dividing the total data storage of the priced configuration (expressed in GB) by the size chosen for the test database as defined in Clause 4.1.3.1. The ratio must be reported to the nearest 1/100th, rounded up. For example, a system configured with 96 disks of 2.1 GB capacity for a 100GB test database has a data storage ratio of 2.02.

**Comment**: For the reporting of configured disk capacity, gigabyte (GB) is defined to be 2^30 bytes. Since disk manufacturers typically report disk size using base ten (i.e., GB = 10^9), it may be necessary to convert the advertised size from base ten to base two.

8.3.5.8 The details of the database load must be disclosed, including a block diagram illustrating the overall process. Disclosure of the load procedure includes all steps, scripts, input and configuration files required to completely reproduce the test and qualification databases.

8.3.5.9 Any differences between the configuration of the qualification database and the test database must be disclosed.

### 8.3.6 Clause 5 - Performance Metrics and Execution Rules Related Items

8.3.6.1 Any system activity on the SUT that takes place between the conclusion of the load test and the beginning of the performance test must be fully disclosed including listings of scripts or command logs.

8.3.6.2 The details of the steps followed to implement the power test (e.g., system boot, database restart, etc.) must be disclosed.

8.3.6.3 The timing intervals (see Clause 5.3.7) for each query and for both refresh functions must be reported for the power test.

8.3.6.4 The number of query streams used for the throughput test must be disclosed.

8.3.6.5 The start time and finish time for each query stream must be reported for the throughput test.

8.3.6.6 The total elapsed time of the measurement interval (see Clause 5.3.6) must be reported for the throughput test.

8.3.6.7 The start time and finish time for each refresh function in the refresh stream must be reported for the throughput test.

8.3.6.8 The timing intervals (see Clause 5.3.7) for each query of each stream and for each refresh function must be reported for the throughput test.

8.3.6.9 The computed performance metric, related numerical quantities and the price/performance metric must be reported.

8.3.6.10 The performance metric (QphH@Size) and the numerical quantities (TPC-H Power@Size and TPC-H Throughput@Size) from both of the runs must be disclosed (see Clause 5.4.1).

8.3.6.11 Any activity on the SUT that takes place between the conclusion of Run1 and the beginning of Run2 must be fully disclosed including listings of scripts or command logs along with any system reboots or database restarts.

### 8.3.7 Clause 6 - SUT and Driver Implementation Related Items

8.3.7.1 A detailed textual description of how the driver performs its functions, how its various components interact and any product functionalities or environmental settings on which it relies must be provided. All related source code, scripts and configuration files must be disclosed. The information provided should be sufficient for an independent reconstruction of the driver.

8.3.7.2 If an implementation specific layer is used, then a detailed description of how it performs its functions, how its various components interact and any product functionalities or environmental setting on which it relies must be provided. All related source code, scripts and configuration files must be disclosed. The information provided should be sufficient for an independent reconstruction of the implementation specific layer.

8.3.7.3 If profile-directed optimization as described in Clause 5.2.9 is used, such use must be disclosed. In particular, the procedure and any scripts used to perform the optimization must be disclosed.

### 8.3.8 Clause 9 - Audit Related Items

8.3.8.1 The auditor's agency name, address, phone number, and attestation letter with a brief audit summary report indicating compliance must be included in the full disclosure report. A statement should be included specifying whom to contact in order to obtain further information regarding the audit process.

## 8.4 Executive Summary

The executive summary is meant to be a high level overview of a TPC-H implementation. It should provide the salient characteristics of a benchmark execution (metrics, configuration, pricing, etc.) without the exhaustive detail found in the FDR. The executive summary has three components:

- Implementation Overview
- Pricing Spreadsheet
- Numerical Quantities

### 8.4.1 Page Layout

Each component of the executive summary should appear on a page by itself. Each page should use a standard header and format, including

- 1/2 inch margins, top and bottom;
- 3/4 inch left margin, 1/2 inch right margin;
- 2 pt. frame around the body of the page. All interior lines should be 1 pt.;
- Sponsor identification and System identification, each set apart by a 1 pt. rule, in 16-20 pt. Times Bold font;
- Benchmark name(i.e., TPC-H), revision using three tier versioning(e.g., 1.2.3) and report date, separated from other header items and each other by a 1 pt. Rule, in 9-12 pt. Times font.

**Comment 1:** It is permissible to use or include company logos when identifying the sponsor.

**Comment 2:** The report date must be disclosed with a precision of 1 day. The precise format is left to the test sponsor.

**Note:** Appendix E contains a sample executive summary. It is meant to help clarify the requirements in section 8.4 and is provided solely as an example.

### 8.4.2 Implementation Overview

The implementation overview page contains six sets of data, each laid out across the page as a sequence of boxes using 1 pt. rule, with a title above the required quantity. Both titles and quantities should use a 9-12 pt. Times font unless otherwise noted.

8.4.2.1 The first section contains the results that were obtained from the reported run of the Performance test.

**Table 13: Implementation Overview Information**

| Title | Quantity | Precision | Units | Font |
|---|---|---|---|---|
| Total System Cost | 3 yr. Cost of ownership (see Clause 7) | 1 | $1 | 16-20 pt. Bold |
| TPC-H Composite Query per Hour Metric | QphH (see Clause 5.4.3) | 0.1 | QphH@nGB | 16-20 pt. Bold |
| Price/Performance | $/QphH (see Clause 5.4.4) | 1 | $/QphH@nGB | 16-20 pt. Bold |

8.4.2.2 The next section details the system configuration

**Table 14: System Configuration Information**

| Title | Quantity | Precision | Units | Font |
|---|---|---|---|---|
| Database Size | Raw data size of test database (see Clause 4.1.3 and Clause 8.3.5.7) | 1 | GB (see Clause 8.3.5.7) | 9-12 pt. Times |
| DBMS Manager | Brand, Software Version of DBMS used | | | 9-12 pt. Times |
| Operating System | Brand, Software Version of OS used | | | 9-12 pt. Times |
| Other Software | Brand, Software Version of other software components | | | 9-12 pt. Times |
| System Availability Date | System Availability Date (see Clause 7.2.2.1) | 1 day | | 9-12 pt. Times |

**Comment:** The Software Version must uniquely identify the orderable software product referenced in the Priced Configuration (e.g., RALF/2000 4.2.1)

8.4.2.3 This section is the largest in the implementation overview, and contains a graphic representation of the reported query times. Each query and refresh function executed during the benchmark should be listed in the graph, with any query variants clearly identified. In addition:

- All labels and scales must use a 10 point Courier font, except for the legend and the graph title which must use a Times font;
- All line sizes must be 1 point;
- The legend must be reproduced as depicted in the example, and must be placed where needed to avoid over-

lapping any portion of the graph;

- The query time axis must labeled with no more than 8 values, including the zero origin;
- Each pair of bars must be separated by a gap of 50% of the bar's width;
- A zero-based linear scale must be used for the query times;
- The upper bound of the time scale must be no greater than 120% of the longest query timing interval;
- The bars used for the power test must be sized based on the measured (i.e., without the adjustment defined in Clause 5.4.1.4) query timing intervals of the power test, and must be solid white;
- The bars used for the throughput test must be sized based on the arithmetic mean by query type of the measured query timing intervals of the throughput test, and must be solid black;
- The geometric mean of the power test components must be computed using unadjusted timings of queries and refresh functions and must be placed on the graph as a dashed line labeled on top with its value. It must be expressed using the same format and precision as TPC-H Power specified in Clause 5;
- The arithmetic mean of the throughput test must be calculated using unadjusted timings with the following computation:

$$\frac{\sum_{i=1}^{22}\sum_{s=1}^{S} QI(i,s)}{(S \times 22)}$$

where QI(i,s) is defined in Clause 5.3.7.2, and S is defined in Clause 5.1.2.3;

- A solid line representing the mean must be placed on the graph intersecting only the queries and must be labeled on top with its value. The arithmetic mean of the throughput test must be expressed with the same format and precision as TPC-H Throughput specified in Clause 5;
- All query numbers must be followed by a variant letter when a variant was used in the tests.

8.4.2.4 This section contains the database load and sizing information

**Table 15: Database Load and Sizing Information**

| Title | Quantity | Precision | Units | Font |
|---|---|---|---|---|
| Database Load Time | Load Time (see Clause 4.3) | 1 sec. | hh:mm:ss | 9-12 pt. Times |
| Total Disk/Database Size | Data Storage Ratio (see Clause 8.3.5.7) | 0.01 | | 9-12 pt. Times |
| Load includes backup | Y/N (see Clause 4.3.6) | N/A | N/A | 9-12 pt. Times |
| RAID (Base tables only) | Y/N (see Clause 8.3.5.4) | N/A | N/A | 9-12 pt. Times |
| RAID (Base tables and auxiliary data structures) | Y/N (see Clause 8.3.5.4) | N/A | N/A | 9-12 pt. Times |
| RAID (Everything) | Y/N (see Clause 8.3.5.4) | N/A | N/A | 9-12 pt. Times |

8.4.2.5 The next section of the Implementation Overview should contain a synopsis of the SUT's major system components, including

- total number of nodes used/total number of processors used with their types and speeds in GHz/ total number of cores used/total number of threads used;
- Main and cache memory sizes;
- Network and I/O connectivity;
- Disk quantity and geometry.

If the implementation used a two-tier architecture, front-end and back-end systems should be detailed separately.

8.4.2.6 The final section of the implementation Overview should contain a note stating:

"Database Size includes only raw data (e.g., no temp, index, redundant storage space, etc.)."

### 8.4.3 Pricing Spreadsheet

The major categories in the Price Spreadsheet, as appropriate, are:

- Server Hardware
- Server Storage
- Server Software

Discounts (may optionally be included with above major category subtotal calculations)t.

8.4.4 Numerical Quantities Summary

The Numerical Quantities Summary page contains three sets of data, presented in tabular form, detailing the execution timings for the reported execution of the performance test. Each set of data should be headed by its given title and clearly separated from the other tables.

8.4.4.1 The first section contains measurement results from the benchmark execution.

Se ction Title: **Measurement Results**

| Item Title | Precision | Notes |
|---|---|---|
| Database Scale Factor | 1 | |
| Total Data Storage/Database Size | 0.01 | |
| Start of Database Load | yyyy-mm-dd hh:mm:ss | |
| End of Database Load | yyyy-mm-dd hh:mm:ss | |
| Database Load Time | hh:mm:ss | |
| Query Streams for Throughput Test | 1 | |
| TPC-H Power | 0.1 | |
| TPC-H Throughput | 0.1 | |
| TPC-H Composite Query-per-Hour Metric (QphH@Size) | 0.1 | |

| Item Title | Precision | Notes |
|---|---|---|
| Total System Price Over 3 Years | $1 | (1) |
| TPC-H Price Performance Metric ($/QphH@Size) | $0.01 | (1) |

(1) depending on the currency used for publication this sign has to be exchanged with the ISO currency symbol

8.4.4.2 The second section contains query and query stream timing information.

Section Title: **Measurement Intervals**

| Item Title | Precision | Notes |
|---|---|---|
| Measurement Interval in Throughput Test (Ts) | 1 second | |
| Duration of Stream Execution | | (1) |
| Stream | 1 | |
| Seed | 1 | |
| Start Date/Time | mm/dd/yy hh:mm:ss | |
| End Date/Time | mm/dd/yy hh:mm:ss | |
| Total Time | hh:mm:ss | |
| Refresh Start Date/Time | mm/dd/yy hh:mm:ss | |
| Refresh End Date/Time | mm/dd/yy hh:mm:ss | |

(1) The remaining items in this section should be reported as a sub-table, with one entry for each stream executed during the performance test.

8.4.4.3 The final section, titled **Timing Intervals (in Sec.)** contains individual query and refresh function timings. The data should be presented as a table with one entry for each query stream executed during the Performance Test. For each stream entry, the total elapsed time for each query in the stream and for its associated refresh functions should be reported separately to a resolution of 0.1 seconds. In addition, the minimum, maximum and average execution time for each query and refresh function must be reported to a resolution of 0.1 seconds.

## 8.5 Availability of the Full Disclosure Report

8.5.1 The full disclosure report must be readily available to the public at a reasonable charge, similar to charges for comparable documents by that test sponsor. The report must be made available when results are made public. In order to use the phrase “TPC Benchmark H”, the full disclosure report must have been submitted in hard copy and electronically to the TPC using the procedure described in the TPC Policies and Guidelines document.

8.5.2 The official full disclosure report must be available in English but may be translated to additional languages.

## 8.6 Revisions to the Full Disclosure Report

Revisions to the full disclosure documentation shall be handled as follows:

8.6.1 Substitutions will be open to challenge for a 60 day period. No other portion of the FDR is challengable.

8.6.2 During the normal product life cycle, problems will be uncovered that require changes, sometimes referred to as ECOs, FCOs, patches, updates, etc. When the cumulative result of applied changes causes the QphH/QphR rating of the system to decrease by more than 2% from the initially reported QphH/QphR, then the test sponsor is required to re-validate the benchmark results. The complete revision history is maintained following the query timing interval section showing the revision date and description.

8.6.3 Full disclosure report revisions may be required for other reasons according to TPC policies (see TPC Policy Document)

# 9: AUDIT

Rules for auditing **Pricing** information are included in the current revision of the TPC Pricing Specification Version 1 located at www.tpc.org.

## 9.1 General Rules

9.1.1 An independent audit of the benchmark results by a TPC certified auditor is required. The term independent is defined as "the outcome of the benchmark carries no financial benefit to the auditing agency other than fees earned directly related to the audit." In addition, the auditing agency cannot have supplied any performance consulting under contract for the benchmark.

In addition, the following conditions must be met:

a) The auditing agency cannot be financially related to the sponsor. For example, the auditing agency is financially related if it is a dependent division of the sponsor, the majority of its stock is owned by the sponsor, etc.

b) The auditing agency cannot be financially related to any one of the suppliers of the measured/priced configuration, e.g., the DBMS supplier, the disk supplier, etc.

9.1.2 The auditor's attestation letter is to be made readily available to the public as part of the full disclosure report. A detailed report from the auditor is not required.

9.1.3 TPC-H results can be used as the basis for new TPC-H results if and only if:

a) The auditor ensures that the hardware and software products are the same as those used in the prior result;

b) The auditor reviews the FDR of the new results and ensures that they match what is contained in the original sponsor's FDR;

c) The auditor can attest to the validity of the pricing used in the new FDR.

**Comment 1**: The intent of this clause is to allow a reseller of equipment from a given supplier to publish under the re-seller's name a TPC-H result already published by the supplier.

**Comment 2**: In the event that all conditions listed in Clause 9.1.3 are met, the auditor is not required to follow the remaining auditor's check list items from Clause 9.2.

9.1.4 Ensure that any auxiliary data structures satisfy the requirements of Clause 1.5.10.

9.1.5 In the event that a remote audit procedure is used in the context of a change-based audit, a remote connection to the SUT must be available for the auditor to verify selected audit items from Clause 9.2.

## 9.2 Auditor's Check List

### 9.2.1 Clause 1 Related Items

9.2.1.1 Verify that the data types used for each column are conformant. For example, verify that decimal columns can be incremented by 0.01 from -9,999,999,999.99.

9.2.1.2 Verify that the tables have the required list of columns.

9.2.1.3 Verify that the implementation rules are met by the test database.

9.2.1.4 Verify that the test database meets the data access transparency requirements.

9.2.1.5 Verify that conforming arbitrary data values can be inserted into any of the tables. Examples of verification tests include:

- Inserting a row that is a complete duplicate of an existing row except for a distinct primary key;
- Inserting a row with column values within the domain of the data type and check constraints but beyond the range of existing values.

9.2.1.6 Verify that the set of auxiliary data structures (as defined in Clause 1.5.7) that exist at the end of the load test are the same as those which exist at the end of the performance test. A similar check may be performed at any point during the performance test at the discretion of the auditor.

**Comment:** The purpose of this check is to verify that no auxiliary data structures automatically generated during the performance test may be accessed by more than one query execution.

### 9.2.2 Clause 2 Related Items

9.2.2.1 Verify that the basis for the SQL used for each query is either the functional query definition or an approved variant.

9.2.2.2 Verify that any deviation in the SQL from either the functional query definition or an approved variant is compliant with the specified minor query modifications. Verify that minor query modifications have been applied consistently to the set of functional query definitions or approved variants used.

9.2.2.3 Verify that the executable query text produces the required output when executed against the qualification database using the validation values for substitution parameters.

9.2.2.4 Note the method used to generate the values for substitution parameters (i.e., QGEN, modified version of QGEN, other method). If QGEN was used, note the version number, release number, modification number and patch level of QGEN. Verify that the version matches the benchmark specification.

9.2.2.5 Verify that the generated substitution parameters are reasonably diverse among the streams.

9.2.2.6 Verify that no aspect of the system under test, except for the database size, has changed between the demonstration of compliance against the qualification database and the execution of the reported measurements.

9.2.2.7 Verify that the refresh functions are implemented according to their definition.

9.2.2.8 Verify that the transaction requirements are met by the implementation of the refresh functions.

9.2.2.9 Note the method used to execute database maintenance operations

### 9.2.3 Clause 3 Related Items

9.2.3.1 Verify that the required ACID properties are supported by the system under test as configured for the execution of the reported measurements.

9.2.3.2 If one or more of the ACID tests defined in Clause 3 were not executed, note the rationale for waiving such demonstration of support of the related ACID property.

### 9.2.4 Clause 4 Related Items

9.2.4.1 Verify that the qualification database is properly scaled and populated.

9.2.4.2 Verify that the test database is properly scaled and populated. For example, extract some small number of rows at random from each table and verify that the values of the columns from these rows are the values generated by DBGEN.

9.2.4.3 Verify that the qualification and test databases were constructed in the same manner so that correct behavior on the qualification database is indicative of correct behavior on the test database.

9.2.4.4 Note the method used to populate the database (i.e., DBGEN, modified version of DBGEN, or other method). If DBGEN was used, note the version number, release number, modification number and patch level of DBGEN. Verify that the version matches the benchmark specification.

9.2.4.5 Verify that storage and processing elements that are not included in the priced configuration are physically removed or made inaccessible during the performance test.

9.2.4.6 Verify that the database load time is measured according to the requirements.

### 9.2.5 Clause 5 Related Items

9.2.5.1 Verify that the driver meets the requirements of Clause 5.2 and Clause 6.3.

9.2.5.2 Verify that the execution rules are followed for the power test.

9.2.5.3 Verify that the queries are executed against the test database.

9.2.5.4 Verify that the execution rules are followed for the throughput test.

9.2.5.5 Verify that a single stream is used for refresh functions in the throughput test and that the required number of refresh function pairs is executed according to the execution rules.

9.2.5.6 Verify that the query sequencing rules are followed.

9.2.5.7 Verify that the measurement interval for the throughput test is measured as required.

9.2.5.8 Verify that the method used to measure the timing intervals is compliant.

9.2.5.9 Verify that the metrics are computed as required. Note whether Clause 5.4.1.4 concerning the ratio between the longest and the shortest timing intervals had to be applied.

9.2.5.10 Verify that the reported metrics are repeatable.

### 9.2.6 Clause 6 Related Items

9.2.6.1 Verify that the composition of the SUT is compliant and that its components will be commercially available software or hardware products according to Clause 7.2.2.

9.2.6.2 Note whether an implementation specific layer is used and verify its compliance with Clause 6.2.4.

9.2.6.3 Verify that the driver's implementation is compliant.

9.2.6.4 Verify that any profile-directed optimization performed by the test sponsor conforms to the requirements of Clause 5.2.9.

### 9.2.7 Clause 8 Related Items

9.2.7.1 Verify that major portions of the full disclosure report are accurate and comply with the reporting requirements. This includes:

- The executive summary;
- The numerical quantity summary;
- The diagrams of both measured and priced configurations;
- The block diagram illustrating the database load process.

# Appendix A: ORDERED SETS

Following are the ordered sets that must be used for sequencing query execution as described in Clause 5.3.5. They are adapted from Moses and Oakford, *Tables of Random Permutations*, Stanford University Press, 1963. pp. 52-53.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Power Test | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 14 | 2 | 9 | 20 | 6 | 17 | 18 | 8 | 21 | 13 | 3 | 22 | 16 | 4 | 11 | 15 | 1 | 10 | 19 | 5 | 7 | 12 |
| Throughput Test | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 21 | 3 | 18 | 5 | 11 | 7 | 6 | 20 | 17 | 12 | 16 | 15 | 13 | 10 | 2 | 8 | 14 | 19 | 9 | 22 | 1 | 4 |
| 2 | 6 | 17 | 14 | 16 | 19 | 10 | 9 | 2 | 15 | 8 | 5 | 22 | 12 | 7 | 13 | 18 | 1 | 4 | 20 | 3 | 11 | 21 |
| 3 | 8 | 5 | 4 | 6 | 17 | 7 | 1 | 18 | 22 | 14 | 9 | 10 | 15 | 11 | 20 | 2 | 21 | 19 | 13 | 16 | 12 | 3 |
| 4 | 5 | 21 | 14 | 19 | 15 | 17 | 12 | 6 | 4 | 9 | 8 | 16 | 11 | 2 | 10 | 18 | 1 | 13 | 7 | 22 | 3 | 20 |
| 5 | 21 | 15 | 4 | 6 | 7 | 16 | 19 | 18 | 14 | 22 | 11 | 13 | 3 | 1 | 2 | 5 | 8 | 20 | 12 | 17 | 10 | 9 |
| 6 | 10 | 3 | 15 | 13 | 6 | 8 | 9 | 7 | 4 | 11 | 22 | 18 | 12 | 1 | 5 | 16 | 2 | 14 | 19 | 20 | 17 | 21 |
| 7 | 18 | 8 | 20 | 21 | 2 | 4 | 22 | 17 | 1 | 11 | 9 | 19 | 3 | 13 | 5 | 7 | 10 | 16 | 6 | 14 | 15 | 12 |
| 8 | 19 | 1 | 15 | 17 | 5 | 8 | 9 | 12 | 14 | 7 | 4 | 3 | 20 | 16 | 6 | 22 | 10 | 13 | 2 | 21 | 18 | 11 |
| 9 | 8 | 13 | 2 | 20 | 17 | 3 | 6 | 21 | 18 | 11 | 19 | 10 | 15 | 4 | 22 | 1 | 7 | 12 | 9 | 14 | 5 | 16 |
| 10 | 6 | 15 | 18 | 17 | 12 | 1 | 7 | 2 | 22 | 13 | 21 | 10 | 14 | 9 | 3 | 16 | 20 | 19 | 11 | 4 | 8 | 5 |
| 11 | 15 | 14 | 18 | 17 | 10 | 20 | 16 | 11 | 1 | 8 | 4 | 22 | 5 | 12 | 3 | 9 | 21 | 2 | 13 | 6 | 19 | 7 |
| 12 | 1 | 7 | 16 | 17 | 18 | 22 | 12 | 6 | 8 | 9 | 11 | 4 | 2 | 5 | 20 | 21 | 13 | 10 | 19 | 3 | 14 | 15 |
| 13 | 21 | 17 | 7 | 3 | 1 | 10 | 12 | 22 | 9 | 16 | 6 | 11 | 2 | 4 | 5 | 14 | 8 | 20 | 13 | 18 | 15 | 19 |
| 14 | 2 | 9 | 5 | 4 | 18 | 1 | 20 | 15 | 16 | 17 | 7 | 21 | 13 | 14 | 19 | 8 | 22 | 11 | 10 | 3 | 12 | 6 |
| 15 | 16 | 9 | 17 | 8 | 14 | 11 | 10 | 12 | 6 | 21 | 7 | 3 | 15 | 5 | 22 | 20 | 1 | 13 | 19 | 2 | 4 | 18 |
| 16 | 1 | 3 | 6 | 5 | 2 | 16 | 14 | 22 | 17 | 20 | 4 | 9 | 10 | 11 | 15 | 8 | 12 | 19 | 18 | 13 | 7 | 21 |
| 17 | 3 | 16 | 5 | 11 | 21 | 9 | 2 | 15 | 10 | 18 | 17 | 7 | 8 | 19 | 14 | 13 | 1 | 4 | 22 | 20 | 6 | 12 |
| 18 | 14 | 4 | 13 | 5 | 21 | 11 | 8 | 6 | 3 | 17 | 2 | 20 | 1 | 19 | 10 | 9 | 12 | 18 | 15 | 7 | 22 | 16 |
| 19 | 4 | 12 | 22 | 14 | 5 | 15 | 16 | 2 | 8 | 10 | 17 | 9 | 21 | 7 | 3 | 6 | 13 | 18 | 11 | 20 | 19 | 1 |
| 20 | 16 | 15 | 14 | 13 | 4 | 22 | 18 | 19 | 7 | 1 | 12 | 17 | 5 | 10 | 20 | 3 | 9 | 21 | 11 | 2 | 6 | 8 |
| 21 | 20 | 14 | 21 | 12 | 15 | 17 | 4 | 19 | 13 | 10 | 11 | 1 | 16 | 5 | 18 | 7 | 8 | 22 | 9 | 6 | 3 | 2 |
| 22 | 16 | 14 | 13 | 2 | 21 | 10 | 11 | 4 | 1 | 22 | 18 | 12 | 19 | 5 | 7 | 8 | 6 | 3 | 15 | 20 | 9 | 17 |
| 23 | 18 | 15 | 9 | 14 | 12 | 2 | 8 | 11 | 22 | 21 | 16 | 1 | 6 | 17 | 5 | 10 | 19 | 4 | 20 | 13 | 3 | 7 |
| 24 | 7 | 3 | 10 | 14 | 13 | 21 | 18 | 6 | 20 | 4 | 9 | 8 | 22 | 15 | 2 | 1 | 5 | 12 | 19 | 17 | 11 | 16 |
| 25 | 18 | 1 | 13 | 7 | 16 | 10 | 14 | 2 | 19 | 5 | 21 | 11 | 22 | 15 | 8 | 17 | 20 | 3 | 4 | 12 | 6 | 9 |
| 26 | 13 | 2 | 22 | 5 | 11 | 21 | 20 | 14 | 7 | 10 | 4 | 9 | 19 | 18 | 6 | 3 | 1 | 8 | 15 | 12 | 17 | 16 |
| 27 | 14 | 17 | 21 | 8 | 2 | 9 | 6 | 4 | 5 | 13 | 22 | 7 | 15 | 3 | 1 | 18 | 16 | 11 | 10 | 12 | 20 | 19 |
| 28 | 10 | 22 | 1 | 12 | 13 | 18 | 21 | 20 | 2 | 14 | 16 | 7 | 15 | 3 | 4 | 17 | 5 | 19 | 6 | 8 | 9 | 11 |
| 29 | 10 | 8 | 9 | 18 | 12 | 6 | 1 | 5 | 20 | 11 | 17 | 22 | 16 | 3 | 13 | 2 | 15 | 21 | 14 | 19 | 7 | 4 |
| 30 | 7 | 17 | 22 | 5 | 3 | 10 | 13 | 18 | 9 | 1 | 14 | 15 | 21 | 19 | 16 | 12 | 8 | 6 | 11 | 20 | 4 | 2 |
| 31 | 2 | 9 | 21 | 3 | 4 | 7 | 1 | 11 | 16 | 5 | 20 | 19 | 18 | 8 | 17 | 13 | 10 | 12 | 15 | 6 | 14 | 22 |
| 32 | 15 | 12 | 8 | 4 | 22 | 13 | 16 | 17 | 18 | 3 | 7 | 5 | 6 | 1 | 9 | 11 | 21 | 10 | 14 | 20 | 19 | 2 |
| 33 | 15 | 16 | 2 | 11 | 17 | 7 | 5 | 14 | 20 | 4 | 21 | 3 | 10 | 9 | 12 | 8 | 13 | 6 | 18 | 19 | 22 | 1 |
| 34 | 1 | 13 | 11 | 3 | 4 | 21 | 6 | 14 | 15 | 22 | 18 | 9 | 7 | 5 | 10 | 20 | 12 | 16 | 17 | 8 | 19 | 2 |
| 35 | 14 | 17 | 22 | 20 | 8 | 16 | 5 | 10 | 1 | 13 | 2 | 21 | 12 | 9 | 4 | 18 | 3 | 7 | 6 | 19 | 15 | 11 |
| 36 | 9 | 17 | 7 | 4 | 5 | 13 | 21 | 18 | 11 | 3 | 22 | 1 | 6 | 16 | 20 | 14 | 15 | 10 | 8 | 2 | 12 | 19 |
| 37 | 13 | 14 | 5 | 22 | 19 | 11 | 9 | 6 | 18 | 15 | 8 | 10 | 7 | 4 | 17 | 16 | 3 | 1 | 12 | 2 | 21 | 20 |
| 38 | 20 | 5 | 4 | 14 | 11 | 1 | 6 | 16 | 8 | 22 | 7 | 3 | 2 | 12 | 21 | 19 | 17 | 13 | 10 | 15 | 18 | 9 |
| 39 | 3 | 7 | 14 | 15 | 6 | 5 | 21 | 20 | 18 | 10 | 4 | 16 | 19 | 1 | 13 | 9 | 8 | 17 | 11 | 12 | 22 | 2 |
| 40 | 13 | 15 | 17 | 1 | 22 | 11 | 3 | 4 | 7 | 20 | 14 | 21 | 9 | 8 | 2 | 18 | 16 | 6 | 10 | 12 | 5 | 19 |

# Appendix B: APPROVED QUERY VARIANTS

Following are the approved TPC-H query variants as of the publication date of this version of the specification. As new query variants may be approved on an on-going basis, implementers are encouraged to obtain a copy of the latest list of approved query variants from the TPC office (see cover page for coordinates).

Some query variants include statements that create temporary tables. In these statements, column data types are designated in angle brackets (e.g., <Integer>) and refer to the list of data types specified in Clause 1.3.1.

- This appendix is also available in machine readable format -

To obtain a copy of the machine-readable appendices, **please contact the TPC** (see cover page).

## Q8

Variant A **(approved 11-Feb-1998)**

This variant replaces the CASE statement from the Functional Query Definition with equivalent DECODE() syntax.

The justification for this variant was Clause 2.2.4.3 (d), which allows for vendor-specific syntax that, while not SQL-92, provides a simple and direct mapping to approved SQL-92 syntax.

```
select
        o_year,
        sum(decode(nation, '[NATION]', volume, 0)) / sum(volume) as mkt_share
from
        (
                select
                        extract(year from o_orderdate) as o_year,
                        l_extendedprice * (1 - l_discount) as volume,
                        n2.n_name as nation
                from
                        part,
                        supplier,
                        lineitem,
                        orders,
                        customer,
                        nation n1,
                        nation n2,
                        region
                where
                        p_partkey = l_partkey
                        and s_suppkey = l_suppkey
                        and l_orderkey = o_orderkey
                        and o_custkey = c_custkey
                        and c_nationkey = n1.n_nationkey
                        and n1.n_regionkey = r_regionkey
                        and r_name = '[REGION]'
                        and s_nationkey = n2.n_nationkey
                        and o_orderdate between date '1995-01-01' and date '1996-12-31'
```

```
                    and p_type = '[TYPE]'
        ) all_nations
group by
        o_year
order by
        o_year;
```

## Q12

Variant A **(approved 11-Feb-1998)**

This variant replaces the CASE statement from the Functional Query Definition with equivalent DECODE() syntax.

The justification for this variant was Clause 2.2.4.3 (d), which allows for vendor-specific syntax that, while not SQL-92, provides a simple and direct mapping to approved SQL-92 syntax.

```
select
        l_shipmode,
        sum(decode(o_orderpriority, '1-URGENT', 1, '2-HIGH', 1, 0)) as
                high_line_count,
        sum(decode(o_orderpriority, '1-URGENT', 0, '2-HIGH', 0, 1)) as
                low_line_count
from
        orders,
        lineitem
where
        o_orderkey = l_orderkey
        and l_shipmode in ('[SHIPMODE1]', '[SHIPMODE2]')
        and l_commitdate < l_receiptdate
        and l_shipdate < l_commitdate
        and l_receiptdate >= date '[DATE]'
        and l_receiptdate < date '[DATE]' + interval '1' year
group by
        l_shipmode
order by
        l_shipmode;
```

## Q13

Variant A **(approved 5 March 1998)**

This variant was required by a vendor which did not support two aggregates in a nested table expression.

```
create view orders_per_cust[STREAM_ID] (custkey, ordercount) as
        select
                c_custkey,
                count(o_orderkey)
```

```
        from
                customer left outer join orders on
                        c_custkey = o_custkey
                and o_comment not like '%[WORD1]%[WORD2]%'
        group by
                c_custkey;

select
        ordercount,
        count(*) as custdist
from
        orders_per_cust[STREAM_ID]
group by
        ordercount
order by
        custdist desc,
        ordercount desc;

drop view orders_per_cust[STREAM_ID];
```

## Q14

Variant A **(approved 5 March 1998)**

This variant replaces the CASE statement with the equivalent DECODE() syntax.

```
select
        100.00 * sum(decode(substring(p_type from 1 for 5), 'PROMO',
                l_extendedprice * (1-l_discount), 0)) /
                        sum(l_extendedprice * (1-l_discount)) as promo_revenue
from
        lineitem,
        part
where
        l_partkey = p_partkey
        and l_shipdate >= date '[DATE]'
        and l_shipdate < date '[DATE]' + interval '1' month;
```

## Q15

Variant A **(approved 11-Feb-1998)**

This variant was approved because it contains new SQL syntax that is relevant to the benchmark. The SQL3 standard, which was moved to an Approved Committee Draft in May 1996, contains the definition of common table expressions. TPC-H already makes extensive use of nested table expressions. Common table expressions can be thought of as shared table expressions or "inline views" that last only for the duration of the query.

```
with revenue (supplier_no, total_revenue) as (
```

```
	select
		l_suppkey,
		sum(l_extendedprice * (1-l_discount))
	from
		lineitem
	where
		l_shipdate >= date '[DATE]'
		and l_shipdate < date '[DATE]' + interval '3' month
	group by
		l_suppkey
)
select
	s_suppkey,
	s_name,
	s_address,
	s_phone,
	total_revenue
from
	supplier,
	revenue
where
	s_suppkey = supplier_no
	and total_revenue = (
		select
			max(total_revenue)
		from
			revenue
	)
order by
	s_suppkey;
```

# Appendix C: QUERY VALIDATION

Following are the input values and output data for validation of executable query text against the qualification database.

- This appendix is available in machine-readable format only -

To obtain a copy of the machine-readable appendices, **please contact the TPC** (see Cover page).

# Appendix D: DATA AND QUERY GENERATION PROGRAMS

The QGEN (seeClause 2.1.4) and DBGEN (see Clause 4.2.1) programs should be used to generate the executable query text and the data that populate the TPC-H Databases. These programs produce flat files that can be used by the test sponsor to implement the benchmark.

- This appendix is available in machine readable format only -

To obtain a copy of the machine readable appendices, **please contact the TPC** (see Cover page).

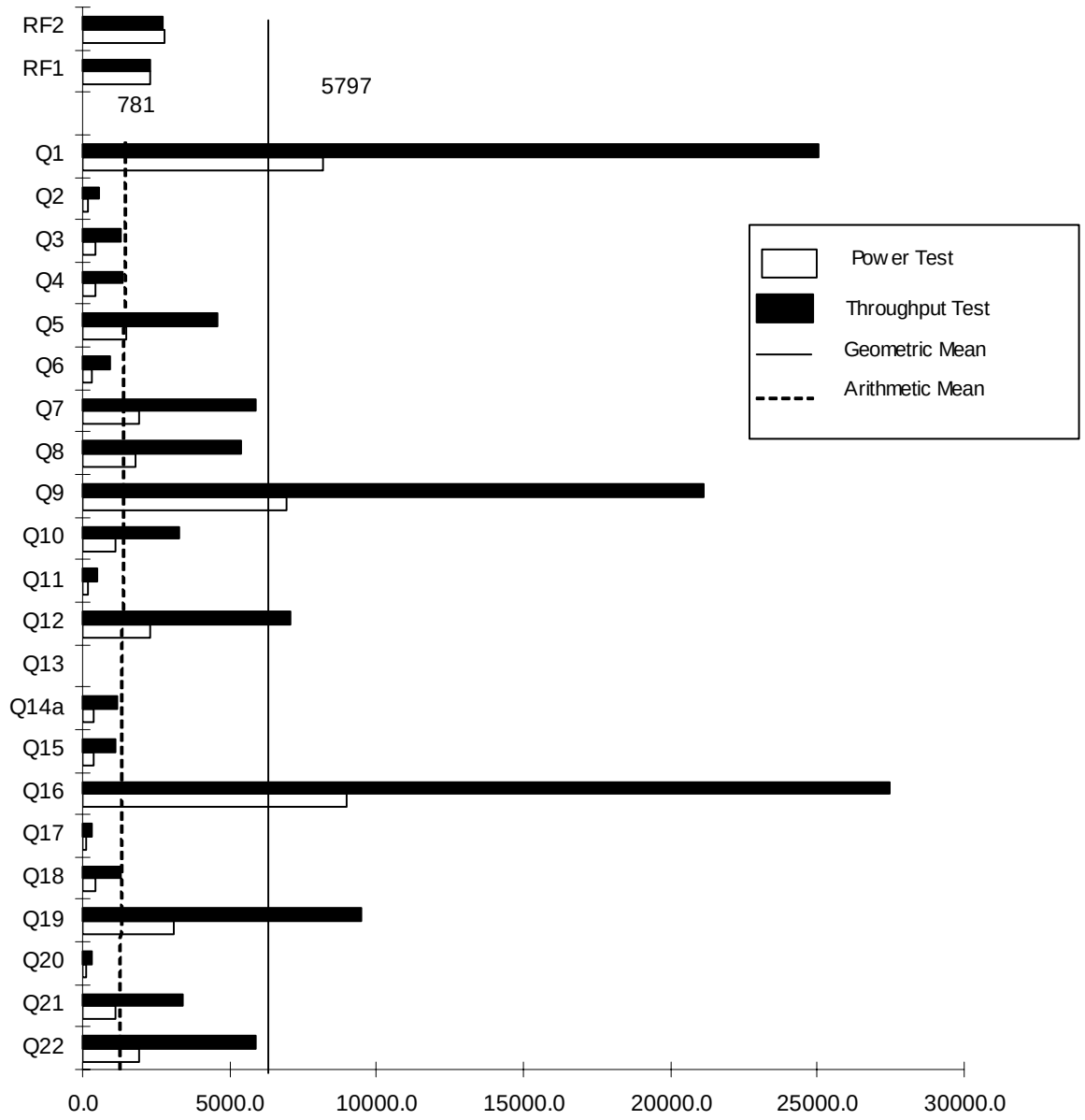
# Appendix E: SAMPLE EXECUTIVE SUMMARY

This appendix includes a sample Executive Summary. See Clause 8.4 for a detailed description of the required format of the Executive Summary. This sample is provided only as an illustration of the requirements set forth in Clause 8.4 and Clause 7.3 of the specification. In the event of a conflict between this example and the specification, the specification shall prevail.

| Sponsor (LOGO optional) | System Identification | Report Date: 25-Oct-99 |
|---|---|---|

| Total System Cost | Composite Query per Hour Rating | Price/Performance |
|---|---|---|
| **$287,913** | **702.8** QphH@100GB* | **410** $/QphH@100GB* |

| Database Size | Database Manager | Operating System | Other Software | Availability Date |
|---|---|---|---|---|
| **100GB*** | **RalfSoft/2000** | **OS/3** | **None** | **01-Jan-00** |




| Database Load Time = 24:14:12 | Load includes backup: Y | Total Data Storage / Database Size = 5.3 |
|---|---|---|
| RAID (base tables): N | RAID (Base Tables and Auxiliary Data Structures): Y | RAID (All): N |

## System Configuration

Processor/cores/threads: 4/16/32/64 x a243DX 3GHz with 4 MByte Second Level Cache
Memory: 2GB Main Memory
Disk Drives: 58 X 9GB Disk Drives
2 X 4GB Disk Drives

*Database Size includes only raw data (e.g., no temp, index, redundant storage space, etc.)

| **Sponsor** (LOGO optional) | **System Identification** | TPC-H Rev. 1.2.1<br>Report Date:<br>25-Oct-99 |
|---|---|---|

| Description | Part Number | Source | Reference Price | Qty | Disc | Extended Price | 5 Yr. Maint. Cost |
|---|---|---|---|---|---|---|---|
| **Server Hardware** | | | | | | | |
| Base Unit | JBOD-0001 | SponsorA | 8,250 | 1 | | 8,250 | 3,120 |
| CPU board with | | | | | | | |
| 2-200MHz/512KB CPUs | JBOD-0200 | SponsorA | 5,660 | 2 | | 11,320 | 1,920 |
| 512 MB memory kit | JBOD-M512 | SponsorA | 6,312 | 4 | | 25,248 | 16,080 |
| UltraWide SCSI Adapters | JBOD-SCSI-2691 | SponsorA | 700 | 4 | | 2,800 | 1,200 |
| Differential SCSI Cables | JBOD-FSR-2761 | SponsorA | 135 | 4 | | 540 | - |
| 4.3 GB internal disks | JBOD-DSK-2658 | SponsorA | 990 | 2 | | 1,980 | 960 |
| 72" Data Center Cabinet | JBOD-EVB-2747 | SponsorA | 3,150 | 1 | | 3,150 | 3,000 |
| 4 mm DAT Tape Drive | JBOD-TPE-2704 | SponsorA | 1,430 | 1 | | 1,430 | 480 |
| Command Package (see note 1) | JBOD-TTY1 | SponsorA | 1,080 | 1 | | 1,080 | - |
| UPS | JBOD-UPS-2738 | SponsorA | 3,500 | 1 | | 3,500 | 3,000 |
| 10/100baseT Ethernet Adapter | JBOD-NET-2683 | SponsorA | 100 | 1 | | 100 | 300 |
| Subtotal | | | | | | 59,398 | 30,060 |
| SponsorA Volume Discount | (see note 2) | | | | 20% | (11,880) | (6,012) |
| **Subtotal** | | | | | | **47,518** | **54,108** |
| | | | | | | | |
| **Storage** | | | | | | | |
| AnswerCache Storage Module | JBOD-ASM-8120 | SponsorA | - | 2 | | - | 38,400 |
| Intelligent Memory (96 MB kit) | 4500-001-7381 | ThirdParty2 | 1,998 | 2 | | 3,995 | See note 4 |
| 9 GB disk drives (20-count) | XV-689-20 | ThirdParty3 | 42,987 | 2 | | 85,974 | See note 4 |
| Storage Expansion Module | 4500-001-7410 | ThirdParty2 | 1,973 | 2 | | 3,945 | 9,600 |
| 9 GB disk drives (10-count) | XV-689-10 | ThirdParty3 | 26,555 | 2 | | 53,109 | See note 4 |
| **Subtotal** | | | | | | **147,023** | **48,000** |
| | | | | | | | |
| **Hardware Subtotal** | | | | | | **194,541** | **102,108** |
| | | | | | | | |
| **Server Software** | | | | | | | |
| OS/3 Server | NTE-0109 | ThirdParty1 | 809 | 1 | | 809 | 4,045 |
| RalfSoft 2000 (40 user license) | | SponsorB | 3,000 | 40 | | 120,000 | 133,600 |
| SponsorB Volume Discount | (see note 3) | | | | 29% | (34,800) | (38,744) |
| **Subtotal** | | | | | | **86,009** | **98,901** |
| | | | | | | | |
| **Total** | | | | | | **149,352** | **138,561** |

***Notes***

1. Includes Monitor/Keyboard/Mouse/Cables
2. Based on Sales volume in excess of $50,000
3. Available for all purchases of 35 seats or more
4. Maintenance included in AnswerCache Module

| | | | | |
|---|---|---|---|---|
| 5 Year Cost of Ownership | | | $ | 287,913 |
| QphD@ | 100 | GB | | 703 |
| **$/QphD@** | **100** | **GB** | **$** | **410** |

Audited By:

Prices used in TPC benchmarks reflect the actual prices a customer would pay for a one-time purchase of the stated components. Individually negotiated discounts are not permitted. Special prices based on assumptions about past or future purchases are not permitted. All discounts reflect standard pricing policies for the listed components. For complete details, see the pricing sections of the TPC benchmark specifications. If you find that the stated prices are not available according to these terms, please inform the TPC at pricing@tpc.org. Thank you.

| Sponsor (LOGO optional) | System Identification | TPC-H Rev. 1.2.1<br>Report Date:<br>25-Oct-99 |
|---|---|---|

Measurement Results

| | | | |
|---|---|---|---|
| Database Scale Factor | | 100 | |
| Total Data Storage / Database Size | | 5.30 | |
| Start of Database Load | | 01/30/98 | 8:59:57 |
| End of Database Load | | 01/31/98 | 9:14:09 |
| Database Load Time | | 24:14:12 | |
| Query Streams for Throughput Test | | 5 | |
| TPC-H Power | | 1845.7 | |
| TPC-H Throughput | | 267.6 | |
| Composite Query per Hour Rating (QphH@100GB) | | 702.8 | |
| Total System Price Over 5 Years | | 287,913 | |
| TPC-H Price/Performance Metric | | $ 410 | |

Measurement Intervals

| | | |
|---|---|---|
| Measurement Interval in Throughput Test (Ts) | = | 147984 |

Duration of stream execution

| | Seed | Stream Start Date/Time | Stream End Date/Time | Duration |
|---|---|---|---|---|
| Stream 0 | 131091409 | 01/31/98 9:14:09 | 01/31/98 22:15:54 | 13:01:45 |
| Stream 1 | 131091410 | 01/31/98 22:15:54 | 02/02/98 9:47:43 | 11:31:49 |
| Stream 2 | 131091411 | 01/31/98 22:15:54 | 02/02/98 11:12:14 | 12:56:20 |
| Stream 3 | 131091412 | 01/31/98 22:15:54 | 02/02/98 12:35:44 | 14:19:50 |
| Stream 4 | 131091413 | 01/31/98 22:15:54 | 02/02/98 13:59:12 | 15:43:18 |
| Stream 5 | 131091414 | 01/31/98 22:15:54 | 02/02/98 15:22:18 | 17:06:24 |
| Refresh | | 02/02/98 11:35:22 | 02/02/98 15:22:18 | 3:46:56 |

| Sponsor (LOGO optional) | System Identification | TPC-H Rev. 1.2.1<br>Report Date: 25-Oct-99 |
|---|---|---|

**TPC-H Timing Intervals (in seconds):**

| Query | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 |
|---|---|---|---|---|---|---|---|---|
| Stream 0 | 8197.3 | 190.0 | 432.1 | 446.4 | 1510.4 | 296.7 | 1926.9 | 1772.1 |
| Stream 1 | 25411.6 | 589.0 | 1339.4 | 1383.9 | 4682.2 | 919.7 | 5973.3 | 5493.6 |
| Stream 2 | 24141.0 | 559.6 | 1272.4 | 1314.7 | 4448.1 | 873.7 | 5674.6 | 5218.9 |
| Stream 3 | 24623.8 | 570.7 | 1297.9 | 1341.0 | 4537.0 | 891.2 | 5788.1 | 5323.3 |
| Stream 4 | 26428.0 | 612.6 | 1393.0 | 1439.2 | 4869.4 | 956.5 | 6212.2 | 5713.3 |
| Stream 5 | 24649.2 | 571.3 | 1299.2 | 1342.3 | 4541.7 | 892.1 | 5794.1 | 5328.8 |
| Min Qi | 24141.0 | 559.6 | 1272.4 | 1314.7 | 4448.1 | 873.7 | 5674.6 | 5218.9 |
| Max Qi | 26428.0 | 612.6 | 1393.0 | 1439.2 | 4869.4 | 956.5 | 6212.2 | 5713.3 |
| Avg Qi | 25050.7 | 580.6 | 1320.4 | 1364.2 | 4615.7 | 906.7 | 5888.5 | 5415.6 |
| **Query** | **Q9** | **Q10** | **Q11** | **Q12** | **Q13** | **Q14a** | **Q15** | **Q16** |
| Stream 0 | 6916.2 | 1084.8 | 158.0 | 2321.3 | 5.5 | 380.5 | 359.2 | 8983.3 |
| Stream 1 | 21440.3 | 3362.9 | 489.8 | 7196.0 | 17.1 | 1179.6 | 1113.6 | 27848.2 |
| Stream 2 | 20368.3 | 3194.8 | 465.3 | 6836.2 | 16.2 | 1120.7 | 1057.9 | 26455.8 |
| Stream 3 | 20775.6 | 3258.7 | 474.6 | 6972.9 | 16.5 | 1143.1 | 1079.1 | 26984.9 |
| Stream 4 | 22297.9 | 3497.4 | 509.4 | 7483.8 | 17.7 | 1226.8 | 1158.1 | 28962.1 |
| Stream 5 | 20797.1 | 3262.0 | 475.1 | 6980.1 | 16.5 | 1144.3 | 1080.2 | 27012.7 |
| Min Qi | 20368.3 | 3194.8 | 465.3 | 6836.2 | 16.2 | 1120.7 | 1057.9 | 26455.8 |
| Max Qi | 22297.9 | 3497.4 | 509.4 | 7483.8 | 17.7 | 1226.8 | 1158.1 | 28962.1 |
| Avg Qi | 21135.8 | 3315.2 | 482.8 | 7093.8 | 16.8 | 1162.9 | 1097.8 | 27452.7 |
| **Query** | **Q17** | **Q18** | **Q19** | **Q20** | **Q21** | **Q22** | **RF1** | **RF2** |
| Stream 0 | 103.4 | 417.6 | 3112.0 | 96.7 | 1106.3 | 1919.1 | 2274.406 | 2795.813 |
| Stream 1 | 320.5 | 1294.6 | 9647.2 | 299.8 | 3429.5 | 5949.2 | 2302.2 | 2685.1 |
| Stream 2 | 304.4 | 1229.8 | 9164.8 | 284.8 | 3258.1 | 5651.7 | 2291.5 | 2780.0 |
| Stream 3 | 310.5 | 1254.4 | 9348.1 | 290.5 | 3323.2 | 5764.8 | 2282.7 | 2732.8 |
| Stream 4 | 333.3 | 1346.3 | 10033.1 | 311.8 | 3566.7 | 6187.2 | 2312.6 | 2697.2 |
| Stream 5 | 310.8 | 1255.7 | 9357.8 | 290.8 | 3326.6 | 5770.7 | 2279.0 | 2707.3 |
| Min Qi | 304.4 | 1229.8 | 9164.8 | 284.8 | 3258.1 | 5651.7 | 2279.0 | 2685.1 |
| Max Qi | 333.3 | 1346.3 | 10033.1 | 311.8 | 3566.7 | 6187.2 | 2312.6 | 2780.0 |
| Avg Qi | 315.9 | 1276.2 | 9510.2 | 295.5 | 3380.8 | 5864.7 | 2293.6 | 2720.5 |