

TPC BENCHMARK™ C

Standard Specification Revision 5.6

December 2005

Transaction Processing Performance Council (TPC)
www.tpc.org
info@tpc.org
© 2005 Transaction Processing Performance Council

Acknowledgments

The TPC acknowledges the substantial contribution of François Raab, consultant to the TPC-C subcommittee and technical editor of the TPC-C benchmark standard. The TPC also acknowledges the work and contributions of the TPC-C subcommittee member companies: Amdahl, Bull, CDC, DEC, DG, Fujitsu/ICL, HP, IBM, Informix, Mips, Oracle, Sequent, Sun, Sybase, Tandem, and Unisys.

TPC Membership

(as of December 2005)

Full Members



Associate Members



Document History

<u>Date</u>	<u>Version</u>	<u>Description</u>
22 June 1992	Draft 6.6	Mail ballot version (proposed standard)
13 August 1992	Revision 1.0	Standard specification released to the public
1 June 1993	Revision 1.1	First minor revision
20 October 1993	Revision 2.0	First major revision
15 February 1995	Revision 3.0	Second major revision
4 June 1996	Revision 3.1	Minor changes to rev 3.1.
27 August 1996	Revision 3.2	Changed mix back to 3.0 values.
12 September 1996	Revision 3.2.1	Fixed Member list and added index
15 January 1997	Revision 3.2.2	Added wording for TAB Ids #197, 221 & 224
6 February 1997	Revision 3.2.3	Added wording for TAB Ids #205, 222 & 226
8 April 1997	Revision 3.3	New Clauses 2.3.6 & 9.2.2.3 (TAB Id #225)
9 April 1997	Revision 3.3.1	Wording added for availability date in Clause 8.1.8.3
25 June 1997	Revision 3.3.2	Editorial changes in Clauses 8.1.6.7 and 9.1.4
16 April 1998	Revision 3.3.3	Editorial changes in Clauses 2.5.2.2 and 4.2.2
24 August 1998	Revision 3.4	New Clause 5.7 and changed wording in Clause 8.3
25 August 1999	Revision 3.5	Modify wording in Clause 7.1.3
18 October 2000	Revision 5.0	Change pricing, 2 Hour Measurement, 60 Day Space
6 December 2000	Revision 5.0	7x24 Maintenance, Mail Ballot Draft
26 February 2001	Revision 5.0	Official Version 5.0 Specification
11 December 2002	Revision 5.1	Clause 3.5.4, PDO Limitations, Cluster Durability, Checkpoint Interval, Typographical Errors
11 December 2003	Revision 5.2	Modified Clause 7.1.3, Clause 8.3, Clause 7.1.6, and Clause 8.1.8.8. Replaced Clause 8.1.1.2, and Clause 8.1.8.2. Modified Clause 5.4.4 (truncated reported MQTh)
22 April 2004	Revision 5.3	Clause 8.3 (9), Executive Summary, Modify 7.1.3 (5), New Comment 4 and 5 to 7.1.3
21 April 2005	Revision 5.4	Modified Clause 3.3.3.2, Modified Clause 5.3.3, Integrated TPC Pricing Specification
20 October 2005	Revision 5.5	Modified Clauses 8.1.1.7 and 8.1.9.1, Added Comment to Clause 8.1.1.2 and added Clause 9.2.9.
8 December 2005	Revision 5.6	Modified Clauses 5.5.1.2, 8.1.1.2. Replaced 6.6.6

TPC Benchmark™, TPC-C, and tpmC are trademarks of the Transaction Processing Performance Council.

Permission to copy without fee all or part of this material is granted provided that the TPC copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the Transaction Processing Performance Council. To copy otherwise requires specific permission.

TABLE OF CONTENTS

TABLE OF CONTENTS.....	4
Clause 0: PREAMBLE	6
0.1 Introduction.....	6
0.2 General Implementation Guidelines.....	7
0.3 General Measurement Guidelines.....	8
Clause 1: LOGICAL DATABASE DESIGN.....	9
1.1 Business and Application Environment.....	9
1.2 Database Entities, Relationships, and Characteristics	10
1.3 Table Layouts	10
1.4 Implementation Rules.....	17
1.5 Integrity Rules.....	18
1.6 Data Access Transparency Requirements	19
Clause 2: TRANSACTION and TERMINAL PROFILES.....	20
2.1 Definition of Terms	20
2.2 General Requirements for Terminal I/O.....	22
2.3 General Requirements for Transaction Profiles	25
2.4 The New-Order Transaction	27
2.5 The Payment Transaction.....	32
2.6 The Order-Status Transaction.....	36
2.7 The Delivery Transaction	39
2.8 The Stock-Level Transaction.....	43
Clause 3: TRANSACTION and SYSTEM PROPERTIES	46
3.1 The ACID Properties.....	46
3.2 Atomicity Requirements.....	46
3.3 Consistency Requirements	47
3.4 Isolation Requirements	50
3.5 Durability Requirements	56
Clause 4: SCALING and DATABASE POPULATION.....	59
4.1 General Scaling Rules.....	59
4.2 Scaling Requirements.....	59
4.3 Database Population	62
Clause 5: PERFORMANCE METRICS and RESPONSE TIME	67
5.1 Definition of Terms	67
5.2 Pacing of Transactions by Emulated Users.....	67
5.3 Response Time Definition	70
5.4 Computation of Throughput Rating.....	71
5.5 Measurement Interval Requirements	72
5.6 Required Reporting	74
5.7 Primary Metrics	76
Clause 6: SUT, DRIVER, and COMMUNICATIONS DEFINITION	77
6.1 Models of the Target System.....	77
6.2 Test Configuration.....	78
6.3 System Under Test (SUT) Definition	78
6.4 Driver Definition	78
6.5 Communications Interface Definitions.....	79

6.6	Further Requirements on the SUT and Driver System.....	79
Clause 7: PRICING		83
7.1	Pricing Methodology.....	83
7.2	Priced System.....	83
7.3	Required Reporting.....	85
Clause 8: FULL DISCLOSURE.....		86
8.1	Full Disclosure Report Requirements.....	86
Clause 9: AUDIT		97
9.1	General Rules	97
9.2	Auditor's check list.....	97
Index.....		101
Appendix A: SAMPLE PROGRAMS		104
A.1	The New-Order Transaction.....	104
A.2	The Payment Transaction.....	106
A.3	The Order-Status Transaction.....	108
A.4	The Delivery Transaction	110
A.5	The Stock-Level Transaction.....	112
A.6	Sample Load Program	113
Appendix B: EXECUTIVE SUMMARY STATEMENT.....		126
Appendix C: NUMERICAL QUANTITIES SUMMARY		128

Clause 0: PREAMBLE

0.1 Introduction

TPC Benchmark™ C (TPC-C) is an OLTP workload. It is a mixture of read-only and update intensive transactions that simulate the activities found in complex OLTP application environments. It does so by exercising a breadth of system components associated with such environments, which are characterized by:

- The simultaneous execution of multiple transaction types that span a breadth of complexity
- On-line and deferred transaction execution modes
- Multiple on-line terminal sessions
- Moderate system and application execution time
- Significant disk input/output
- Transaction integrity (ACID properties)
- Non-uniform distribution of data access through primary and secondary keys
- Databases consisting of many tables with a wide variety of sizes, attributes, and relationships
- Contention on data access and update

The performance metric reported by TPC-C is a "business throughput" measuring the number of orders processed per minute. Multiple transactions are used to simulate the business activity of processing an order, and each transaction is subject to a response time constraint. The performance metric for this benchmark is expressed in transactions-per-minute-C (tpmC). To be compliant with the TPC-C standard, all references to TPC-C results must include the tpmC rate, the associated price-per-tpmC, and the availability date of the priced configuration.

Although these specifications express implementation in terms of a relational data model with conventional locking scheme, the database may be implemented using any commercially available database management system (DBMS), database server, file system, or other data repository that provides a functionally equivalent implementation. The terms "table", "row", and "column" are used in this document only as examples of logical data structures.

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

Despite the fact that this benchmark offers a rich environment that emulates many OLTP applications, this benchmark does not reflect the entire range of OLTP requirements. In addition, the extent to which a customer can achieve the results reported by a vendor is highly dependent on how closely TPC-C 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-C 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, insofar as they adhere to the model described and pictorially illustrated in Clause 6. A Full Disclosure Report of the implementation details, as specified in Clause 8, must be made available along with the reported results.

Comment: While separated from the main text for readability, comments are a part of the standard and must be enforced. However, the sample programs, included as Appendix A, the summary statements, included as Appendix B, and the numerical quantities summary, included as Appendix C, are provided only as examples and are specifically not part of this standard.

0.2 General Implementation Guidelines

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-A models and represents high-volume, simple OLTP environments).
- A significant number of users in the market segment the benchmark models or represents would plausibly implement.

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, 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 specials," implementations that improve benchmark results but not real-world performance or pricing, are prohibited.

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

- Is the implementation generally available, documented, and supported?
- Does the implementation have significant restrictions on its use or applicability that limits its use beyond TPC benchmarks?
- Is the implementation or part of the implementation poorly integrated into the larger product?
- Does the implementation take special advantage of the limited nature of TPC benchmarks (e.g., transaction profile, transaction mix, transaction concurrency and/or contention, transaction isolation) in a manner that would not be generally applicable to the environment the benchmark represents?
- 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.)
- Does the implementation require uncommon sophistication on the part of the end-user, programmer, or system administrator?

- Is the pricing unusual or non-customary for the vendor or unusual or non-customary to normal business practices? See the current revision of Version 1 of the TPC Pricing Specification for additional information.
- Is the implementation being used (including beta) or purchased by end-users in the market area the benchmark represents? How many? Multiple sites? If the implementation is not currently being used by end-users, is there any evidence to indicate that it will be used by a significant number of users?

0.3 General Measurement Guidelines

TPC benchmark results are expected to be accurate representations of system performance. Therefore, there are certain guidelines which are expected to be followed when measuring those results. The approach or methodology is explicitly outlined in or described in the specification.

- The approach is an accepted engineering practice or standard.
- The approach does not enhance the result.
- Equipment used in measuring 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 benchmark requirements.

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

Clause 1: LOGICAL DATABASE DESIGN

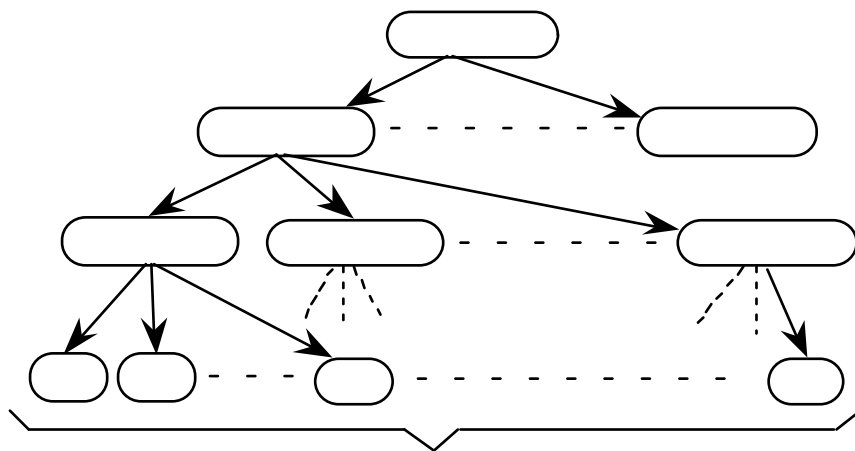
1.1 Business and Application Environment

TPC Benchmark™ C is comprised of a set of basic operations designed to exercise system functionalities in a manner representative of complex OLTP application environments. These basic operations have been given a life-like context, portraying the activity of a wholesale supplier, to help users relate intuitively to the components of the benchmark. The workload is centered around the activity of processing orders and provides a logical database design, which can be distributed without structural changes to transactions.

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

The purpose of a benchmark is to reduce the diversity of operations found in a production application, while retaining the application's essential performance characteristics, namely: the level of system utilization and the complexity of operations. A large number of functions have to be performed to manage a production order entry system. Many of these functions are not of primary interest for performance analysis, since they are proportionally small in terms of system resource utilization or in terms of frequency of execution. Although these functions are vital for a production system, they merely create excessive diversity in the context of a standard benchmark and have been omitted in TPC-C.

The Company portrayed by the benchmark is a wholesale supplier with a number of geographically distributed sales districts and associated warehouses. As the Company's business expands, new warehouses and associated sales districts are created. Each regional warehouse covers 10 districts. Each district serves 3,000 customers. All warehouses maintain stocks for the 100,000 items sold by the Company. The following diagram illustrates the warehouse, district, and customer hierarchy of TPC-C's business environment.



- **fixed text, size N** means that the attribute must be able to hold any string of characters of a fixed length of N.
- **date and time** means that the attribute must be able to hold any date between 1st January 1900 and 31st December 2100 with a resolution of at least one second.
- **numeric, N digits** means that the attribute must be able to hold any N decimal digits value. Numeric fields that contain monetary values (W_YTD, D_YTD, C_CREDIT_LIM, C_BALANCE, C_YTD_PAYMENT, H_AMOUNT, OL_AMOUNT, I_PRICE) must use data types that give exact representation to at least the smallest monetary unit in the currency being used in the benchmark implemented. For example, C_BALANCE in U.S. dollars may be represented as (12,2) digit signed decimal (with implicit scaling), or scaled to cents in a signed integer of at least 41 bits, or scaled to cents in a double precision (64 bit) REAL.
- **null** means out of the range of valid values for a given attribute and always the same value for that attribute.

Comment 1: For each table, the following list of attributes can be implemented in any order, using any physical representation available from the tested system.

Comment 2: Table and attribute names are used for illustration purposes only; different names may be used by the implementation.

WAREHOUSE Table Layout

<u>Field Name</u>	<u>Field Definition</u>	<u>Comments</u>
W_ID	2*W unique IDs	
W_NAME	variable text, size 10	
W_STREET_1	variable text, size 20	
W_STREET_2	variable text, size 20	
W_CITY	variable text, size 20	
W_STATE	fixed text, size 2	
W_ZIP	fixed text, size 9	
W_TAX	numeric, 4 digits	
W_YTD	numeric, 12 digits	

Primary Key: W_ID

DISTRICT Table Layout

<u>Field Name</u>	<u>Field Definition</u>	<u>Comments</u>
D_ID	20 unique IDs	
D_W_ID	2*W unique IDs	
D_NAME	variable text, size 10	
D_STREET_1	variable text, size 20	
D_STREET_2	variable text, size 20	
D_CITY	variable text, size 20	
D_STATE	fixed text, size 2	
D_ZIP	fixed text, size 9	
D_TAX	numeric, 4 digits	
D_YTD	numeric, 12 digits	
D_NEXT_O_ID	10,000,000 unique IDs	

Primary Key: (D_W_ID, D_ID)

D_W_ID Foreign Key, references W_ID

CUSTOMER Table Layout

<u>Field Name</u>	<u>Field Definition</u>	<u>Comments</u>
C_ID	96,000 unique IDs	
C_D_ID	20 unique IDs	
C_W_ID	2*W unique IDs	
C_FIRST	variable text, size 16	
C_MIDDLE	fixed text, size 2	
C_LAST	variable text, size 16	
C_STREET_1	variable text, size 20	
C_STREET_2	variable text, size 20	
C_CITY	variable text, size 20	
C_STATE	fixed text, size 2	
C_ZIP	fixed text, size 9	
C_PHONE	fixed text, size 16	
C_SINCE	date and time	
C_CREDIT	fixed text, size 2	!" # !
C_CREDIT_LIM	numeric, 12 digits	
C_DISCOUNT	numeric, 4 digits	
C_BALANCE	signed numeric, 12 digits	
C_YTD_PAYMENT	numeric, 12 digits	
C_PAYMENT_CNT	numeric , 4 digits	
C_DELIVERY_CNT	numeric, 4 digits	
C_DATA	variable text, size 500	\$ %

Primary Key: (C_W_ID, C_D_ID, C_ID)

(C_W_ID, C_D_ID) Foreign Key, references (D_W_ID, D_ID)

HISTORY Table Layout

<u>Field Name</u>	<u>Field Definition</u>	<u>Comments</u>
H_C_ID	96,000 unique IDs	
H_C_D_ID	20 unique IDs	
H_C_W_ID	2*W unique IDs	
H_D_ID	20 unique IDs	
H_W_ID	2*W unique IDs	
H_DATE	date and time	
H_AMOUNT	numeric, 6 digits	
H_DATA	variable text, size 24	\$ %

Primary Key: none

(H_C_W_ID, H_C_D_ID, H_C_ID) Foreign Key, references (C_W_ID, C_D_ID, C_ID)

(H_W_ID, H_D_ID) Foreign Key, references (D_W_ID, D_ID)

Comment: Rows in the History table do not have a primary key as, within the context of the benchmark, there is no need to uniquely identify a row within this table.

Note: The TPC-C application does not have to be capable of utilizing the increased range of C_ID values beyond 6,000.

NEW-ORDER Table Layout

<u>Field Name</u>	<u>Field Definition</u>	<u>Comments</u>
NO_O_ID	10,000,000 unique IDs	
NO_D_ID	20 unique IDs	
NO_W_ID	2*W unique IDs	

Primary Key: (NO_W_ID, NO_D_ID, NO_O_ID)

(NO_W_ID, NO_D_ID, NO_O_ID) Foreign Key, references (O_W_ID, O_D_ID, O_ID)

ORDER Table Layout

<u>Field Name</u>	<u>Field Definition</u>	<u>Comments</u>
O_ID	10,000,000 unique IDs	
O_D_ID	20 unique IDs	
O_W_ID	2*W unique IDs	
O_C_ID	96,000 unique IDs	
O_ENTRY_D	date and time	
O_CARRIER_ID	10 unique IDs, or null	
O_OL_CNT	from 5 to 15	% &'
O_ALL_LOCAL	numeric, 1 digit	

Primary Key: (O_W_ID, O_D_ID, O_ID)

(O_W_ID, O_D_ID, O_C_ID) Foreign Key, references (C_W_ID, C_D_ID, C_ID)

ORDER-LINE Table Layout

<u>Field Name</u>	<u>Field Definition</u>	<u>Comments</u>
OL_O_ID	10,000,000 unique IDs	
OL_D_ID	20 unique IDs	
OL_W_ID	2*W unique IDs	
OL_NUMBER	15 unique IDs	
OL_I_ID	200,000 unique IDs	
OL_SUPPLY_W_ID	2*W unique IDs	
OL_DELIVERY_D	date and time, or null	
OL_QUANTITY	numeric, 2 digits	
OL_AMOUNT	numeric, 6 digits	
OL_DIST_INFO	fixed text, size 24	

Primary Key: (OL_W_ID, OL_D_ID, OL_O_ID, OL_NUMBER)

(OL_W_ID, OL_D_ID, OL_O_ID) Foreign Key, references (O_W_ID, O_D_ID, O_ID)

(OL_SUPPLY_W_ID, OL_I_ID) Foreign Key, references (S_W_ID, S_I_ID)

ITEM Table Layout

<u>Field Name</u>	<u>Field Definition</u>	<u>Comments</u>
I_ID	200,000 unique IDs	
I_IM_ID	200,000 unique IDs	(" () (
I_NAME	variable text, size 24	
I_PRICE	numeric, 5 digits	
I_DATA	variable text, size 50	# %

Primary Key: I_ID

STOCK Table Layout

<u>Field Name</u>	<u>Field Definition</u>	<u>Comments</u>
S_I_ID	200,000 unique IDs	
S_W_ID	2*W unique IDs	
S_QUANTITY	numeric, 4 digits	
S_DIST_01	fixed text, size 24	
S_DIST_02	fixed text, size 24	
S_DIST_03	fixed text, size 24	
S_DIST_04	fixed text, size 24	
S_DIST_05	fixed text, size 24	
S_DIST_06	fixed text, size 24	
S_DIST_07	fixed text, size 24	
S_DIST_08	fixed text, size 24	
S_DIST_09	fixed text, size 24	
S_DIST_10	fixed text, size 24	
S_YTD	numeric, 8 digits	
S_ORDER_CNT	numeric, 4 digits	
S_REMOTE_CNT	numeric, 4 digits	
S_DATA	variable text, size 50	\$ * %

Primary Key: (S_W_ID, S_I_ID)

S_W_ID Foreign Key, references W_ID

S_I_ID Foreign Key, references I_ID

1.4 Implementation Rules

1.4.1 The physical clustering of records within the database is allowed.

1.4.2 A view which represents the rows to avoid logical read/writes is excluded.

Comment: The intent of this clause is to insure that the application implements the number of logical operations defined in the transaction profiles without combining several operations in one, via the use of a view.

1.4.3 All tables must have the properly scaled number of rows as defined by the database population requirements (see Clause 4.3).

1.4.4 Horizontal partitioning of tables is allowed. Groups of rows from a table may be assigned to different files, disks, or areas. If implemented, the details of such partitioning must be disclosed.

1.4.5 Vertical partitioning of tables is allowed. Groups of attributes (columns) of one table may be assigned to files, disks, or areas different from those storing the other attributes of that table. If implemented, the details of such partitioning must be disclosed (see Clause 1.4.9 for limitations).

Comment: in the two clauses above (1.4.4 and 1.4.5) assignment of data to different files, disks, or areas not based on knowledge of the logical structure of the data (e.g., knowledge of row or attribute boundaries) is not considered partitioning. For example, distribution or stripping over multiple disks of a physical file which stores one or more logical tables is not considered partitioning as long as this distribution is done by the hardware or the operating system without knowledge of the logical structure stored in the physical file.

1.4.6 Replication is allowed for all tables. All copies of tables which are replicated must meet all requirements for atomicity, consistency, and isolation as defined in Clause 3. If implemented, the details of such replication must be disclosed.

Comment: Only one copy of a replicated table needs to meet the durability requirements defined in Clause 3.

1.4.7 Attributes may be added and/or duplicated from one table to another as long as these changes do not improve performance.

1.4.8 Each attribute, as described in Clause 1.3.1, must be logically discrete and independently accessible by the data manager. For example, W_STREET_1 and W_STREET_2 cannot be implemented as two sub-parts of a discrete attribute W_STREET.

1.4.9 Each attribute, as described in Clause 1.3.1, must be accessible by the data manager as a single attribute. For example, S_DATA cannot be implemented as two discrete attributes S_DATA_1 and S_DATA_2. The following attributes are exceptions to this clause. No vertical partitioning can be defined between the two attributes used to implement these exceptions.

- All attributes holding a time-and-date value (i.e., C_SINCE, H_DATE, O_ENTRY_D, and OL_DELIVERY_D) can be implemented as a combination of two attributes: a date attribute and a time attribute.
- The attribute C_DATA can be implemented as two distinct attributes of equal size and using the same datatype.

1.4.10 The primary key of each table must not directly represent the physical disk addresses of the row or any offsets thereof. The application may not reference rows using relative addressing since they are simply offsets from the beginning of the storage space. This does not preclude hashing schemes or other file organizations which have provisions for adding, deleting, and modifying records in the ordinary course of processing. Exception: The History table can use relative addressing but all other requirements apply.

Comment 1: It is the intent of this clause that the application program (see Clause 2.1.7) executing the transaction, or submitting the transaction request, not use physical identifiers, but logical identifiers for all accesses, and contain no user written code which translates or aids in the translation of a logical key to the location within the table of the associated row or rows. For example, it is not legitimate for the application to build a "translation table" of logical-to-physical addresses and use it to enhance performance.

Comment 2: Internal record or row identifiers, for example, Tuple IDs or cursors, may be used under the following conditions:

1. For each transaction executed, initial access to any row must be via the key(s) specified in the transaction profile and no other attributes. Initial access includes insertion, deletion, retrieval, and update of any row.
2. Clause 1.4.10 may not be violated.

1.4.11 While inserts and deletes are not performed on all tables, the system must not be configured to take special advantage of this fact during the test. Although inserts are inherently limited by the storage space available on the configured system, there must be no restriction on inserting in any of the tables a minimum number of rows equal to 5% of the table cardinality and with a key value of at least double the range of key values present in that table.

Comment: It is required that the space for the additional 5% table cardinality be configured for the test run and priced (as static space per Clause 4.2.3) accordingly. For systems where space is configured and dynamically allocated at a later time, this space must be considered as allocated and included as static space when priced.

1.4.12 The minimum decimal precision for any computation performed as part of the application program must be the maximum decimal precision of all the individual items in that calculation.

1.4.13 Any other rules specified elsewhere in this document apply to the implementation (e.g., the consistency rules in Clause 3.3).

1.5 Integrity Rules

1.5.1 In any committed state, the primary key values must be unique within each table. For example, in the case of a horizontally partitioned table, primary key values of rows across all partitions must be unique.

1.5.2 In any committed state, no ill-formed rows may exist in the database. An ill-formed row occurs when the value of any attributes cannot be determined. For example, in the case of a vertically partitioned table, a row must exist in all the partitions.

1.6 Data Access Transparency Requirements

Data Access Transparency is the property of the system which removes from the application program any knowledge of the location and access mechanisms of partitioned data. An implementation which uses vertical and/or horizontal partitioning must meet the requirements for transparent data access described here.

No finite series of test 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.

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 below the application program such as the data/file manager (DBMS), the operating system, the hardware, or any combination of these.

1.6.1 Each of the nine tables described in Clause 1.3 must be identifiable by names which have no relationship to the partitioning of tables. All data manipulation operations in the application program (see Clause 2.1.7) must use only these names.

1.6.2 The system must prevent any data manipulation operation performed using the names described in Clause 1.6.1 which would result in a violation of the integrity rules (see Clause 1.5). For example: the system must prevent a non-TPC-C application from committing the insertion of a row in a vertically partitioned table unless all partitions of that row have been inserted.

1.6.3 Using the names which satisfy Clause 1.6.1, any arbitrary non-TPC-C application must be able to manipulate any set of rows or columns:

- Identifiable by any arbitrary condition supported by the underlying DBMS
- Using the names described in Clause 1.6.1 and using the same data manipulation semantics and syntax for all tables.

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

Comment: The intent is that the TPC-C application program uses general purpose mechanisms to manipulate data in the database.

Clause 2: TRANSACTION and TERMINAL PROFILES

2.1 Definition of Terms

2.1.1 The term **select** as used in this specification refers to the action of identifying (e.g., referencing, pointing to) a row (or rows) in the database without requiring retrieval of the actual content of the identified row(s).

2.1.2 The term **retrieve** as used in this specification refers to the action of accessing (i.e., fetching) the value of an attribute from the database and passing this value to the application program.

Note: Fields that correspond to database attributes are in UPPERCASE. Other fields, such as fields used by the SUT, or the RTE, for computations, or communication with the terminal, but not stored in the database, are in

+

2.1.3 The term **database transaction** as used in this specification refers to a unit of work on the database with full ACID properties as described in Clause 3. A **business transaction** is comprised of one or more database transactions. When used alone, the term transaction refers to a business transaction.

2.1.4 The term [..] represents a closed range of values starting with and ending with , .

2.1.5 The term **randomly selected within** [..] means independently selected at random and uniformly distributed between and , , inclusively, with a mean of (+,)/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.

2.1.6 The term **non-uniform random**, used only for generating customer numbers, customer last names, and item numbers, means an independently selected and non-uniformly distributed random number over the specified range of values [.. ,]. This number must be generated by using the function **NURand** which produces positions within the range [.. ,]. The results of NURand might have to be converted to produce a name or a number valid for the implementation.

$$\text{NURand}(A, x, y) = (((\text{random}(0, A) \mid \text{random}(x, y)) + C) \% (y - x + 1)) + x$$

where:

exp-1 | exp-2 stands for the bitwise logical OR operation between exp-1 and exp-2

exp-1 % exp-2 stands for exp-1 modulo exp-2

random(x, y) stands for randomly selected within [.. ,]

A is a constant chosen according to the size of the range [x .. y]

for C_LAST, the range is [0 .. 999] and A = 255

for C_ID, the range is [1 .. 3000] and A = 1023

for OL_I_ID, the range is [1 .. 100000] and A = 8191

C is a run-time constant randomly chosen within [0 .. A] that can be varied without altering performance. The same C value, per field (C_LAST, C_ID, and OL_I_ID), must be used by all emulated terminals.

2.1.6.1 In order that the value of C used for C_LAST does not alter performance the following must be true:

Let C-Load be the value of C used to generate C_LAST when populating the database. C-Load is a value in the range of [0..255] including 0 and 255.

Let C-Run be the value of C used to generate C_LAST for the measurement run.

Let C-Delta be the absolute value of the difference between C-Load and C-Run. C-Delta must be a value in the range of [65..119] including the values of 65 and 119 and excluding the value of 96 and 112.

2.1.7 The term **application program** refers to code that is not part of the commercially available components of the system, but produced specifically to implement the transaction profiles (see Clauses 2.4.2, 2.5.2, 2.6.2, 2.7.4, and 2.8.2) of this benchmark. For example, stored procedures, triggers, and referential integrity constraints are considered part of the application program when used to implement any portion of the transaction profiles, but are not considered part of the application program when solely used to enforce integrity rules (see Clause 1.5) or transparency requirements (see Clause 1.6) independently of any transaction profile.

2.1.8 The term **terminal** as used in this specification refers to the interface device capable of entering and displaying characters from and to a user with a minimum display of 24x80. A terminal is defined as the components that facilitate end-user input and the display of the output as defined in Clause 2. The terminal may not contain any knowledge of the application except field format, type, and position.

2.2 General Requirements for Terminal I/O

2.2.1 Input/Output Screen Definitions

2.2.1.1 The layout (position on the screen and size of titles and fields) of the input/output screens, as defined in Clauses 2.4.3.1, 2.5.3.1, 2.6.3.1, 2.7.3.1, and 2.8.3.1, must be reproduced by the test sponsor as closely as possible given the features and limitations of the implemented system. Any deviation from the input/output screens must be explained.

2.2.1.2 Input/output screens may be altered to circumvent limitations of the implementation providing that no performance advantage is gained. However, the following rules apply:

1. Titles can be translated into any language.
2. The semantic content cannot be altered.
3. The number of individual fields cannot be altered.
4. The number of characters within the fields (i.e., field width) cannot be decreased.
5. Reordering or repositioning of fields is allowed.
6. A copy of the new screen specifications and layout must be included in the Full Disclosure Report.

2.2.1.3 The amount and price fields defined in Clause 2 are formatted for U.S. currency. These formats can be modified to satisfy different currency representation (e.g., use another currency sign, move the decimal point retaining at least one digit on its right).

2.2.1.4 For input/output screens with unused fields (or groups of fields), it is not required to enter or display these fields. For example, when an order has less than 15 items, the groups of fields corresponding to the remaining items on the input/output screen are unused and need not be entered or displayed after being cleared. Similarly, when selecting a customer using its last name, the customer number field is unused and need not be entered or displayed after being cleared.

2.2.1.5 All input and output fields that may change must be cleared at the beginning of each transaction even when the same transaction type is consecutively selected by a given terminal. Fields should be cleared by displaying them as spaces or zeros.

Comment: In Clauses 2.2.1.4 and 2.2.1.5, if the test sponsor does not promote using space or zero as a clear character for its implementation, other clear characters can be used as long as a given field always uses the same clear character.

2.2.1.6 A **menu** is used to select the next transaction type. The menu, consisting of one or more lines, must be displayed at the very top or at the very bottom of the input/output screen. If an input field is needed to enter the menu selection, it must be located on the line(s) reserved for the menu.

Comment: The menu is in addition to the screen formats defined in the terminal I/O Clause for each transaction type.

2.2.1.7 The menu must display explicit text (i.e., it must contain the full name of each transaction and the action to be taken by the user to select each transaction). A minimum of 60 characters (excluding spaces) must be displayed on the menu.

2.2.1.8 Any input and output field(s), other than the mandatory fields specified in the input/output screens as defined in Clauses 2.4.3.1, 2.5.3.1, 2.6.3.1, 2.7.3.1, and 2.8.3.1, must be disclosed, and the purpose of such field(s) explained.

2.2.2 Entering and Displaying Fields

2.2.2.1 A field is said to be entered once all the significant characters that compose the input data for that field have been communicated to the SUT by the emulated terminal.

2.2.2.2 A field is said to be displayed once all significant characters that compose the data for that field have been communicated by the SUT to the emulated terminal for display.

2.2.2.3 Communicating input and output data does not require transferring any specific number of bytes. Methods for optimizing this communication, such as message compression and data caching, are allowed.

2.2.2.4 The following features must be provided to the emulated user:

1. The input characters appear on the input/output screen (i.e., are echoed) as they are keyed in. This requirement can be satisfied by visual inspection at full load where there are no perceivable delays. Otherwise, it is required that the character echoing be verified by actual measurements. For example, that can be done using a protocol analyzer, RTE measurement, etc. to show that the echo response time is less than 1 second. If local echo or block mode devices are used then verification is not required.

Comment: A web browser implementation, or a terminal or PC emulating a terminal in either local echo or block mode, will meet the echo response time requirement of one second, so there is no need for an echo test.

2. Input is allowed only in the positions of an input field (i.e., output-only fields, labels, and blanks spaces on the input/output screen are protected from input).
3. Input-capable fields are designated by some method of clearly identifying them (e.g., highlighted areas, underscores, reverse video, column dividers, etc.).
4. It must be possible to key in only significant characters into fields. For alphanumeric fields, non-keyed positions must be translated to blanks or nulls. For numeric fields, keyed input of less than the maximum allowable digits must be presented right justified on the output screen.
5. All fields for which a value is necessary to allow the application to complete are required to contain input prior to the start of the measurement of the transaction RT, or the application must contain a set of error-handling routines to inform the user that required fields have not been entered.
6. Fields can be keyed and re-keyed in any order. Specifically:
 - The emulated user must be able to move the input cursor forward and backward directly to the input capable fields.
 - The application cannot rely on fields being entered in any particular order.
 - The user can return to a field that has been keyed in and change its value prior to the start of the measurement of the transaction RT.
7. Numeric fields must be protected from non-numeric input. If one or more non-numeric characters is entered in a numeric field, a data entry error must be signaled to the user.

Comment: Input validation may either be performed by the terminal, by the application, or a combination of both. Input validation required by Item 5 and Item 7 must occur prior to starting a database transaction. Specifically, invalid data entry may not result in a rolled back transaction.

2.2.2.5 All output fields that display values that are updated in the database by the current business transaction must display the "new" (i.e., committed) values for those fields.

2.3 General Requirements for Transaction Profiles

Each transaction must be implemented according to the specified transaction profiles. In addition:

2.3.1 The order of the data manipulations within the transaction bounds is immaterial (unless otherwise specified, see Clause 2.4.2.3), and is left to the latitude of the test sponsor, as long as the implemented transactions are functionally equivalent to those specified in the transaction profiles.

2.3.2 The transaction profiles specify minimal data retrieval and update requirements for the transactions. Additional navigational steps or data manipulation operations implemented within the database transactions must be disclosed, and the purpose of such addition(s) must be explained.

2.3.3 Each attribute must be obtained from the designated table in the transaction profiles.

Comment: The intent of this clause is to prevent reducing the number of logical database operations required to implement each transaction.

2.3.4 No data manipulation operation from the transaction profile can be performed before all input data have been communicated to the SUT, or after any output data have been communicated by the SUT to the emulated terminal.

Comment: The intent of this clause is to ensure that, for a given business transaction, no data manipulation operation from the transaction profile is performed prior to the timestamp taken at the beginning of the Transaction RT or after the timestamp taken at the end of the Transaction RT (see Clause 5.3). For example, in the New-Order transaction the SUT is not allowed to fetch the matching row from the CUSTOMER table until all input data have been communicated to the SUT, even if this row is fetched again later during the execution of that same transaction.

2.3.5 If transactions are routed or organized within the SUT, a commercially available transaction processing monitor or equivalent commercially available software (hereinafter referred to as TM) is required with the following features/functionality:

Operation - The TM must allow for:

- request/service prioritization
- multiplexing/de multiplexing of requests/services
- automatic load balancing
- reception, queuing, and execution of multiple requests/services concurrently

Security - The TM must allow for:

- the ability to validate and authorize execution of each service at the time the service is requested.
- the restriction of administrative functions to authorized users.

Administration/Maintenance - The TM must have the predefined capability to perform centralized, non programmatic (i.e., must be implemented in the standard product and not require programming) and dynamic configuration management of TM resources including hardware, network, services (single or group), queue management prioritization rules, etc.

Recovery - The TM must have the capability to:

- post error codes to an application
- detect and terminate long-running transactions based on predefined time-out intervals

Application Transparency - The message context(s) that exist between the client and server application programs must be managed solely by the TM. The client and server application programs must not have any knowledge of the message context or the underlying communication mechanisms that support that context.

Comment 1: The following are examples of implementations that are non-compliant with the Application Transparency requirement.

1. Client and server application programs use the same identifier (e.g., handle or pointer) to maintain the message context for multiple transactions.
2. Change and/or recompilation of the client and/or server application programs is required when the number of queues or equivalent data structures used by the TM to maintain the message context between the client and server application programs is changed by TM administration.

Comment 2: The intent of this clause is to encourage the use of general purpose, commercially available transaction monitors, and to exclude special purpose software developed for benchmarking or other limited use. It is recognized that implementations of features and functionality described above vary across vendors' architectures. Such differences do not preclude compliance with the requirements of this clause.

Comment 3: Functionality of TM or equivalent software is not required if the DBMS maintains an individual context for each emulated user.

2.3.6 Any error that would result in an invalid TPC-C transaction must be detected and reported. An invalid TPC-C transaction includes transactions that, if committed, would violate the level of database consistency defined in Clause 3.3. These transactions must be rolled back. The detection of these invalid transactions must be reported to the user as part of the output screen or, in the case of the deferred portion of the delivery transaction, the delivery log.

Comment 1: Some examples of the types of errors which could result in an invalid transaction are:

- § Select or update of a non-existent record
- § Failure on insert of a new record
- § Failure to delete an existing record
- § Failure on select or update of an existing record

Comment 2: The exact information reported when an error occurs is implementation specific and not defined beyond the requirement that an error be reported.

2.4 The New-Order Transaction

The New-Order business transaction consists of entering a complete order through a single database transaction. It represents a mid-weight, read-write transaction with a high frequency of execution and stringent response time requirements to satisfy on-line users. This transaction is the backbone of the workload. It is designed to place a variable load on the system to reflect on-line database activity as typically found in production environments.

2.4.1 Input Data Generation

2.4.1.1 For any given terminal, the home warehouse number (W_ID) is constant over the whole measurement interval (see Clause 5.5).

2.4.1.2 The district number (D_ID) is randomly selected within [1 .. 10] from the home warehouse ($D_W_ID = W_ID$). The non-uniform random customer number (C_ID) is selected using the $NURand(1023,1,3000)$ function from the selected district number ($C_D_ID = D_ID$) and the home warehouse number ($C_W_ID = W_ID$).

2.4.1.3 The number of items in the order (-) is randomly selected within [5 .. 15] (an average of 10). This field is not entered. It is generated by the terminal emulator to determine the size of the order. O_OL_CNT is later displayed after being computed by the SUT.

2.4.1.4 A fixed 1% of the New-Order transactions are chosen at random to simulate user data entry errors and exercise the performance of rolling back update transactions. This must be implemented by generating a random number * within [1 .. 100].

Comment: All New-Order transactions must have independently generated input data. The input data from a rolled back transaction cannot be used for a subsequent transaction.

2.4.1.5 For each of the - items on the order:

1. A non-uniform random item number (OL_I_ID) is selected using the $NURand(8191,1,100000)$ function. If this is the last item on the order and * = 1 (see Clause 2.4.1.4), then the item number is set to an unused value.

Comment: An **unused** value for an item number is a value not found in the database such that its use will produce a "not-found" condition within the application program. This condition should result in rolling back the current database transaction.

2. A supplying warehouse number ($OL_SUPPLY_W_ID$) is selected as the home warehouse 99% of the time and as a remote warehouse 1% of the time. This can be implemented by generating a random number within [1 .. 100];

- If > 1 , the item is supplied from the home warehouse ($OL_SUPPLY_W_ID = W_ID$).

- If = 1, the item is supplied from a remote warehouse ($OL_SUPPLY_W_ID$ is randomly selected within the range of active warehouses (see Clause 4.2.2) other than W_ID).

Comment 1: With an average of 10 items per order, approximately 90% of all orders can be supplied in full by stocks from the home warehouse.

Comment 2: If the system is configured for a single warehouse, then all items are supplied from that single home warehouse.

3. A quantity ($OL_QUANTITY$) is randomly selected within [1 .. 10].

2.4.1.6 The order entry date (O_ENTRY_D) is generated within the SUT by using the current system date and time.

2.4.1.7 An order-line is said to be **home** if it is supplied by the home warehouse (i.e., when OL_SUPPLY_W_ID equals O_W_ID).

2.4.1.8 An order-line is said to be **remote** when it is supplied by a remote warehouse (i.e., when OL_SUPPLY_W_ID does not equal O_W_ID).

2.4.2 Transaction Profile

2.4.2.1 Entering a new order is done in a single database transaction with the following steps:

1. Create an order header, comprised of:
 - 2 row selections with data retrieval,
 - 1 row selection with data retrieval and update,
 - 2 row insertions.
2. Order a variable number of items (average - = 10), comprised of:
 - (1 * -) row selections with data retrieval,
 - (1 * -) row selections with data retrieval and update,
 - (1 * -) row insertions.

Note: The above summary is provided for information only. The actual requirement is defined by the detailed transaction profile below.

2.4.2.2 For a given warehouse number (W_ID), district number (D_W_ID , D_ID), customer number (C_W_ID , C_D_ID , C_ID), count of items (- , not communicated to the SUT), and for a given set of items (OL_I_ID), supplying warehouses (OL_SUPPLY_W_ID), and quantities (OL_QUANTITY):

- The input data (see Clause 2.4.3.2) are communicated to the SUT.
- A database transaction is started.
- The row in the WAREHOUSE table with matching W_ID is selected and W_TAX, the warehouse tax rate, is retrieved.
- The row in the DISTRICT table with matching D_W_ID and D_ID is selected, D_TAX, the district tax rate, is retrieved, and D_NEXT_O_ID, the next available order number for the district, is retrieved and incremented by one.
- The row in the CUSTOMER table with matching C_W_ID, C_D_ID, and C_ID is selected and C_DISCOUNT, the customer's discount rate, C_LAST, the customer's last name, and C_CREDIT, the customer's credit status, are retrieved.
- A new row is inserted into both the NEW-ORDER table and the ORDER table to reflect the creation of the new order. O_CARRIER_ID is set to a null value. If the order includes only home order-lines, then O_ALL_LOCAL is set to 1, otherwise O_ALL_LOCAL is set to 0.
- The number of items, O_OL_CNT, is computed to match - +

- For each O_OL_CNT item on the order:
 - The row in the ITEM table with matching I_ID (equals OL_I_ID) is selected and I_PRICE, the price of the item, I_NAME, the name of the item, and I_DATA are retrieved. If I_ID has an unused value (see Clause 2.4.1.5), a "not-found" condition is signaled, resulting in a rollback of the database transaction (see Clause 2.4.2.3).
 - The row in the STOCK table with matching S_I_ID (equals OL_I_ID) and S_W_ID (equals OL_SUPPLY_W_ID) is selected. S_QUANTITY, the quantity in stock, S_DIST_xx, where xx represents the district number, and S_DATA are retrieved. If the retrieved value for S_QUANTITY exceeds OL_QUANTITY by 10 or more, then S_QUANTITY is decreased by OL_QUANTITY; otherwise S_QUANTITY is updated to (S_QUANTITY - OL_QUANTITY)+91. S_YTD is increased by OL_QUANTITY and S_ORDER_CNT is incremented by 1. If the order-line is remote, then S_REMOTE_CNT is incremented by 1.
 - The amount for the item in the order (OL_AMOUNT) is computed as:

$$OL_QUANTITY * I_PRICE$$
 - The strings in I_DATA and S_DATA are examined. If they both include the string "ORIGINAL", the & field for that item is set to "B", otherwise, the & field is set to "G".
 - A new row is inserted into the ORDER-LINE table to reflect the item on the order. OL_DELIVERY_D is set to a null value, OL_NUMBER is set to a unique value within all the ORDER-LINE rows that have the same OL_O_ID value, and OL_DIST_INFO is set to the content of S_DIST_xx, where xx represents the district number (OL_D_ID)
- The & for the complete order is computed as:

$$\text{sum}(OL_AMOUNT) * (1 - C_DISCOUNT) * (1 + W_TAX + D_TAX)$$
- The database transaction is committed, unless it has been rolled back as a result of an & value for the last item number (see Clause 2.4.1.5).
- The output data (see Clause 2.4.3.3) are communicated to the terminal.

2.4.2.3 For transactions that rollback as a result of an unused item number, the complete transaction profile must be executed with the exception that the following steps need not be done:

- Selecting and retrieving the row in the STOCK table with S_I_ID matching the unused item number.
- Examining the strings I_DATA and S_DATA for the unused item.
- Inserting a new row into the ORDER-LINE table for the unused item.
- Adding the amount for the unused item to the sum of all OL_AMOUNT.

The transaction is not committed. Instead, the transaction is rolled back.

Comment 1: The intent of this clause is to ensure that within the New-Order transaction all valid items are processed prior to processing the unused item. Knowledge that an item is unused, resulting in rolling back the transaction, can only be used to skip execution of the above steps. No other optimization can result from this knowledge (e.g., skipping other steps, changing the execution of other steps, using a different type of transaction, etc.).

Comment 2: This clause is an exception to Clause 2.3.1. The order of data manipulations prior to signaling a "not found" condition is immaterial.

2.4.3 Terminal I/O

2.4.3.1 For each transaction the originating terminal must display the following input/output screen with all input and output fields cleared (with either spaces or zeros) except for the Warehouse field which has not changed and must display the fixed W_ID value associated with that terminal.

```

1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
1
2                               New Order
3 Warehouse: 9999   District: 99                               Date: DD-MM-YYYY hh:mm:ss
4 Customer:  9999   Name: XXXXXXXXXXXXXXXXXXXX   Credit: XX   %Disc: 99.99
5 Order Number: 99999999   Number of Lines: 99           W_tax: 99.99   D_tax: 99.99
6
7 Supp_W  Item_Id  Item Name                               Qty  Stock  B/G  Price  Amount
8 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
9 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
10 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
11 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
12 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
13 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
14 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
15 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
16 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
17 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
18 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
19 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
20 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
21 9999   999999   XXXXXXXXXXXXXXXXXXXXXXXXXXXX           99   999    X   $999.99 $9999.99
22 Execution Status: XXXXXXXXXXXXXXXXXXXXXXXXXXXX                               Total: $99999.99
23
24

```

2.4.3.2 The emulated user must enter, in the appropriate fields of the input/output screen, the required input data which is divided in two groups and organized as follows:

- Two fields: D_ID and C_ID.
Comment: The value for - cannot be entered, but must be determined by the application upon processing of the input data.
- One repeating group of fields: OL_I_ID, OL_SUPPLY_W_ID and OL_QUANTITY. The group is repeated - times (once per item in the order). The values of these fields are chosen as per Clause 2.4.1.5.
Comment: In order to maintain a reasonable amount of keyed input, the supply warehouse fields must be filled in for each item, even when the supply warehouse is the home warehouse.

2.4.3.3 The emulated terminal must display, in the appropriate fields of the input/output screen, all input data and the output data resulting from the execution of the transaction. The display fields are divided in two groups as follows:

- One non-repeating group of fields: W_ID, D_ID, C_ID, O_ID, O_OL_CNT, C_LAST, C_CREDIT, C_DISCOUNT, W_TAX, D_TAX, O_ENTRY_D, - , and an optional execution status message other than "Item number is not valid".

- One repeating group of fields: OL_SUPPLY_W_ID, OL_I_ID, I_NAME, OL_QUANTITY, S_QUANTITY, I_PRICE, and OL_AMOUNT. The group is repeated O_OL_CNT times (once per item in the order), equal to the computed value of - +

2.4.3.4 For transactions that are rolled back as a result of an unused item number (1% of all New-Order transactions), the emulated terminal must display in the appropriate fields of the input/output screen the fields: W_ID, D_ID, C_ID, C_LAST, C_CREDIT, O_ID, and the execution status message "Item number is not valid". Note that no execution status message is required for successfully committed transactions. However, this field may not display "Item number is not valid" if the transaction is successful.

Comment: The number of the rolled back order, O_ID, must be displayed to verify that part of the transaction was processed.

2.4.3.5 The following table summarizes the terminal I/O requirements for the New-Order transaction:

	Enter	Display After rollback	Display Row/Column	Coordinates
Non-repeating Group		W_ID	W_ID	2/12
	D_ID	D_ID	D_ID	2/29
	C_ID	C_ID	C_ID	3/12
		C_LAST	C_LAST	3/25
		C_CREDIT	C_CREDIT	3/52
		C_DISCOUNT		3/64
		W_TAX		4/51
		D_TAX		4/67
		O_OL_CNT		4/42
		O_ID	O_ID	4/15
		O_ENTRY_D		2/61
		&		22/71
			"Item number is not valid"	22/19
Repeating Group	OL_SUPPLY_W_ID	OL_SUPPLY_W_ID		7-22/3
	OL_I_ID	OL_I_ID		7-22/10
		I_NAME		7-22/20
	OL_QUANTITY	OL_QUANTITY		7-22/45
		S_QUANTITY		7-22/51
		&		7-22/58
		I_PRICE		7-22/63
		OL_AMOUNT		7-22/72

2.4.3.6 For general terminal I/O requirements, see Clause 2.2.

2.5 The Payment Transaction

The Payment business transaction updates the customer's balance and reflects the payment on the district and warehouse sales statistics. It represents a light-weight, read-write transaction with a high frequency of execution and stringent response time requirements to satisfy on-line users. In addition, this transaction includes non-primary key access to the CUSTOMER table.

2.5.1 Input Data Generation

2.5.1.1 For any given terminal, the home warehouse number (W_ID) is constant over the whole measurement interval.

2.5.1.2 The district number (D_ID) is randomly selected within [1 ..10] from the home warehouse (D_W_ID) = W_ID). The customer is randomly selected 60% of the time by last name (C_W_ID , C_D_ID, C_LAST) and 40% of the time by number (C_W_ID , C_D_ID , C_ID). Independent of the mode of selection, the customer resident warehouse is the home warehouse 85% of the time and is a randomly selected remote warehouse 15% of the time. This can be implemented by generating two random numbers r_1 and r_2 , within [1 .. 100];

- If $r_1 \leq 85$ a customer is selected from the selected district number (C_D_ID = D_ID) and the home warehouse number (C_W_ID = W_ID). The customer is paying through his/her own warehouse.
- If $r_1 > 85$ a customer is selected from a random district number (C_D_ID is randomly selected within [1 .. 10]), and a random remote warehouse number (C_W_ID is randomly selected within the range of active warehouses (see Clause 4.2.2), and $C_W_ID \neq W_ID$). The customer is paying through a warehouse and a district other than his/her own.
- If $r_2 \leq 60$ a customer last name (C_LAST) is generated according to Clause 4.3.2.3 from a non-uniform random value using the NURand(255,0,999) function. The customer is using his/her last name and is one of the possibly several customers with that last name.

Comment: This case illustrates the situation when a customer does not use his/her unique customer number.

- If $r_2 > 60$ a non-uniform random customer number (C_ID) is selected using the NURand(1023,1,3000) function. The customer is using his/her customer number.

Comment: If the system is configured for a single warehouse, then all customers are selected from that single home warehouse.

2.5.1.3 The payment amount (H_AMOUNT) is randomly selected within [1.00 .. 5,000.00].

2.5.1.4 The payment date (H_DATE) is generated within the SUT by using the current system date and time.

2.5.1.5 A Payment transaction is said to be **home** if the customer belongs to the warehouse from which the payment is entered (when C_W_ID = W_ID).

2.5.1.6 A Payment transaction is said to be **remote** if the warehouse from which the payment is entered is not the one to which the customer belongs (when C_W_ID does not equal W_ID).

2.5.2 Transaction Profile

2.5.2.1 The Payment transaction enters a customer's payment with a single database transaction and is comprised of:

Case 1, the customer is selected based on customer number:

- 3 row selections with data retrieval and update,
- 1 row insertion.

Case 2, the customer is selected based on customer last name:

- 2 row selections (on average) with data retrieval,
- 3 row selections with data retrieval and update,
- 1 row insertion.

Note: The above summary is provided for information only. The actual requirement is defined by the detailed transaction profile below.

2.5.2.2 For a given warehouse number (W_ID), district number (D_W_ID , D_ID), customer number (C_W_ID , C_D_ID , C_ID) or customer last name (C_W_ID , C_D_ID , C_LAST), and payment amount (H_AMOUNT):

- The input data (see Clause 2.5.3.2) are communicated to the SUT.
- A database transaction is started.
- The row in the WAREHOUSE table with matching W_ID is selected. W_NAME , W_STREET_1 , W_STREET_2 , W_CITY , W_STATE , and W_ZIP are retrieved and W_YTD , the warehouse's year-to-date balance, is increased by H_AMOUNT .
- The row in the DISTRICT table with matching D_W_ID and D_ID is selected. D_NAME , D_STREET_1 , D_STREET_2 , D_CITY , D_STATE , and D_ZIP are retrieved and D_YTD , the district's year-to-date balance, is increased by H_AMOUNT .
- **Case 1**, the customer is selected based on customer number: the row in the CUSTOMER table with matching C_W_ID , C_D_ID and C_ID is selected. C_FIRST , C_MIDDLE , C_LAST , C_STREET_1 , C_STREET_2 , C_CITY , C_STATE , C_ZIP , C_PHONE , C_SINCE , C_CREDIT , C_CREDIT_LIM , $C_DISCOUNT$, and $C_BALANCE$ are retrieved. $C_BALANCE$ is decreased by H_AMOUNT . $C_YTD_PAYMENT$ is increased by H_AMOUNT . $C_PAYMENT_CNT$ is incremented by 1.

Case 2, the customer is selected based on customer last name: all rows in the CUSTOMER table with matching C_W_ID , C_D_ID and C_LAST are selected sorted by C_FIRST in ascending order. Let n be the number of rows selected. C_ID , C_FIRST , C_MIDDLE , C_STREET_1 , C_STREET_2 , C_CITY , C_STATE , C_ZIP , C_PHONE , C_SINCE , C_CREDIT , C_CREDIT_LIM , $C_DISCOUNT$, and $C_BALANCE$ are retrieved from the row at position ($n/2$ rounded up to the next integer) in the sorted set of selected rows from the CUSTOMER table. $C_BALANCE$ is decreased by H_AMOUNT . $C_YTD_PAYMENT$ is increased by H_AMOUNT . $C_PAYMENT_CNT$ is incremented by 1.

- If the value of C_CREDIT is equal to "BC", then C_DATA is also retrieved from the selected customer and the following history information: C_ID , C_D_ID , C_W_ID , D_ID , W_ID , and H_AMOUNT , are inserted at the left of the C_DATA field by shifting the existing content of C_DATA to the right by an equal number of bytes and by discarding the bytes that are shifted out of the right side of the C_DATA field. The content of the C_DATA field never exceeds 500 characters. The selected customer is updated with the new C_DATA field. If C_DATA is implemented as two fields (see Clause 1.4.9), they must be treated and operated on as one single field.

Comment: The format used to store the history information must be such that its display on the input/output screen is in a readable format. (e.g. the W_ID portion of C_DATA must use the same display format as the output field W_ID).

- H_DATA is built by concatenating W_NAME and D_NAME separated by 4 spaces.
- A new row is inserted into the HISTORY table with H_C_ID = C_ID, H_C_D_ID = C_D_ID, H_C_W_ID = C_W_ID, H_D_ID = D_ID, and H_W_ID = W_ID.
- The database transaction is committed.
- The output data (see Clause 2.5.3.3) are communicated to the terminal.

2.5.3 Terminal I/O

2.5.3.1 For each transaction the originating terminal must display the following input/output screen with all input and output fields cleared (with either spaces or zeros) except for the Warehouse field which has not changed and must display the fixed W_ID value associated with that terminal. In addition, all address fields (i.e., W_STREET_1, W_STREET_2, W_CITY, W_STATE, and W_ZIP) of the warehouse may display the fixed values for these fields if these values were already retrieved in a previous transaction.

```

1                                     2                                     3                                     4                                     5                                     6                                     7                                     8
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
1                                     Payment
2 Date: DD-MM-YYYY hh:mm:ss
3
4 Warehouse: 9999                               District: 99
5 XXXXXXXXXXXXXXXXXXXXXXXXXXXX                XXXXXXXXXXXXXXXXXXXXXXXXXXXX
6 XXXXXXXXXXXXXXXXXXXXXXXXXXXX                XXXXXXXXXXXXXXXXXXXXXXXXXXXX
7 XXXXXXXXXXXXXXXXXXXXXXXXXXXX XX XXXXX-XXXX  XXXXXXXXXXXXXXXXXXXXXXXXXXXX XX XXXXX-XXXX
8
9 Customer: 9999  Cust-Warehouse: 9999  Cust-District: 99
10 Name:   XXXXXXXXXXXXXXXXXXXX XX XXXXXXXXXXXXXXXXXXXX          Since: DD-MM-YYYY
11          XXXXXXXXXXXXXXXXXXXX                                Credit: XX
12          XXXXXXXXXXXXXXXXXXXX                                %Disc: 99.99
13          XXXXXXXXXXXXXXXXXXXX XX XXXXX-XXXX                Phone: XXXXXX-XXX-XXX-XXXX
14
15 Amount Paid:           $9999.99           New Cust-Balance: $-9999999999.99
16 Credit Limit:    $9999999999.99
17
18 Cust-Data: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
19              XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
20              XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
21              XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
22
23
24

```

2.5.3.2 The emulated user must enter, in the appropriate fields of the input/output screen, the required input data which is organized as the distinct fields: D_ID, C_ID or C_LAST, C_D_ID, C_W_ID, and H_AMOUNT.

Comment: In order to maintain a reasonable amount of keyed input, the customer warehouse field must be filled in even when it is the same as the home warehouse.

2.5.3.3 The emulated terminal must display, in the appropriate fields of the input/output screen, all input data and the output data resulting from the execution of the transaction. The following fields are displayed: W_ID, D_ID, C_ID, C_D_ID, C_W_ID, W_STREET_1, W_STREET_2, W_CITY, W_STATE, W_ZIP, D_STREET_1, D_STREET_2, D_CITY, D_STATE, D_ZIP, C_FIRST, C_MIDDLE, C_LAST, C_STREET_1, C_STREET_2, C_CITY, C_STATE, C_ZIP, C_PHONE, C_SINCE, C_CREDIT, C_CREDIT_LIM, C_DISCOUNT, C_BALANCE, the first 200 characters of C_DATA (only if C_CREDIT = "BC"), H_AMOUNT, and H_DATE.

2.5.3.4 The following table summarizes the terminal I/O requirements for the Payment transaction:

Enter	Display Row/Column	Coordinates
Non-repeating Group	W_ID	4/12
D_ID	D_ID	4/52
C_ID ¹	C_ID	9/11
C_D_ID	C_D_ID	9/54
C_W_ID	C_W_ID	9/33
H_AMOUNT	H_AMOUNT	15/24
	H_DATE	2/7
	W_STREET_1	5/1
	W_STREET_2	6/1
	W_CITY	7/1
	W_STATE	7/22
	W_ZIP	7/25
	D_STREET_1	5/42
	D_STREET_2	6/42
	D_CITY	7/42
	D_STATE	7/63
	D_ZIP	7/66
	C_FIRST	10/9
	C_MIDDLE	10/26
C_LAST ²	C_LAST	10/29
	C_STREET_1	11/9
	C_STREET_2	12/9
	C_CITY	13/9
	C_STATE	13/30
	C_ZIP	13/33
	C_PHONE	13/58
	C_SINCE	10/58
	C_CREDIT	11/58
	C_CREDIT_LIM	16/18
	C_DISCOUNT	12/58
	C_BALANCE	15/56
	C_DATA ³	18-21/12

¹ Enter only for payment by customer number

2

Enter only for payment by customer last name

3

Display the first 200 characters only if C_CREDIT = "BC"

2.5.3.5 For general terminal I/O requirements, see Clause 2.2.

2.6 The Order-Status Transaction

The Order-Status business transaction queries the status of a customer's last order. It represents a mid-weight read-only database transaction with a low frequency of execution and response time requirement to satisfy on-line users. In addition, this table includes non-primary key access to the CUSTOMER table.

2.6.1 Input Data Generation

2.6.1.1 For any given terminal, the home warehouse number (W_ID) is constant over the whole measurement interval.

2.6.1.2 The district number (D_ID) is randomly selected within [1 ..10] from the home warehouse. The customer is randomly selected 60% of the time by last name (C_W_ID, C_D_ID, C_LAST) and 40% of the time by number (C_W_ID, C_D_ID, C_ID) from the selected district (C_D_ID = D_ID) and the home warehouse number (C_W_ID = W_ID). This can be implemented by generating a random number , within [1 .. 100];

- If , ≤ 60 a customer last name (C_LAST) is generated according to Clause 4.3.2.3 from a non-uniform random value using the NURand(255,0,999) function. The customer is using his/her last name and is one of the, possibly several, customers with that last name.

Comment: This case illustrates the situation when a customer does not use his/her unique customer number.

- If , > 60 a non-uniform random customer number (C_ID) is selected using the NURand(1023,1,3000) function. The customer is using his/her customer number.

2.6.2 Transaction Profile

2.6.2.1 Querying for the status of an order is done in a single database transaction with the following steps:

1. Find the customer and his/her last order, comprised of:

Case 1, the customer is selected based on customer number:

2 row selections with data retrieval.

Case 2, the customer is selected based on customer last name:

4 row selections (on average) with data retrieval.

2. Check status (delivery date) of each item on the order (average items-per-order = 10), comprised of:
(1 * items-per-order) row selections with data retrieval.

Note: The above summary is provided for information only. The actual requirement is defined by the detailed transaction profile below.

2.6.2.2 For a given customer number (C_W_ID , C_D_ID , C_ID):

- The input data (see Clause 2.6.3.2) are communicated to the SUT.
- A database transaction is started.
- **Case 1**, the customer is selected based on customer number: the row in the CUSTOMER table with matching C_W_ID, C_D_ID, and C_ID is selected and C_BALANCE, C_FIRST, C_MIDDLE, and C_LAST are retrieved.

Case 2, the customer is selected based on customer last name: all rows in the CUSTOMER table with matching C_W_ID, C_D_ID and C_LAST are selected sorted by C_FIRST in ascending order. Let n be the number of rows selected. C_BALANCE, C_FIRST, C_MIDDLE, and C_LAST are retrieved from the row at position $\lceil n/2 \rceil$ rounded up in the sorted set of selected rows from the CUSTOMER table.

- The row in the ORDER table with matching O_W_ID (equals C_W_ID), O_D_ID (equals C_D_ID), O_C_ID (equals C_ID), and with the largest existing O_ID, is selected. This is the most recent order placed by that customer. O_ID, O_ENTRY_D, and O_CARRIER_ID are retrieved.
- All rows in the ORDER-LINE table with matching OL_W_ID (equals O_W_ID), OL_D_ID (equals O_D_ID), and OL_O_ID (equals O_ID) are selected and the corresponding sets of OL_I_ID, OL_SUPPLY_W_ID, OL_QUANTITY, OL_AMOUNT, and OL_DELIVERY_D are retrieved.
- The database transaction is committed.

Comment: a commit is not required as long as all ACID properties are satisfied (see Clause 3).

- The output data (see Clause 2.6.3.3) are communicated to the terminal.

2.6.3 Terminal I/O

2.6.3.1 For each transaction the originating terminal must display the following input/output screen with all input and output fields cleared (with either spaces or zeros) except for the Warehouse field which has not changed and must display the fixed W_ID value associated with that terminal.

```

1                                     2                                     3                                     4                                     5                                     6                                     7                                     8
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
1                                     Order-Status
2 Warehouse: 9999 District: 99
3 Customer: 9999 Name: XXXXXXXXXXXXXXXXXXXX XX XXXXXXXXXXXXXXXXXXXX
4 Cust-Balance: $-99999.99
5
6 Order-Number: 99999999 Entry-Date: DD-MM-YYYY hh:mm:ss Carrier-Number: 99
7 Supply-W      Item-Id    Qty      Amount      Delivery-Date
8 9999          999999    99       $99999.99   DD-MM-YYYY
9 9999          999999    99       $99999.99   DD-MM-YYYY
10 9999          999999    99       $99999.99   DD-MM-YYYY
11 9999          999999    99       $99999.99   DD-MM-YYYY
12 9999          999999    99       $99999.99   DD-MM-YYYY
13 9999          999999    99       $99999.99   DD-MM-YYYY
14 9999          999999    99       $99999.99   DD-MM-YYYY
15 9999          999999    99       $99999.99   DD-MM-YYYY
16 9999          999999    99       $99999.99   DD-MM-YYYY
17 9999          999999    99       $99999.99   DD-MM-YYYY
18 9999          999999    99       $99999.99   DD-MM-YYYY
19 9999          999999    99       $99999.99   DD-MM-YYYY
20 9999          999999    99       $99999.99   DD-MM-YYYY
21 9999          999999    99       $99999.99   DD-MM-YYYY
22 9999          999999    99       $99999.99   DD-MM-YYYY
23
24

```

2.6.3.2 The emulated user must enter, in the appropriate field of the input/output screen, the required input data which is organized as the distinct fields: D_ID and either C_ID or C_LAST.

2.6.3.3 The emulated terminal must display, in the appropriate fields of the input/output screen, all input data and the output data resulting from the execution of the transaction. The display fields are divided in two groups as follows:

- One non-repeating group of fields: W_ID, D_ID, C_ID, C_FIRST, C_MIDDLE, C_LAST, C_BALANCE, O_ID, O_ENTRY_D, and O_CARRIER_ID;
- One repeating group of fields: OL_SUPPLY_W_ID, OL_I_ID, OL_QUANTITY, OL_AMOUNT, and OL_DELIVERY_D. The group is repeated O_OL_CNT times (once per item in the order).

Comment 1: The order of items shown on the Order-Status screen does not need to match the order in which the items were entered in its corresponding New-Order screen.

Comment 2: If OL_DELIVERY_D is null (i.e., the order has not been delivered), the terminal must display an implementation specific null date representation (e.g., blanks, 99-99-9999, etc.). The chosen null date representation must not change during the test.

2.6.3.4 The following table summarizes the terminal I/O requirements for the Order-Status transaction:

	Enter	Display Row/Column	Coordinates
Non-repeating Group		W_ID	2/12
	D_ID	D_ID	2/29
	C_ID ¹	C_ID	3/11
		C_FIRST	3/24
		C_MIDDLE	3/41
	C_LAST ²	C_LAST	3/44
		C_BALANCE	4/16
		O_ID	6/15
		O_ENTRY_D	6/38
		O_CARRIER_ID	6/76
Repeating Group		OL_SUPPLY_W_ID	8-22/3
		OL_I_ID	8-22/14
		OL_QUANTITY	8-22/25
		OL_AMOUNT	8-22/33
		OL_DELIVERY_D	8-22/47

¹ Enter only for query by customer number.
Enter only for query by customer last name.

2

2.6.3.5 For general terminal I/O requirements, see Clause 2.2.

2.7 The Delivery Transaction

The Delivery business transaction consists of processing a batch of 10 new (not yet delivered) orders. Each order is processed (delivered) in full within the scope of a read-write database transaction. The number of orders delivered as a group (or batched) within the same database transaction is implementation specific. The business transaction, comprised of one or more (up to 10) database transactions, has a low frequency of execution and must complete within a relaxed response time requirement.

The Delivery transaction is intended to be executed in deferred mode through a queuing mechanism, rather than interactively, with terminal response indicating transaction completion. The result of the deferred execution is recorded into a result file.

2.7.1 Input Data Generation

2.7.1.1 For any given terminal, the home warehouse number (W_ID) is constant over the whole measurement interval.

2.7.1.2 The carrier number (O_CARRIER_ID) is randomly selected within [1 .. 10].

2.7.1.3 The delivery date (OL_DELIVERY_D) is generated within the SUT by using the current system date and time.

2.7.2 Deferred Execution

2.7.2.1 Unlike the other transactions in this benchmark, the Delivery transaction must be executed in deferred mode. This mode of execution is primarily characterized by queuing the transaction for deferred execution, returning control to the originating terminal independently from the completion of the transaction, and recording execution information into a result file.

2.7.2.2 Deferred execution of the Delivery transaction must adhere to the following rules:

1. The business transaction is queued for deferred execution as a result of entering the last input character.
2. The deferred execution of the business transaction must follow the profile defined in Clause 2.7.4 with the input data defined in Clause 2.7.1 as entered through the input/output screen and communicated to the deferred execution queue.
3. At least 90% of the business transactions must complete within 80 seconds of their being queued for execution.
4. Upon completion of the business transaction, the following information must have been recorded into a result file:
 - The time at which the business transaction was queued.
 - The warehouse number (W_ID) and the carrier number (O_CARRIER_ID) associated with the business transaction.
 - The district number (D_ID) and the order number (O_ID) of each order delivered by the business transaction.
 - The time at which the business transaction completed.

2.7.2.3 The **result file** associated with the deferred execution of the Delivery business transaction is only for the purpose of recording information about that transaction and is not relevant to the business function being performed. The result file must adhere to the following rules:

1. All events must be completed before the related information is recorded (e.g., the recording of a district and order number must be done after the database transaction, within which this order was delivered, has been committed);
2. No ACID property is required (e.g., the recording of a district and order number is not required to be atomic with the actual delivery of that order) as the result file is used for benchmarking purposes only.
3. During the measurement interval the result file must be located either on a durable medium (see clause 3.5.1) or in the internal memory of the SUT. In this last case, the result file must be transferred onto a durable medium after the last measurement interval of the test run (see Clause 5.5).

2.7.3 Terminal I/O

2.7.3.1 For each transaction the originating terminal must display the following input/output screen with all input and output fields cleared (with either spaces or zeros) except for the Warehouse field which has not changed and must display the fixed W_ID value associated with that terminal.

```
1          2          3          4          5          6          7          8
1234567890123456789012345678901234567890123456789012345678901234567890
1          Delivery
2 Warehouse: 9999
3
4 Carrier Number: 99
5
6 Execution Status: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

2.7.3.2 The emulated user must enter, in the appropriate input field of the input/output screen, the required input data which is organized as one distinct field: O_CARRIER_ID.

2.7.3.3 The emulated terminal must display, in the appropriate output field of the input/output screen, all input data and the output data which results from the queuing of the transaction. The following fields are displayed: W_ID, O_CARRIER_ID, and the status message "Delivery has been queued".

2.7.3.4 The following table summarizes the terminal I/O requirements for the Delivery transaction:

	Enter	Display Row/Column	Coordinates
Non-repeating Group		W_ID	2/12
	O_CARRIER_ID	O_CARRIER_ID	4/17
		"Delivery has been queued"	6/19

2.7.3.5 For general terminal I/O requirements, see Clause 2.2.

2.7.4 Transaction Profile

2.7.4.1 The deferred execution of the Delivery transaction delivers one outstanding order (average items-per-order = 10) for each one of the 10 districts of the selected warehouse using one or more (up to 10) database transactions. Delivering each order is done in the following steps:

1. Process the order, comprised of:
 - 1 row selection with data retrieval,
 - (1 + items-per-order) row selections with data retrieval and update.
2. Update the customer's balance, comprised of:
 - 1 row selections with data update.
3. Remove the order from the new-order list, comprised of:
 - 1 row deletion.

Comment: This business transaction can be done within a single database transaction or broken down into up to 10 database transactions to allow the test sponsor the flexibility to implement the business transaction with the most efficient number of database transactions.

Note: The above summary is provided for information only. The actual requirement is defined by the detailed transaction profile below.

2.7.4.2 For a given warehouse number (W_ID), for each of the 10 districts_(D_W_ID , D_ID) within that warehouse, and for a given carrier number (O_CARRIER_ID):

- The input data (see Clause 2.7.3.2) are retrieved from the deferred execution queue.
- A database transaction is started unless a database transaction is already active from being started as part of the delivery of a previous order (i.e., more than one order is delivered within the same database transaction).
- The row in the NEW-ORDER table with matching NO_W_ID (equals W_ID) and NO_D_ID (equals D_ID) and with the lowest NO_O_ID value is selected. This is the oldest undelivered order of that district. NO_O_ID, the order number, is retrieved. If no matching row is found, then the delivery of an order for this district is skipped. The condition in which no outstanding order is present at a given district must be handled by skipping the delivery of an order for that district only and resuming the delivery of an order from all remaining districts of the selected warehouse. If this condition occurs in more than 1%, or in more than one, whichever is greater, of the business transactions, it must be reported. The result file must be organized in such a way that the percentage of skipped deliveries and skipped districts can be determined.

- The selected row in the NEW-ORDER table is deleted.
- The row in the ORDER table with matching O_W_ID (equals W_ID), O_D_ID (equals D_ID), and O_ID (equals NO_O_ID) is selected, O_C_ID, the customer number, is retrieved, and O_CARRIER_ID is updated.
- All rows in the ORDER-LINE table with matching OL_W_ID (equals O_W_ID), OL_D_ID (equals O_D_ID), and OL_O_ID (equals O_ID) are selected. All OL_DELIVERY_D, the delivery dates, are updated to the current system time as returned by the operating system and the sum of all OL_AMOUNT is retrieved.
- The row in the CUSTOMER table with matching C_W_ID (equals W_ID), C_D_ID (equals D_ID), and C_ID (equals O_C_ID) is selected and C_BALANCE is increased by the sum of all order-line amounts (OL_AMOUNT) previously retrieved. C_DELIVERY_CNT is incremented by 1.
- The database transaction is committed unless more orders will be delivered within this database transaction.
- Information about the delivered order (see Clause 2.7.2.2) is recorded into the result file (see Clause 2.7.2.3).

2.8 The Stock-Level Transaction

The Stock-Level business transaction determines the number of recently sold items that have a stock level below a specified threshold. It represents a heavy read-only database transaction with a low frequency of execution, a relaxed response time requirement, and relaxed consistency requirements.

2.8.1 Input Data Generation

2.8.1.1 Each terminal must use a unique value of (W_ID, D_ID) that is constant over the whole measurement, i.e., D_IDs cannot be re-used within a warehouse.

2.8.1.2 The threshold of minimum quantity in stock (threshold) is selected at random within [10 .. 20].

2.8.2 Transaction Profile

2.8.2.1 Examining the level of stock for items on the last 20 orders is done in one or more database transactions with the following steps:

1. Examine the next available order number, comprised of:
1 row selection with data retrieval.
2. Examine all items on the last 20 orders (average items-per-order = 10) for the district, comprised of:
(20 * items-per-order) row selections with data retrieval.
3. Examine, for each distinct item selected, if the level of stock available at the home warehouse is below the threshold, comprised of:
At most (20 * items-per-order) row selections with data retrieval.

Note: The above summary is provided for information only. The actual requirement is defined by the detailed transaction profile below.

2.8.2.2 For a given warehouse number (W_ID), district number (D_W_ID, D_ID), and stock level threshold ():

- The input data (see Clause 2.8.3.2) are communicated to the SUT.
- A database transaction is started.
- The row in the DISTRICT table with matching D_W_ID and D_ID is selected and D_NEXT_O_ID is retrieved.
- All rows in the ORDER-LINE table with matching OL_W_ID (equals W_ID), OL_D_ID (equals D_ID), and OL_O_ID (lower than D_NEXT_O_ID and greater than or equal to D_NEXT_O_ID minus 20) are selected. They are the items for 20 recent orders of the district.
- All rows in the STOCK table with matching S_I_ID (equals OL_I_ID) and S_W_ID (equals W_ID) from the list of distinct item numbers and with S_QUANTITY lower than are counted (giving - *).

Comment: Stocks must be counted only for distinct items. Thus, items that have been ordered more than once in the 20 selected orders must be aggregated into a single summary count for that item.

- The current database transaction is committed.

Comment: A commit is not needed as long as all the required ACID properties are satisfied (see Clause 2.8.2.3).

- The output data (see Clause 2.8.3.3) are communicated to the terminal.

2.8.2.3 Full serializability and repeatable reads are not required for the Stock-Level business transaction. All data read must be committed and no older than the most recently committed data prior to the time this business transaction was initiated. All other ACID properties must be maintained.

Comment: This clause allows the business transaction to be broken down into more than one database transaction.

2.8.3 Terminal I/O

2.8.3.1 For each transaction the originating terminal must display the following input/output screen with all input and output fields cleared (with either spaces or zeros) except for the Warehouse and District fields which have not changed and must display the fixed W_ID and D_ID values associated with that terminal.

```

      1           2           3           4           5           6           7           8
      1234567890123456789012345678901234567890123456789012345678901234567890
1      Stock-Level
2      Warehouse: 9999   District: 99
3
4      Stock Level Threshold: 99
5
6      low stock: 999
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24

```

2.8.3.2 The emulated user must enter, in the appropriate field of the input/output screen, the required input data which is organized as the distinct field:

2.8.3.3 The emulated terminal must display, in the appropriate field of the input/output screen, all input data and the output data which results from the execution of the transaction. The following fields are displayed: W_ID, D_ID, , and - *.

2.8.3.4 The following table summarizes the terminal I/O requirements for the Stock-Level transaction:

Enter	Display Row/Column	Coordinates
Non-repeating Group	W_ID	2/12
	D_ID	2/29
		4/24
	low_stock	6/12

2.8.3.5 For general terminal I/O requirements, see Clause 2.2.

Clause 3: TRANSACTION and SYSTEM PROPERTIES

3.1 The ACID Properties

It is the intent of this section to informally define the ACID properties and to specify a series of tests that must be performed to demonstrate that these properties are met.

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 running of this benchmark. The only exception to this rule is to allow non-repeatable reads for the Stock-Level transaction (see Clause 2.8.2.3).

3.1.2 No finite series of tests can prove that the ACID properties are fully supported. Passing the specified tests is a necessary, but not sufficient, condition for meeting the ACID requirements. However, for fairness of reporting, only the tests specified here are required and must appear in the Full Disclosure Report for this benchmark.

Comment: These tests are intended to demonstrate that the ACID principles are supported by the SUT and enabled during the performance measurement interval. They are not intended to be an exhaustive quality assurance test.

3.1.3 All mechanisms needed to insure full ACID properties must be enabled during both the test period and the 8 hours of steady state. For example, if the system under test relies on undo logs, then logging must be enabled for all transactions including those which do not include rollback in the transaction profile. When this benchmark is implemented on a distributed system, tests must be performed to verify that home and remote transactions, including remote transactions that are processed on two or more nodes, satisfy the ACID properties (See Clauses 2.4.1.7, 2.4.1.8, 2.5.1.5, and 2.5.1.6 for the definition of home and remote transactions).

3.1.4 Although the ACID tests do not exercise all transaction types of TPC-C, the ACID properties must be satisfied for all the TPC-C transactions.

3.1.5 Test sponsors reporting TPC results may perform ACID tests on any one system for which results have been disclosed, provided that they use the same software executables (e.g., operating system, data manager, transaction programs). For example, this clause would be applicable when results are reported for multiple systems in a product line. However, the durability tests described in Clauses 3.5.3.2 and 3.5.3.3 must be run on all the systems that are measured. All Full Disclosure Reports must identify the systems which were used to verify ACID requirements and full details of the ACID tests conducted and results obtained.

3.2 Atomicity Requirements

3.2.1 Atomicity Property Definition

The system under test must guarantee that database 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 Payment transaction for a randomly selected warehouse, district, and customer (by customer number as specified in Clause 2.5.1.2) and verify that the records in the CUSTOMER, DISTRICT, and WAREHOUSE tables have been changed appropriately.

3.2.2.2 Perform the Payment transaction for a randomly selected warehouse, district, and customer (by customer number as specified in Clause 2.5.1.2) and substitute a ROLLBACK of the transaction for the COMMIT of the transaction. Verify that the records in the CUSTOMER, DISTRICT, and WAREHOUSE tables have NOT been changed.

3.3 Consistency Requirements

3.3.1 Consistency Property Definition

Consistency is the property of the application that requires any execution of a database transaction to take the database from one consistent state to another, assuming that the database is initially in a consistent state.

3.3.2 Consistency Conditions

Twelve consistency conditions are defined in the following clauses to specify the level of database consistency required across the mix of TPC-C transactions. A database, when populated as defined in Clause 4.3, must meet all of these conditions to be consistent. If data is replicated, each copy must meet these conditions. Of the twelve conditions, explicit demonstration that the conditions are satisfied is required for the first four only. Demonstration of the last eight consistency conditions is not required because of the lengthy tests which would be necessary.

Comment: The consistency conditions were chosen so that they would remain valid within the context of a larger order-entry application that includes the five TPC-C transactions (See Clause 1.1). They are designed to be independent of the length of time for which such an application would be executed. Thus, for example, a condition involving I_PRICE was not included here since it is conceivable that within a larger application I_PRICE is modified from time to time.

3.3.2.1 Consistency Condition 1

Entries in the WAREHOUSE and DISTRICT tables must satisfy the relationship:

$$W_YTD = \text{sum}(D_YTD)$$

for each warehouse defined by ($W_ID = D_W_ID$).

3.3.2.2 Consistency Condition 2

Entries in the DISTRICT, ORDER, and NEW-ORDER tables must satisfy the relationship:

$$D_NEXT_O_ID - 1 = \max(O_ID) = \max(NO_O_ID)$$

for each district defined by ($D_W_ID = O_W_ID = NO_W_ID$) and ($D_ID = O_D_ID = NO_D_ID$). This condition does not apply to the NEW-ORDER table for any districts which have no outstanding new orders (i.e., the number of rows is zero).

3.3.2.3 Consistency Condition 3

Entries in the NEW-ORDER table must satisfy the relationship:

$$\max(\text{NO_O_ID}) - \min(\text{NO_O_ID}) + 1 = [\text{number of rows in the NEW-ORDER table for this district}]$$

for each district defined by NO_W_ID and NO_D_ID. This condition does not apply to any districts which have no outstanding new orders (i.e., the number of rows is zero).

3.3.2.4 Consistency Condition 4

Entries in the ORDER and ORDER-LINE tables must satisfy the relationship:

$$\text{sum}(\text{O_OL_CNT}) = [\text{number of rows in the ORDER-LINE table for this district}]$$

for each district defined by (O_W_ID = OL_W_ID) and (O_D_ID = OL_D_ID).

3.3.2.5 Consistency Condition 5

For any row in the ORDER table, O_CARRIER_ID is set to a null value if and only if there is a corresponding row in the NEW-ORDER table defined by (O_W_ID, O_D_ID, O_ID) = (NO_W_ID, NO_D_ID, NO_O_ID).

3.3.2.6 Consistency Condition 6

For any row in the ORDER table, O_OL_CNT must equal the number of rows in the ORDER-LINE table for the corresponding order defined by (O_W_ID, O_D_ID, O_ID) = (OL_W_ID, OL_D_ID, OL_O_ID).

3.3.2.7 Consistency Condition 7

For any row in the ORDER-LINE table, OL_DELIVERY_D is set to a null date/time if and only if the corresponding row in the ORDER table defined by (O_W_ID, O_D_ID, O_ID) = (OL_W_ID, OL_D_ID, OL_O_ID) has O_CARRIER_ID set to a null value.

3.3.2.8 Consistency Condition 8

Entries in the WAREHOUSE and HISTORY tables must satisfy the relationship:

$$\text{W_YTD} = \text{sum}(\text{H_AMOUNT})$$

for each warehouse defined by (W_ID = H_W_ID).

3.3.2.9 Consistency Condition 9

Entries in the DISTRICT and HISTORY tables must satisfy the relationship:

$$\text{D_YTD} = \text{sum}(\text{H_AMOUNT})$$

for each district defined by (D_W_ID, D_ID) = (H_W_ID, H_D_ID).

3.3.2.10 Consistency Condition 10

Entries in the CUSTOMER, HISTORY, ORDER, and ORDER-LINE tables must satisfy the relationship:

$$C_BALANCE = \text{sum}(OL_AMOUNT) - \text{sum}(H_AMOUNT)$$

where:

H_AMOUNT is selected by $(C_W_ID, C_D_ID, C_ID) = (H_C_W_ID, H_C_D_ID, H_C_ID)$

and

OL_AMOUNT is selected by:

$(OL_W_ID, OL_D_ID, OL_O_ID) = (O_W_ID, O_D_ID, O_ID)$ and

$(O_W_ID, O_D_ID, O_C_ID) = (C_W_ID, C_D_ID, C_ID)$ and

(OL_DELIVERY_D is not a null value)

3.3.2.11 Consistency Condition 11

Entries in the CUSTOMER, ORDER and NEW-ORDER tables must satisfy the relationship:

$$(\text{count}(*) \text{ from ORDER}) - (\text{count}(*) \text{ from NEW-ORDER}) = 2100$$

for each district defined by $(O_W_ID, O_D_ID) = (NO_W_ID, NO_D_ID) = (C_W_ID, C_D_ID)$.

3.3.2.12 Consistency Condition 12

Entries in the CUSTOMER and ORDER-LINE tables must satisfy the relationship:

$$C_BALANCE + C_YTD_PAYMENT = \text{sum}(OL_AMOUNT)$$

for any randomly selected customers and where OL_DELIVERY_D is not set to a null date/time.

3.3.3 Consistency Tests

3.3.3.1 Verify that the database is initially consistent by verifying that it meets the consistency conditions defined in Clauses 3.3.2.1 to 3.3.2.4. Describe the steps used to do this in sufficient detail so that the steps are independently repeatable.

3.3.3.2 Immediately after performing the verification process described in Clause 3.3.3.1, do the following:

1. Use the standard driving mechanism to submit transactions to the SUT. The transaction rate must be at least 90% of the reported tpmC rate and meet all other requirements of a reported measurement interval (see Clause 5.5). The test sponsor must include at least one check-point (as defined in Clause 5.5.2.2) within this interval. The SUT must be run at this rate for at least 5 minutes.
2. Stop submitting transactions to the SUT and then repeat the verification steps done for Clause 3.3.3.1. The database must still be consistent after applying transactions. Consistency Condition 4 need only be verified for rows added to the ORDER and ORDER-LINE tables since the previous verification.

3.4 Isolation Requirements

3.4.1 Isolation Property Definition

Isolation can be defined in terms of phenomena that can occur during the execution of concurrent database transactions. The following phenomena are possible:

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.

Isolation Level	P0	P1	P2	P3
0	Not Possible	Possible	Possible	Possible
1	Not Possible	Not Possible	Possible	Possible
2	Not Possible	Not Possible	Not Possible	Possible
3	Not Possible	Not Possible	Not Possible	Not Possible

The following terms are defined:

T₁ = New-Order transaction

T₂ = Payment transaction

T₃ = Delivery transaction

T₄ = Order-Status transaction

T₅ = Stock-Level transaction

T_n = Any arbitrary transaction

Although arbitrary, the transaction T_n may not do dirty writes.

The following table defines the isolation requirements which must be met by the TPC-C transactions.

Req. #	For transactions in this set:	these phenomena:	must NOT be seen by this transaction:	Textual Description:
1.	$\{T_i, T_j\}$ $1 \leq i, j \leq 4$	P0, P1, P2, P3	T_i	Level 3 isolation between New-Order, Payment, Delivery, and Order-Status transactions.
2.	$\{T_i, T_n\}$ $1 \leq i \leq 4$	P0, P1, P2	T_i	Level 2 isolation for New-Order, Payment, Delivery, and Order-Status transactions relative to any arbitrary transaction.
3.	$\{T_i, T_5\}$ $1 \leq i \leq n$	P0, P1	T_5	Level 1 isolation for Stock-Level transaction relative to TPC-C transactions and any arbitrary transaction.

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

3.4.2 Isolation Tests

For conventional locking schemes, isolation should 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. (Examples of different validation techniques are shown in Isolation Test 7, Clause 3.4.2.7).

3.4.2.1 Isolation Test 1

This test demonstrates isolation for read-write conflicts of Order-Status and New-Order transactions. Perform the following steps:

1. Start a New-Order transaction T1.
2. Stop transaction T1 immediately prior to COMMIT.
3. Start an Order-Status transaction T2 for the same customer used in T1. Transaction T2 attempts to read the data for the order T1 has created.
4. Verify that transaction T2 waits.
5. Allow transaction T1 to complete. T2 should now complete.
6. Verify that the results from T2 match the data entered in T1.

3.4.2.2 Isolation Test 2

This test demonstrates isolation for read-write conflicts of Order-Status and New-Order transactions when the New-Order transaction is ROLLED BACK. Perform the following steps:

1. Perform an Order-Status transaction T0 for some customer. Let T0 complete.
2. Start a New-Order transaction T1 for the same customer used in T0.
3. Stop transaction T1 immediately prior to COMMIT.
4. Start an Order-Status transaction T2 for the same customer used in T0. Transaction T2 attempts to read the data for the order T1 has created.
5. Verify that transaction T2 waits.
6. ROLLBACK transaction T1. T2 should now complete.
7. Verify that the data returned from T2 match the data returned by T0.

3.4.2.3 Isolation Test 3

This test demonstrates isolation for write-write conflicts of two New-Order transactions. Perform the following steps:

1. Start a New-Order transaction T1.
2. Stop transaction T1 immediately prior to COMMIT.
3. Start another New-Order transaction T2 for the same customer as T1.
4. Verify that transaction T2 waits.
5. Allow transaction T1 to complete. T2 should now complete.
6. Verify that the order number returned for T2 is one greater than the order number for T1. Verify that the value of D_NEXT_O_ID reflects the results of both T1 and T2, i.e., it has been incremented by two and is one greater than the order number for T2.

3.4.2.4 Isolation Test 4

This test demonstrates isolation for write-write conflicts of two New-Order transactions when one transaction is ROLLED BACK. Perform the following steps:

1. Start a New-Order transaction T1 which contains an invalid item number.
2. Stop transaction T1 immediately prior to ROLLBACK.
3. Start another New-Order transaction T2 for the same customer as T1.
4. Verify that transaction T2 waits.
5. Allow transaction T1 to complete. T2 should now complete.
6. Verify that the order number returned for T2 is one greater than the previous order number. Verify that the value of D_NEXT_O_ID reflects the result of only T2, i.e., it has been incremented by one and is one greater than the order number for T2.

3.4.2.5 Isolation Test 5

This test demonstrates isolation for write-write conflicts of Payment and Delivery transactions. Perform the following steps:

1. Start a Delivery transaction T1.
2. Stop transaction T1 immediately prior to COMMIT.
3. Start a Payment transaction T2 for the same customer as one of the new orders being delivered by T1.
4. Verify that transaction T2 waits.
5. Allow transaction T1 to complete. T2 should now complete.
6. Verify that C_BALANCE reflects the results of both T1 and T2.

Comment: If the Delivery business transaction is executed as multiple database transactions, then the transaction T1, in bullet 6 above, can be chosen to be one of these database transactions.

3.4.2.6 Isolation Test 6

This test demonstrates isolation for write-write conflicts of Payment and Delivery transactions when the Delivery transaction is ROLLED BACK. Perform the following steps:

1. Start a Delivery transaction T1.
2. Stop transaction T1 immediately prior to COMMIT.
3. Start a Payment transaction T2 for the same customer as one of the new orders being delivered by T1.
4. Verify that transaction T2 waits.
5. ROLLBACK transaction T1. T2 should now complete.
6. Verify that C_BALANCE reflects the results of only transaction T2.

3.4.2.7 Isolation Test 7

This test demonstrates repeatable reads for the New-Order transaction while an interactive transaction updates the price of an item. Given two random item number i_1 and i_2 , perform the following steps:

1. Start a transaction T1. Query I_PRICE from items i_1 and i_2 . COMMIT transaction T1.
2. Start a New-Order transaction T2 for a group of items including item i_1 twice and item i_2 .
3. Stop transaction T2 after querying the price of item i_1 a first time and immediately before querying the prices of item i_1 and of item i_2 a second time.
4. Start a transaction T3. Increase the price of items i_1 and i_2 by 10 percent.

Case A, if transaction T3 stalls:

- 5A. Continue transaction T2 and verify that the price of items i_1 (the second time) and i_2 match the values read by transaction T1. COMMIT transaction T2.
- 6A. Transaction T3 should now complete and be COMMITTED.
- 7A. Start a transaction T4. Query I_PRICE from items i_1 and i_2 . COMMIT transaction T4.

8A. Verify that the prices read by transaction T4 match the values set by transaction T3.

Case B, if transaction T3 does not stall and transaction T2 ROLLS BACK:

5B. Transaction T3 has completed and has been COMMITTED.

6B. Continue transaction T2 and verify that it is instructed to ROLL BACK by the data manager.

7B. Start a transaction T4. Query I_PRICE from items and , . COMMIT transaction T4

8B. Verify that the prices read by transaction T4 match the values set by transaction T3.

Case C, if transaction T3 ROLLS BACK:

5C. Verify that transaction T3 is instructed to ROLL BACK by the data manager.

6C. Continue transaction T2 and verify that the price of items (the second time) and , match the values read by transaction T1. COMMIT transaction T2.

7C. Start a transaction T4. Query I_PRICE from items and , . COMMIT transaction T4

8C. Verify that the prices read by transaction T4 match the values read by transactions T1 and T2.

Case D, if transaction T3 does not stall and no transaction is ROLLED BACK:

5D. Transaction T3 has completed and has been COMMITTED.

6D. Continue transaction T2 and verify that the price of items (the second time) and , match the values read by transaction T1. COMMIT transaction T2.

7D. Start a transaction T4. Query I_PRICE from items and , . COMMIT transaction T4

8D. Verify that the prices read by transaction T4 match the values set by transaction T3.

Comment 1: This test is successfully executed if either case A, B, C or D of the above steps are followed. The test sponsor must disclose the case followed during the execution of this test.

Comment 2: If the implementation uses replication on the ITEM table and all transactions in Isolation Test 7 use the same copy of the ITEM table, updates to the ITEM table are not required to be propagated to other copies of the ITEM table. This relaxation of ACID properties on a replicated table is only valid under the above conditions and in the context of Isolation Test 7.

Comment 3: Transactions T1, T2, and T4 are not used to measure throughput and are only used in the context of Isolation Test 7.

3.4.2.8 Isolation Test 8

This test demonstrates isolation for Level 3 (phantom) protection between a Delivery and a New-Order transaction. Perform the following steps:

1. Remove all rows for a randomly selected district and warehouse from the NEW-ORDER table.
2. Start a Delivery transaction T1 for the selected warehouse.
3. Stop T1 immediately after reading the NEW-ORDER table for the selected district. No qualifying row should be found.
4. Start a New-Order transaction T2 for the same warehouse and district.

Case A, if transaction T2 stalls:

- 5A. Continue transaction T1 by repeating the read of the NEW-ORDER table for the selected district.
- 6A. Verify that there is still no qualifying row found.
- 7A. Complete and COMMIT transaction T1.
- 8A. Transaction T2 should now complete.

Case B, if transaction T2 does not stall:

- 5B. Complete and COMMIT transaction T2.
- 6B. Continue transaction T1 by repeating the read of the NEW-ORDER table for the selected district.
- 7B. Verify that there is still no qualifying row found.
- 8B. Complete and COMMIT transaction T1.

Comment: Note that other cases, besides A and B, are possible. The intent of this test is to demonstrate that in all cases when T1 repeats the read of the NEW-ORDER table for the selected district, there is still no qualifying row found.

3.4.2.9 Isolation Test 9

This test demonstrates isolation for Level 3 (phantom) protection between an Order-Status and a New-Order transaction. Perform the following steps:

- 1. Start an Order-Status transaction T1 for a selected customer.
- 2. Stop T1 immediately after reading the ORDER table for the selected customer. The most recent order for that customer is found.
- 3. Start a New-Order transaction T2 for the same customer.

Case A, if transaction T2 stalls:

- 5A. Continue transaction T1 by repeating the read of the ORDER table for the selected customer.
- 6A. Verify that the order found is the same as in step 3.
- 7A. Complete and COMMIT transaction T1.
- 8A. Transaction T2 should now complete.

Case B, if transaction T2 does not stall:

- 5B. Complete and COMMIT transaction T2.
- 6B. Continue transaction T1 by repeating the read of the ORDER table for the selected district.
- 7B. Verify that the order found is the same as in step 3.
- 8B. Complete and COMMIT transaction T1.

Comment: Note that other cases, besides A and B, are possible. The intent of this test is to demonstrate that in all cases when T1 repeats the read of the ORDER table for the selected customer, the order found is the same as in step 3.

3.5 Durability Requirements

The tested system 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. However, the limited nature of the tests listed must not be interpreted to allow other unrecoverable single points of failure.

3.5.1 Durable Medium is a data storage medium that is either:

1. An inherently non-volatile medium (e.g., magnetic disk, magnetic tape, optical disk, etc.) or
2. A volatile medium that will ensure the transfer of data automatically, before any data is lost, to an inherently non-volatile medium after the failure of external power independently of reapplication 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 stated in item 2 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

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-C.

Comment 2: Although the order of operations in the transaction profiles (Clause 2) is immaterial, the actual communication of the output data cannot begin until the commit operation has successfully completed.

3.5.3 List of single failures

The Single Points of Failure apply to components of the SUT that contribute to the durability requirement. In configurations where more than one instance of an operating system performs an identical benchmark function, the tests for the failures listed here must be completed on at least one such instance. In addition, if multiple instances of an operating system manage data that is maintained as a single image for the benchmark application (e.g., a database cluster), then the Power Failure test must also be performed simultaneously on all such instances.

Comment 1: An example of multiple systems performing an identical function is a single database image on a clustered system in TPC-C.

Comment 2: A single test can adequately satisfy the requirements of multiple single points of failure (e.g., A single "system crash test" could be used for clauses 3.5.3.2, 3.5.3.3, and 3.5.3.4.)

Comment 3: The power failure requirement can be satisfied by including sufficient UPS's to guarantee system availability of all components that fall under the power failure requirement for a period of at least 30 minutes. Use of

a UPS-protected configuration must not introduce new single points of failure that are not protected by other parts of the configuration. This requirement may be proven either through a measurement or through a calculation of the 30-minute power requirements (in watts) for the portion of the SUT that is protected multiplied by 1.4.

Comment 4: The term "simultaneously" as applied to a power failure of multiple instances within the SUT is interpreted to mean within 3 seconds to allow for variances in a manual procedure that may be used to accomplish the test.

3.5.3.1 Permanent irrecoverable failure of any single durable medium during the Measurement Interval containing TPC-C database tables or recovery log data.

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.

3.5.3.2 Instantaneous interruption (system or subsystem crash/system hang) in processing which causes all or part of the processing of atomic transactions to halt.

Comment 1: This may imply 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.

Comment 2: In configurations where more than one instance of an operating system can participate in an atomic transaction and are connected via a physical medium other than an integrated bus (e.g., bus extender cable, high speed LAN, or other connection methods between the multiple instances of the operating system that could be vulnerable to a loss from physical disruption), the instantaneous interruption of this communication is included in this definition as an item that needs to be tested. Interruption of one instance of redundant connections is required.

Comment 3: It is not the intention of this clause to require interruption of communication to disk towers or a disk subsystem where redundancy exists. For example, log disks can be assumed to provide redundancy for data disks.

3.5.3.3 Failure of all or parts 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.

3.5.3.4 Power Failure

Comment: Loss of all external power to the SUT for an indefinite time period. This must include at least all portions of the SUT that participate in the database portions of transactions.

3.5.4 Durability Tests

The intent of these tests is to demonstrate that all transactions whose output messages have been received at the terminal or RTE 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.

It is required that the system crash test(s) and the loss of memory test(s) described in Clauses 3.5.3.2 and 3.5.3.3 be performed under full terminal load and a fully scaled database. The tpmC of the test run(s) for Clauses 3.5.3.2 and 3.5.3.3 must be at least 90% of the tpmC reported for the benchmark.

The durable media failure test(s) described in Clause 3.5.3.1 may be performed on a subset of the SUT configuration and database. The tpmC of the test run for Clause 3.5.3.1 must be at least 10% of the tomC reported for the benchmark.

For the SUT subset, all multiple hardware components, such as processors and disk/controllers in the full SUT configuration, must be represented by the greater of 10% of the configuration or two of each of the multiple hardware components. The database must be scaled to at least 10% of the fully scaled database, with a minimum of two warehouses. An exception to the configuration requirements stated above may be allowed by the TPC Auditor in order to reduce benchmark complexity. Any such exception must be documented in the attestation letter from the Auditor. Furthermore, the standard driving mechanism must be used in this test. The test sponsor must state that to the best of their knowledge, a fully scaled test would also pass all durability tests.

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

1. Compute the sum of D_NEXT_O_ID for all rows in the DISTRICT table to determine the current count of the total number of orders (count1).
2. Start submitting TPC-C transactions. The transaction rate must be that described above and meet all other requirements of a reported measurement interval (see Clause 5.5), excluding the requirement that the interval contain at least four checkpoint (see Clause 5.5.2.2). The SUT must be run at this rate for at least 5 minutes. On the Driver System, record committed and rolled back New-Order transactions in a "success" file.
3. Cause the failure selected from the list in Clause 3.5.3.
4. Restart the system under test using normal recovery procedures.
5. Compare the contents of the "success" file and the ORDER table to verify that every record in the "success" file for a committed New-Order transaction has a corresponding record in the ORDER table and that no entries exist for rolled back transactions.

Repeat step 1 to determine the total number of orders (count2). Verify that count2-count1 is greater or equal to the number of records in the "success" file for committed New-Order transactions. If there is an inequality, the ORDER table must contain additional records and the difference must be less than or equal to the number of terminals simulated.

Comment: This difference should be due only to transactions which were committed on the system under test, but for which the output data was not displayed on the input/output screen before the failure.

6. Verify Consistency Condition 3 as specified in Clause 3.3.2.3.

3.5.5 Additional Requirements

3.5.5.1 The recovery mechanism cannot use the contents of the HISTORY table to support the durability property.

3.5.5.2 Roll-forward recovery from an archive database copy (e.g., a copy taken prior to the run) using redo log data is not acceptable as the recovery mechanism in the case of failures listed in Clause 3.5.3.2 and 3.5.3.3. Note that "checkpoints", "control points", "consistency points", etc. of the database taken during a run are not considered to be archives.

Clause 4: SCALING and DATABASE POPULATION

4.1 General Scaling Rules

The throughput of the TPC-C benchmark is driven by the activity of the terminals connected to each warehouse. To increase the throughput, more warehouses and their associated terminals must be configured. Each warehouse requires a number of rows to populate the database along with some storage space to maintain the data generated during a defined period of activity called **60-day period**. These requirements define how storage space and database population scale with throughput.

4.1.1 The intent of the scaling requirements is to maintain the ratio between the transaction load presented to the system under test, the cardinality of the tables accessed by the transactions, the required space for storage, and the number of terminals generating the transaction load.

4.1.2 Should any scaling value in Clause 4.2 be exceeded, the others must be increased proportionally to maintain the same ratios among them as in Clause 4.2.

4.1.3 The reported throughput may not exceed the maximum allowed by the scaling requirements in Clause 4.2 and the pacing requirements in Clause 5.2. While the reported throughput may fall short of the maximum allowed by the configured system, the price/performance computation (see Clause 7.1) must report the price for the system as actually configured. To prevent over-scaling of systems, the reported throughput cannot fall short of 9 tpmC per configured warehouse.

Comment: The maximum throughput is achieved with infinitely fast transactions resulting in a null response time and minimum required wait times. The intent of this clause is to prevent reporting a throughput that exceeds this maximum, which is computed to be 12.86 tpmC per warehouse. The above 9 tpmC represents 70% of the computed maximum throughput.

4.2 Scaling Requirements

4.2.1 The WAREHOUSE table is used as the base unit of scaling. The cardinality of all other tables (except for ITEM) is a function of the number of configured warehouses (i.e., cardinality of the WAREHOUSE table). This number, in turn, determines the load applied to the system under test which results in a reported throughput (see Clause 5.4).

Comment 1: The cardinality of the HISTORY, NEW-ORDER, ORDER, and ORDER-LINE tables will naturally vary as a result of repeated test executions. The initial database population and the transaction profiles are designed to minimize the impact of this variation on performance and maintain repeatability between subsequent test results.

Comment 2: The cardinality of the ITEM table is constant regardless of the number of configured warehouses, as all warehouses maintain stocks for the same catalog of items.

4.2.2 Configuration

The following scaling requirements represent the initial configuration for the test described in Clause 5:

1. For each active warehouse in the database, the SUT must accept requests for transactions from a population of 10 terminals.

- For each table that composes the database, the cardinality of the initial population per warehouse is specified as follows:

Table Name	Cardinality (in rows)	Typical ³ Row Length (in bytes)	Typical ³ Table Size (in 1,000 bytes)
WAREHOUSE	1	89	0.089
DISTRICT	10	95	0.950
CUSTOMER	30k	655	19,650
HISTORY ¹	30k	46	1,380
ORDER ⁴	30k	24	720
NEW-ORDER ⁴	9k	8	72
ORDER-LINE ⁴	300k	54	16,200
STOCK	100k	306	30,600
ITEM ²	100k	82	8,200

- Small variations: subject to test execution as rows may be inserted and deleted by transaction activity from test executions.
- Fixed cardinality: does not scale with number of warehouses.
- Typical lengths and sizes given here are examples, not requirements, of what could result from an implementation (sizes do not include storage/access overheads).
- One percent (1%) variation in row cardinality is allowed to account for the random variation encountered during the initial data loading of the database.

Note: The symbol "k" used in the cardinality column means one thousand

- Storage must be priced for sufficient space to store and maintain the data generated during a period of 60 days of activity with an average of 8 hours per day at the reported throughput called the **60-day period**. This space must be computed according to Clause 4.2.3 and must be usable by the data manager to store and maintain the rows that would be added to the HISTORY, ORDER, and ORDER-LINE tables during the 60-day period.
- The increment (granularity) for scaling the database and the terminal population is one warehouse, comprised of one WAREHOUSE row, 10 DISTRICT rows, their associated CUSTOMER, HISTORY, ORDER, NEW-ORDER, and ORDER-LINE rows, 100,000 STOCK rows, 10 terminals, and priced storage for the 60-day period.

Comment: Over-scaling the database, i.e., configuring a larger number of warehouses and associated tables (W_c) than what is actually accessed during the measurement (W_a) is permitted, provided the following conditions are met:

Let, W_c = number of warehouses configured at database generation,
 W_a = number of warehouses accessed during the measurement (active warehouses),
 W_i = number of warehouses not accessed during the measurement (inactive warehouses).

§ It can be demonstrated that inactive warehouses are not accessed during the measurement. This fact must be demonstrated in one of the following ways:

- rows in the WAREHOUSE table that pertain to the inactive warehouses (W_i) must be deleted prior to the measurement,
- show that the sum of $D_NEXT_O_ID$ for each of the inactive warehouses does not change during the measurement, and that W_YTD for each of the inactive warehouses does not change during the measurement.

- the reported throughput cannot fall short of 9 tpmC per configured warehouse (W_c -see Clause 4.1.3),
- the 60-day space computations must be computed based on W_c , the number of warehouses configured at database generation.

4.2.3 60-Day Space Computation

The storage space required for the 60-day period must be determined as follows:

1. The test database must be built including the initial database population (see Clause 4.3) and all indices present during the test.
2. The test database must be built to sustain the reported throughput during an eight hour period. This excludes performing on the database any operation that does not occur during the measurement interval (see Clause 5.5).
3. The total storage space allocated for the test database must be decomposed into the following:
 - **Free-Space:** any space allocated to the test database and which is available for future use. It is comprised of all database storage space not used to store a database entity (e.g., a row, an index, a metadatum) or not used as formatting overhead by the data manager.
 - **Dynamic-Space:** any space used to store existing rows from the dynamic tables (i.e., the HISTORY, ORDER, and ORDER-LINE tables). It is comprised of all database storage space used to store rows and row storage overhead for the dynamic tables. It includes any data that is added to the database as a result of inserting a new row independently of all indices. It does not include index data or other overheads such as index overhead, page overhead, block overhead, and table overhead.
 - **Static-Space:** any space used to store static information and indices. It is comprised of all space allocated to the test database and which does not qualify as either Free-Space or Dynamic-Space.
4. Given that the system must be configured to sustain the reported throughput during an eight hour period, the database must allow the dynamic tables to grow accordingly for at least eight hours without impacting performance. Free-Space used to allow growth of the dynamic tables for an eight hour day at the reported throughput is called the **Daily-Growth**. Given W , the number of configured warehouses on the test system, the Daily-Growth must be computed as:

$$\text{Daily-Growth} = (\text{dynamic-Space} / (W * 62.5)) * \text{tpmC}$$

Note: In the formula above, 62.5 is used as a normalizing factor since the initial database population for each warehouse holds the Dynamic-Space required for an eight hour day of activity at 62.5 tpmC.

5. Any Free-Space beyond 150% of the Daily-Growth is called **Daily-Spread**, and must be added to the Dynamic-Space when computing the storage requirement for the 60-day period. The Daily-Spread must be computed as:

$$\text{Daily-Spread} = \text{Free-Space} - 1.5 * \text{Daily-Growth}$$

If the computed Daily-Spread is negative, then a null value must be used for Daily-Spread.

6. The **60-Day-Space** must be computed as:

$$60\text{-Day-Space} = \text{Static-Space} + 60 * (\text{Daily-Growth} + \text{Daily-Spread})$$

7. The Dynamic-Space present in the test database is considered as part of the 60-Day-Space.

4.3 Database Population

4.3.1 The test described in Clause 5 requires that the properly scaled population be present in the test database. Each table must contain the number of rows defined in Clause 4.2.2 prior to test execution (e.g., the New-Order table must contain 2,000 rows per warehouse).

4.3.2 Definition of Terms

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

Comment: For the purpose of populating the initial database only, random numbers can be generated by selecting entries in sequence from a set of at least 10,000 pregenerated random numbers. This technique cannot be used for the field O_OL_CNT.

4.3.2.2 The notation **random a-string** [..] (respectively, **n-string** [..]) represents a string of random alphanumeric (respectively, numeric) characters of a random length of minimum , maximum , and mean $(, +)/2$.

Comment 1: The character set used must be able to represent a minimum of 128 different characters.

Comment 2: Generating such strings can be implemented by the concatenation of two strings selected at random from two separate arrays of strings, and where:

1. Both arrays contain a minimum of 10 different strings of characters.
2. The first array contains strings of characters.
3. The second array contains strings of lengths uniformly distributed between zero and $(, &)$ characters.
4. Both arrays may contain strings that are pertinent to the row and the attribute (e.g., use an actual first name for C_FIRST) instead of strings of random characters, as long as this does not bring any improvement to the reported metrics.

4.3.2.3 The customer last name (C_LAST) must be generated by the concatenation of three variable length syllables selected from the following list:

0	1	2	3	4	5	6	7	8	9
BAR	OUGHT	ABLE	PRI	PRES	ESE	ANTI	CALLY	ATION	EING

Given a number between 0 and 999, each of the three syllables is determined by the corresponding digit in the three digit representation of the number. For example, the number 371 generates the name PRICALLYOUGHT, and the number 40 generates the name BARPRESBAR.

4.3.2.4 The notation **unique within** [] represents any one value within a set of contiguous values, unique within the group of rows being populated. When several groups of rows of the same type are populated (e.g., there is one group of customer type rows for each district type row), each group must use the same set of contiguous values.

4.3.2.5 The notation **random within** [..] represents a random value independently selected and uniformly distributed between and , inclusively, with a mean of $(+,)/2$, 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.

4.3.2.6 The notation **random permutation of [..]** represents a sequence of numbers from to , arranged into a random order. This is commonly known as a permutation (or selection) without replacement.

4.3.2.7 The warehouse zip code (W_ZIP), the district zip code (D_ZIP) and the customer zip code (C_ZIP) must be generated by the concatenation of:

1. A random n-string of 4 numbers, and
2. The constant string '11111'.

Given a random n-string between 0 and 9999, the zip codes are determined by concatenating the n-string and the constant '11111'. This will create 10,000 unique zip codes. For example, the n-string 0503 concatenated with 11111, will make the zip code 050311111.

Comment: With 30,000 customers per warehouse and 10,000 zip codes available, there will be an average of 3 customers per warehouse with the same zip code.

4.3.3 Table Population Requirements

4.3.3.1 The initial database population must be comprised of:

- 100,000 rows in the ITEM table with:
 - I_ID unique within [100,000]
 - I_IM_ID random within [1 .. 10,000]
 - I_NAME random a-string [14 .. 24]
 - I_PRICE random within [1.00 .. 100.00]
 - I_DATA random a-string [26 .. 50]. For 10% of the rows, selected at random, the string "ORIGINAL" must be held by 8 consecutive characters starting at a random position within I_DATA
- 1 row in the WAREHOUSE table for each configured warehouse with:
 - W_ID unique within [- % " -]
 - W_NAME random a-string [6 .. 10]
 - W_STREET_1 random a-string [10 .. 20]
 - W_STREET_2 random a-string [10 .. 20]
 - W_CITY random a-string [10 .. 20]
 - W_STATE random a-string of 2 letters
 - W_ZIP generated according to Clause 4.3.2.7
 - W_TAX random within [0.0000 .. 0.2000]
 - W_YTD = 300,000.00

For each row in the WAREHOUSE table:

- o 100,000 rows in the STOCK table with:

S_I_ID unique within [100,000]

S_W_ID = W_ID

S_QUANTITY random within [10 .. 100]

S_DIST_01 random a-string of 24 letters

S_DIST_02 random a-string of 24 letters

S_DIST_03 random a-string of 24 letters

S_DIST_04 random a-string of 24 letters

S_DIST_05 random a-string of 24 letters

S_DIST_06 random a-string of 24 letters

S_DIST_07 random a-string of 24 letters

S_DIST_08 random a-string of 24 letters

S_DIST_09 random a-string of 24 letters

S_DIST_10 random a-string of 24 letters

S_YTD = 0

S_ORDER_CNT = 0

S_REMOTE_CNT = 0

S_DATA random a-string [26 .. 50]. For 10% of the rows, selected at random, the string "ORIGINAL" must be held by 8 consecutive characters starting at a random position within S_DATA

- o 10 rows in the DISTRICT table with:

D_ID unique within [10]

D_W_ID = W_ID

D_NAME random a-string [6 .. 10]

D_STREET_1 random a-string [10 .. 20]

D_STREET_2 random a-string [10 .. 20]

D_CITY random a-string [10 .. 20]

D_STATE random a-string of 2 letters

D_ZIP generated according to Clause 4.3.2.7

D_TAX random within [0.0000 .. 0.2000]

D_YTD = 30,000.00

D_NEXT_O_ID = 3,001

For each row in the DISTRICT table:

* 3,000 rows in the CUSTOMER table with:

C_ID unique within [3,000]

C_D_ID = D_ID

C_W_ID = D_W_ID

C_LAST generated according to Clause 4.3.2.3, iterating through the range of [0 .. 999] for the first 1,000 customers, and generating a non-uniform random number using the function NURand(255,0,999) for each of the remaining 2,000 customers. The run-time constant C (see Clause 2.1.6) used for the database population must be randomly chosen independently from the test run(s).

C_MIDDLE = "OE"

C_FIRST random a-string [8 .. 16]

C_STREET_1 random a-string [10 .. 20]

C_STREET_2 random a-string [10 .. 20]

C_CITY random a-string [10 .. 20]

C_STATE random a-string of 2 letters

C_ZIP generated according to Clause 4.3.2.7

C_PHONE random n-string of 16 numbers

C_SINCE date/time given by the operating system when the CUSTOMER table was populated.

C_CREDIT = "GC". For 10% of the rows, selected at random, C_CREDIT = "BC"

C_CREDIT_LIM = 50,000.00

C_DISCOUNT random within [0.0000 .. 0.5000]

C_BALANCE = -10.00

C_YTD_PAYMENT = 10.00

C_PAYMENT_CNT = 1

C_DELIVERY_CNT = 0

C_DATA random a-string [300 .. 500]

For each row in the CUSTOMER table:

- 1 row in the HISTORY table with:

H_C_ID = C_ID

H_C_D_ID = H_D_ID = D_ID

H_C_W_ID = H_W_ID = W_ID

H_DATE current date and time

H_AMOUNT = 10.00

H_DATA random a-string [12 .. 24]

- * 3,000 rows in the ORDER table with:

O_ID unique within [3,000]

O_C_ID selected sequentially from a random permutation of [1 .. 3,000]

O_D_ID = D_ID

O_W_ID = W_ID

O_ENTRY_D current date/time given by the operating system

O_CARRIER_ID random within [1 .. 10] if O_ID < 2,101, null otherwise

O_OL_CNT random within [5 .. 15]

O_ALL_LOCAL = 1

For each row in the ORDER table:

- A number of rows in the ORDER-LINE table equal to O_OL_CNT, generated according to the rules for input data generation of the New-Order transaction (see Clause 2.4.1) with:

OL_O_ID = O_ID

OL_D_ID = D_ID

OL_W_ID = W_ID

OL_NUMBER unique within [O_OL_CNT]

OL_I_ID random within [1 .. 100,000]

OL_SUPPLY_W_ID = W_ID

OL_DELIVERY_D = O_ENTRY_D if OL_O_ID < 2,101, null otherwise

OL_QUANTITY = 5

OL_AMOUNT = 0.00 if OL_O_ID < 2,101, random within [0.01 .. 9,999.99] otherwise

OL_DIST_INFO random a-string of 24 letters

- * 900 rows in the NEW-ORDER table corresponding to the last 900 rows in the ORDER table for that district (i.e., with NO_O_ID between 2,101 and 3,000), with:

NO_O_ID = O_ID

NO_D_ID = D_ID

NO_W_ID = W_ID

Comment: Five percent (5%) variation from the target cardinality of S_DATA with "ORIGINAL", I_DATA with "ORIGINAL", and C_CREDIT with "BC" is allowed to account for the random variation encountered during the initial data loading of the database.

4.3.3.2 The implementation may not take advantage of the fact that some fields are initially populated with a fixed value. For example, storage space cannot be saved by defining a default value for the field C_CREDIT_LIM and storing this value only once in the database.

