- **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** represents the data type for a date value that includes a time component. The date component must be able to hold any date between 1st January 1900 and 31st December 2100. The time component must be capable of representing the range of time values from 00:00:00 to 23:59:59 with a resolution of at least one second. Date and Time must be implemented using data types that are defined by the DBMS for that use.
- **numeric(m [,n])** means an unsigned numeric value with at least m total decimal digits, of which n digits are to the right (after) the decimal point. The attribute must be able to hold all possible values which can be expressed as numeric(m,n). Omitting n, as in numeric(m), indicates the same as numeric(m,0). 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 are defined by the DBMS as being an exact numeric data type or that satisfy the ANSI SQL Standard definition of being an exact numeric representation.
- **signed numeric(m [,n])** is identical to numeric(m [,n]) except that it can represent both positive and negative values.
- **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.

Comment 3: A **signed numeric** data type may be used (at the sponsor's discretion) anywhere a **numeric** data type is defined.

WAREHOUSE Table Layout

Field Name	Field Definition	Comments
W_ID	2*W unique IDs	*W Warehouses are populated*
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	signed numeric(4,4)	*Sales tax*
W_YTD	signed numeric(12,2)	*Year to date balance*

Primary Key: W_ID

DISTRICT Table Layout

Field Name	Field Definition	Comments
D_ID	20 unique IDs	*10 are populated per warehouse*
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	signed numeric(4,4)	*Sales tax*
D_YTD	signed numeric(12,2)	*Year to date balance*
D_NEXT_O_ID	10,000,000 unique IDs	*Next available Order number*

Primary Key: (D_W_ID, D_ID)

D_W_ID Foreign Key, references W_ID

CUSTOMER Table Layout

Field Name	Field Definition	Comments
C_ID	96,000 unique IDs	*3,000 are populated per district*
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	*"GC"=good, "BC"=bad*
C_CREDIT_LIM	signed numeric(12, 2)	
C_DISCOUNT	signed numeric(4, 4)	
C_BALANCE	signed numeric(12, 2)	
C_YTD_PAYMENT	signed numeric(12, 2)	
C_PAYMENT_CNT	numeric(4)	
C_DELIVERY_CNT	numeric(4)	
C_DATA	variable text, size 500	*Miscellaneous information*

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

Field Name	Field Definition	Comments
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	signed numeric(6, 2)	
H_DATA	variable text, size 24	*Miscellaneous information*

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

Field Name	Field Definition	Comments
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

Field Name	Field Definition	Comments
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	numeric(2)	*Count of Order-Lines*
O_ALL_LOCAL	numeric(1)	

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

Field Name	Field Definition	Comments
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)	
OL_AMOUNT	signed numeric(6, 2)	
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

Field Name	Field Definition	Comments
I_ID	200,000 unique IDs	*100,000 items are populated*
I_IM_ID	200,000 unique IDs	*Image ID associated to Item*
I_NAME	variable text, size 24	
I_PRICE	numeric(5, 2)	
I_DATA	variable text, size 50	*Brand information*

Primary Key: I_ID

STOCK Table Layout

Field Name	Field Definition	Comments
S_I_ID	200,000 unique IDs	*100,000 populated per warehouse*
S_W_ID	2*W unique IDs	
S_QUANTITY	signed numeric(4)	
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)	
S_ORDER_CNT	numeric(4)	
S_REMOTE_CNT	numeric(4)	
S_DATA	variable text, size 50	*Make information*

Primary Key: (S_W_ID, S_I_ID)

S_W_ID Foreign Key, references W_ID

S_I_ID Foreign Key, references I_ID

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 (*ol_cnt*) 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 *rbk* 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 *ol_cnt* 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 *rbk* = 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 *x* within [1 .. 100];

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

   - If *x* = 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 *ol_cnt* = 10), comprised of:

   (1 * *ol_cnt*) row selections with data retrieval,

   (1 * *ol_cnt*) row selections with data retrieval and update,

   (1 * *ol_cnt*) 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 (*ol_cnt*, 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 *ol_cnt*.

- 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 *brand-generic* field for that item is set to "B", otherwise, the *brand-generic* 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 *total-amount* for the complete order is computed as:

  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 *unused* 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.

2.5 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 *x* and *y* within [1 .. 100];

- If *x* <= 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 *x* > 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 ≠W_ID). The customer is paying through a warehouse and a district other than his/ her own.
- If *y* <= 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 *y* > 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) in 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.

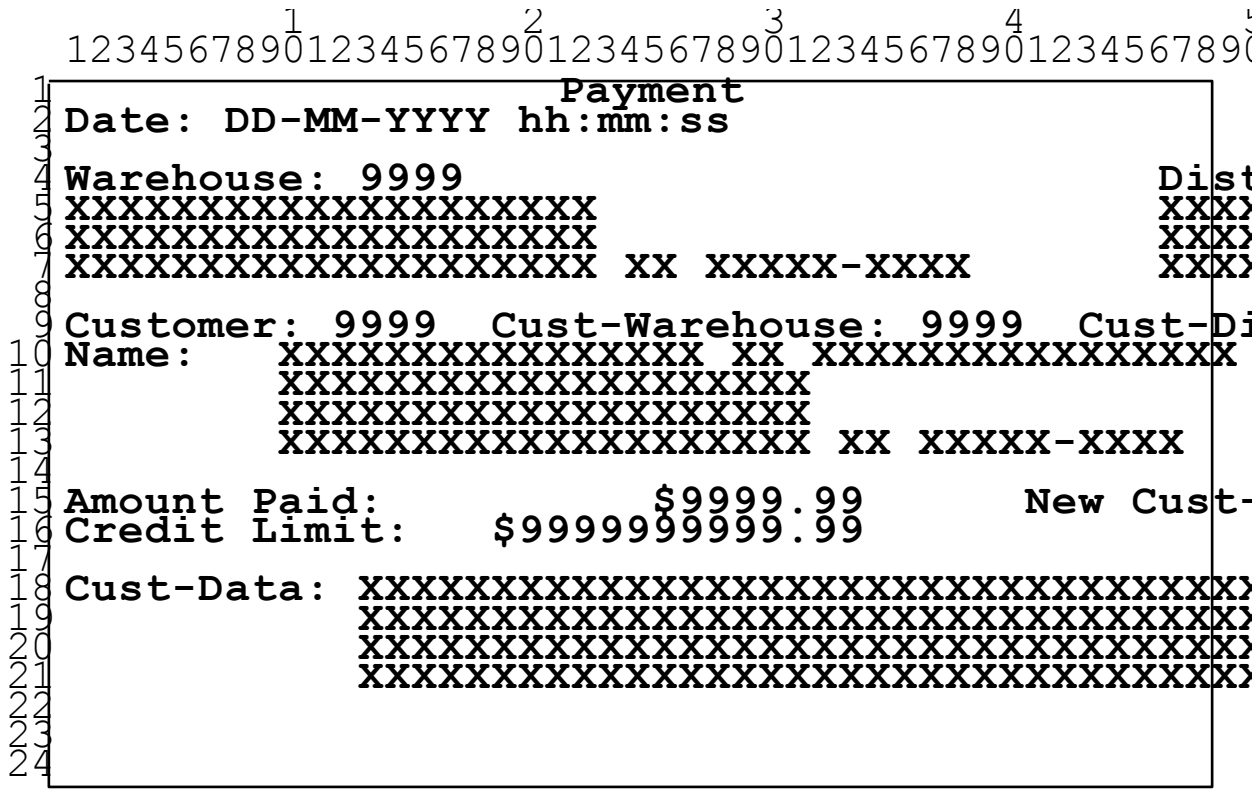
Intentionally ignored

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.



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.

2.6 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 *y* within [1 .. 100];

- If *y* <= 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 *y* > 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 *n*/ 2 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
1234567890123456789012345678901234567890123456789
                    Order-Status
Warehouse: 9999   District: 99
Customer: 9999    Name: XXXXXXXXXXXXXXX XX X
Cust-Balance: $-99999.99

Order-Number: 99999999    Entry-Date: DD-MM-Y
Supply-W      Item-Id     Qty       Amount
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
  9999        999999      99      $99999.99
```

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.

2.7 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 carried 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.

2.7.3.4 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 (*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 *threshold* are counted (giving *low_stock*).

  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.

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.

Comment: All required ACID tests must be performed on newly optimized binaries even if there have not been any source code changes.

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 1: 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.

Comment 2: For Consistency Conditions 2 and 4 (Clauses 3.3.2.2 and 3.3.2.4), sampling the first, last, and two random warehouses is sufficient.

3.3.2.1 Consistency Condition 1

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

W_YTD = 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(NO_O_ID) - min(NO_O_ID) + 1 = [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:

sum(O_OL_CNT) = [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:

W_YTD = sum(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:

D_YTD = sum(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 = sum(OL_AMOUNT) - 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:

(count(*) from ORDER) - (count(*) 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 = 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:

T_1 = New-Order transaction

T_2 = Payment transaction

T_3 = Delivery transaction

T_4 = Order-Status transaction

T_5 = 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 *x* and *y*, perform the following steps:

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

Case A, if transaction T3 stalls:

5A. Continue transaction T2 and verify that the price of items *x* (the second time) and *y* 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 *x* and *y*. 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 *x* and *y*. 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 *x* (the second time) and *y* match the values read by transaction T1. COMMIT transaction T2.

7C. Start a transaction T4. Query I_PRICE from items *x* and *y*. 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 *x* (the second time) and *y* match the values read by transaction T1. COMMIT transaction T2.

7D. Start a transaction T4. Query I_PRICE from items *x* and *y*. 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 2.

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 2.

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.

3.5.4 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.