
STANDARD SPECIFICATION

TPC BENCHMARK™ E

Draft Revision 0.32.2g

18 July, 2006

Transaction Processing Performance Council (TPC)

www.tpc.org

info@tpc.org

© 2006 Transaction Processing Performance Council

All Rights Reserved

Legal Notice

This document and associated source code (the “Work”) is a preliminary version of a benchmark specification being developed by the TPC. The Work is being made available to the public for review and comment only. The TPC reserves all right, title, and interest to the Work as provided under U.S. and international laws, including without limitation all patent and trademark rights therein.

No Warranty

- 1.1 TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE INFORMATION CONTAINED HEREIN IS PROVIDED “AS IS” AND WITH ALL FAULTS, AND THE AUTHORS AND DEVELOPERS OF THE WORK HEREBY DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY (IF ANY) IMPLIED WARRANTIES, DUTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, AND OF LACK OF NEGLIGENCE. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THE WORK.
- 1.2 IN NO EVENT WILL ANY AUTHOR OR DEVELOPER OF THE WORK BE LIABLE TO ANY OTHER PARTY FOR ANY DAMAGES, INCLUDING BUT NOT LIMITED TO THE COST OF PROCURING SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA, OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, ARISING IN ANY WAY OUT OF THIS OR ANY OTHER AGREEMENT RELATING TO THE WORK, WHETHER OR NOT SUCH AUTHOR OR DEVELOPER HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Trademarks

TPC Benchmark™ E and TPC-E are trademarks of the Transaction Processing Performance Council.

Acknowledgments

The TPC acknowledges the work and contributions of the TPC-E subcommittee member companies: AMD, Dell, Fujitsu-Siemens, HP, IBM, InfoSizing, Intel, Microsoft, NEC, Oracle, Sun, Sybase, and Unisys.

TPC-E Membership

AMD	InfoSizing	
Dell	Intel	Oracle
Fujitsu-Siemens	Microsoft	Sun
HP	Unisys	Sybase
IBM	NEC	

(As of April, 2006)

Document Revision History

Date	Version	Description
15-Jan-03	Draft 0.15	Submitted to TPC
15-Jun-03	Draft 0.16	Comments from Unisys and MS – as indicated in 5/28/03 minutes
20-Jun-03	Draft 0.17	Changes from June 03 meeting
18-July-03	Draft 0.18	Added meta-types, and made editorial changes
10-Aug-03	Draft 0.19	Made minor adjustments to for v18 changes based on Aug F-to-F meeting
10-Oct-03	Draft 0.20	Clause 0,1 clean up– wording for lobs and Clause 2 related fixes
10-Nov-03	Draft 0.21	Renamed the benchmark to TPC-E & major changes in Clause 3. Used yellow highlight to mark items that require future review. Made changes to Clause 5, which are not yet voted in. Unapproved wording in the spec is highlighted in blue. Made changes to Clause 4, which are not yet voted on.
19-Jan-04	Draft 0.22	Many editorial changes. Incorporated the new format for transaction profiles in 3.3.3 and 3.3.6 based on the votes takes in Dec 03 F2F.
16-Mar-04	Draft 0.23	Massive changes in Clause 3. Several changes to table definitions in Clause 2. Mix changes in Clause 5. This is a major change from v22.
14-Jun-04	Draft 0.24	Defined term “Transaction” changed to “transaction”. Wording and editorial changes from Francois were made to Clause 3. A Transaction Interfaces table was added to each transaction profile.
10-Aug-04	Draft 0.25	Many editorial changes. Added TRADE_T type definition. Added plus sign to transaction parameters that are to be returned from the Transaction Harness to the driver. Changed the way trade_dts was used in the Trade-Result Transaction Profile. Changed the steady state graph unit from 15 seconds to 30 seconds. Fixed the description of how ca_id was generated for the ACID transaction all through Clause 6. Dropped isolation test 6 which came from TPC-H. Changed the queries for testing database consistency. Made changes in the Durability tests. Made changes in the Durability tests. Updated the database footprints for some of the transactions. Added text to Clause 8.
05-Dec-04	Draft 0.26	Many editorial changes. Added Trade Lookup transaction. Changed Market Watch and Security Detail transactions. Modified Clause 8, added disk layout

		table. Added sample Numerical Quantities Summary in the appendix. Updated the table cardinality section. Added space requirements. Changed minimum load unit and increment unit from 100 customers to 1000 customers. Added wording for Active versus Configured users. Adding wording about RNGSEED usage and status return codes. Removed TR_ST_ID, TR_DTS, CT_CA_ID, SE_CA_ID. Changed H_BUY_DTS to H_DTS, B_NUM_TRAD_YTD to B_NUM_TRADES, B_COMM_YTD to B_COMM_TOTAL, T_TRAD_PRICE to T_TRADE_PRICE, FI_REV to FI_REVENUE, FI_LIAB to FI_LIABILITY, FI_INVENT to FI_INVENTORY, FI_DIVID to FI_DIVIDEND, NI_SRC to NI_SOURCE, NI_AUTH to NI_AUTHOR, and S_DIVID to S_DIVIDEND. Updated diagrams, changed format of Heading 1 and Appendix style.
05-Jan-05	Draft 0.27	Clean-up of editing styles and addition of new "Editing Styles" section in the introduction section of the document. Made many editorial changes. Changed constraints for S_YIELD and S_DIVIDEND to be not null. Changed wording for the database space requirements. Changed "should" to "must" to clarify some clauses. Changed Trade-Lookup frame 3 to select 80 rows not 20. Changed pseudo-code to be consistent. Dropped FI_DIVIDEND.
31-Jan-05	Draft 0.27FR	Addition of proposed text for the Audit Clause. More clean-ups of editing styles.
07-Feb-05	Draft 0.27	Made columns S_52WK_HIGH, S_52WK_HIGH_DATE, S_52WK_LOW, and S_52WK_LOW_DATE not null, per motion 189. Per motions 190, 192 changed Appendix A for CO_OPEN_DATE, CR_FROM_QTY, CR_TO_QTY. Motion 193, changed TT_NAME from char(30) to char(12), and ST_NAME from char(30) to char(10). Changed Security-Detail transaction profile to use the access_lob_flag. Changed Trade-Lookup profile to remove st_completed_id, and to remove TRADE_HISTORY from frame 4 and to fetch 20 not 80 rows in frame 3. Changed Market-Feed transaction to remove trade_dts. Made changes to the Cardinality section. Added extra level 3 heading in clause 2.6. It seems Francois changed the appendices to clause 10 through 14. This has broken some cross references and the table of contents. Added some lines in Clause 9, to remind us that we want to check the CE and MEE input files and the MEE base time. Re-did clause 5.3.1 Input Parameters. Changed clause 4 to talk about CMEE rather than CMarketEmulator. Worked on highlighting some of the defined terms that lost their highlights with Francois's style changes.
11-Apr-05 to 31-May-05	Draft 0.28	More style changes from Francois. Added HOLDING_SUMMARY table and changed transactions to use the new table. Added some pricing wording since the TPC Pricing specification passed. Modified diagrams, added possible configuration diagrams to clause 4. Added definitions appendix. Added executive summary to the appendix. Added Rick Freeman's comments on durability in clause 6. Changed the isolation table to reflect the isolation matrix. Removed the ACID query and ACID transaction. Dropped TIMEOFDAY. Added FI_DIVIDEND back in. Changed DriverInput to CETxnInputGenerator. Changed transaction input/output structure names and frame input/output structure names to match the new names in TxnHarnessStructs.h. Changed clause 4 to talk about the new CCE and CDM classes. Added wording under review in pink to several clauses. After the face-to-face meeting, removed two appendices and replaced them with the developers guide appendix. Modified the network requirements wording in Clause 4. Dropped FI_DIVIDEND for real this time. Clarified the substitution rules. Added new run time cardinality section. Changed 40 hours initially loaded to 400 hours initially loaded. Added changes from the face-to-face meeting. Moved the Defined Terms from an appendix to Clause 1.1. Bookmarked the definition of a defined term in the later clauses, and referenced that bookmark in the defined terms section. Dropped Active and Inactive Customers concept. Dropped COMPANY_INDUSTRY table, dropped CO_SC_ID column, added CO_IN_ID column, added IN_SC_ID column. Re-did scaling and fixed tables in the cardinality section, because now COMPANY and SECURITY related tables scale. Changed Trade-Lookup to 30:30:10:30 and to bring back multiple rows. Changed Data-Maintenance watch list item from "AAA" to "AA" to match the symbols that are available. Changed column width for CO_NAME, S_NAME, S_SYMB, H_S_SYMB, HS_S_SYMB, WI_S_SYMB, T_S_SYMB, TR_S_SYMB, DM_S_SYMB, and LT_S_SYMB because they needed to be bigger so that SECURITY and COMPANY could scale. In clause 4 added wording about not changing EGen code. Changed S_PRICE_T from NUM(12,2)

		to NUM(8,2). Added metatype S_QTY_T. Changed H_QTY, HH_AFTER_QTY, HH_BEFORE_QTY, HS_QTY, T_QTY, TR_QTY, CR_FROM_QTY, CR_TO_QTY from S_COUNT_T to S_QTY_T. Changed CH_CHRG, CT_AMT, SE_AMT, T_CHRG, T_COMM, T_TAX from BALANCE_T to VALUE_T. Moved 60-day space out of clause 2.6.4 into clause 7. Moved STATUS_TYPE from market table to dimension table. Changed Data-Maintenance CUSTOMER_TAXRATE variant. Changed atomicity test to only do Trade-Order, not Trade-Order and Trade-Result. Changed IDENT_T from NUM(9) to NUM(12) this is to allow company to scale with customer. Changed S_SYMB etc. from CHAR(7) to CHAR(13) to CHAR(15). Added account index to Customer-Position transaction parameters changed Customer-Position frame 2 to use an account ID rather than a customer ID. Added Business descriptions for each of the transactions. Made editorial changes to multi-row Trade-Lookup. Added Trade-Update transaction. Added definition for Primary Key and Foreign Key.
05-Aug-05	Draft 0.29	Changed acct_index to acct_id_idx in Customer-Position Transaction input parameters. Modified the wording in clause 2.7.1.1 to talk about input and output flat files. Made numerous editorial changes. Changed TRADE_T from NUM(18) to NUM(15). Changed IDENT_T from NUM(12) to NUM(11). Changed the symbols added to a watch list in Data-Maintenance from ZTEL and ZYI to ZAPS and ZONS. Changed the Sell Short pseudo code in Trade-Result Frame 2 to use (-1) * needed_qty. Added P1-b phenomena. Changed some "RT" to "Response Time". Worked on wording for Scale Factor and Initial Trade Days. Added wording to say processors, cores and threads must be reported. Changed CPU to processor. Removed isolation levels from clause 3. Isolation is described in clause 6. Added definition of Ramp-up time. Move rules for multiple results from one Test Run from clause 4 to clause 8. Move partitioning rules from clause 4 to clause 5. Removed transaction output logging requirements from clause 3, reporting requirements are covered in clause 8. Added extra consistency conditions to clause 6, and changed the definition from pseudo code to English. Clarified some items in clause 5, changed the definition of Test Run and Ramp-up. Added a definition for Ramp-down. Changed the Steady State Graph to a Test Run Graph. Changed clause 5.3.1. Added wording about allowable variances for transaction input mixes. Modified the durability section in clause 6. Consolidated the executive summary and numerical quantities sheets into one appendix. Replaced EGen appendices with EGen Users Guide appendix. Updated cross references, fixed errors from broken references. Scanned document for defined terms that were not bolded, and bolded them. Added EGen Clause from Doug.
18-Aug-05	Draft 0.30	Replaced Executive Summary with a new one from Jamie. Replaced Isolation section with new Isolation section written by Francois. Re-ordered and re-wrote the durability clause again. Reduced the consistency tests from 11 tests to 4 tests. Changed when the consistency tests have to be run. Received approval or addressed issues with some highlighted sections so that they could be un-highlighted. Removed wording from various areas of the specification because that wording was now in the new EGen clause. Added some EGen reporting requirements. Modified Trade-Result frame 6 to return CA_BAL as a frame output parameter. This change was needed for an isolation test. Added EGen defined terms to Clause 1 and to the EGen Clause. Added Doug's updates to the EGen Clause and appendix based on feedback from the face-to-face meeting. Removed "TPC CONFIDENTIAL" from the cover. Added items to the Audit Check List, previously it did not have an "EGen Related Items" section.
04-Oct-05	Draft 0.31	Changed version number and date, changed "Public Draft" to "Company Review". Fixed the formatting of some of the defined terms. Added "referential integrity" as a defined term. Changed the definition of load unit. Removed CA_BAL consistency test which was pending a code change. The subcommittee decided we did not need the test. Removed dead body trade types (Cover-Buy and Short-Sell). Changed Market-Feed transaction profile to pass out the trade type of a triggered Limit-Trade. Changed the description of how broker names are selected as inputs for the Broker-Volume transaction.
14-Nov-05 to 15-Dec-05	Draft 0.31	Changed version number and date. Fixed formatting issues in Word document. Added L3 header in Appendix. Added "Database session" definition. Added Clause 1.2 Model Description to explain trading and how customer accounts are

		<p>set up. The description is to aid readers not familiar with TPC-E. Changed the definition of load unit. Added wording to allow EGenInputFiles on the Driver even though they are copies of data in the database. Changed to the case of "MARGIN" and "CASH ACCOUNT" in the pseudo code for Trade-Result frame 6. Changed initial cardinality of TRADE_TYPE from 7 to 5, COMMISSION_RATE from 336 to 240, CHARGE from 21 to 15. Added text to clause 6.2 to support motion 303 "Errors are not allowed during the Measurement Interval". Francois added the approved wording for the Isolation tests. Added wording to say that the TRADE_REQUEST table should be cleaned out between runs, but not during a durability test. Added a definition for DM. Added wording in Clause 5 that a version of EGenLoader that is compliant with the specification the result is published under must be used. Added wording to say that EGen is presumed to be correct, but may not meet the criteria in clause 2 and 6 thus making the database or run invalid. Changed 90th percentile time for Trade-Order from 1 to 2 seconds, for Trade-Status from 3 to 1 second and for Market-Feed from 1 to 2 seconds (Motion 289). Added max_acct_id to Trade-Lookup parameters and frame 4 parameters and Trade-Update parameters and frame 3 parameters for active versus configured customers changes. Changed the description of how customer IDs and account IDs are generated. Previously parameter descriptions for acct_id in various transaction and frame parameters, said that customer identifiers and account identifier were selected with a uniform distribution. Changed clause 5.3 and 5.9 and 5.10 to follow motions made at the face-to-face meeting regarding EGen bugs. Changed Clause 4.4.3 to no longer allow operator intervention. This is following the changes TPC-C made to version 5.6 of their specification. Changed Initial Trade Days from 50 to 100. Switched Trade-Lookup frames 3 and 4 so that frames 1, 2 and 3 agree with Trade-Update frames 1, 2, 3. Added Trade-Cleanup transaction.</p>
15-Dec-05 to 02-March-06	Draft 0.31	<p>Francois re-wrote clause 10. Made lots of editorial changes based on feed back from Larry, Matt, Zach in IBM, and Francois and Cecil. Modified initial database cardinality for HOLDING, HOLDING_HISTORY, HOLDING_SUMMARY with 100 initial trade days. Modified runtime database size requirements for HOLDING, HOLDING_HISTORY now that we have 100 initial trade days. Changed DB to Database. Modified the Market-Feed transaction to bring send_to_market into the frame implementation and out of the transaction harness. This change was made to avoid over-running the output array. Changed the HOLDING table's foreign keys to reference the HOLDING_SUMMARY table. Re-arranged some code in Trade-Result to reflect the HOLDING/HOLDING_SUMMARY change. Updated Clause 5 with Matt Emmerton's changes. Added to the description of some frame output parameters that the output string from the database should be stripped of trailing blanks so that EGen TxnHarness code would do string comparisons correctly.</p>
21-March-06	Draft 0.31	<p>Applied review comments to Clause 10. Modified wording in clause 10.2.5.6. Merged clause 6.3.2 and 6.3.3 (transaction input parameter mix) into a table. Applied Cecil's wording changes to clause 5.3.1. Modified clause 9 with Charles's bag-o-bits changes. Added time definition in datetime. Fixed some bolding in clause 2.2.1.1. Applied motion 345 mix changes. Added Matt Emmerton's changes to clause 5 and 6. Tables are now used in Clause 5 to list the parameters and what constitutes a valid input. Some wordsmithing changes were made to Clause 6.1 (Input values) to refer to Clause 5 and to retain the wording about using the same input files for all instances. Also in clause 6 the CE partitioning clause was re-worded and an example was provided.</p>
12-April-06	Draft 0.32.0	<p>Updated version number, document date and TPC-E member companies. Set scale factor as 500 and initial trade days as 300. Updated cardinality tables in Clause 2.6 with 300 initial trade days figure. Updated compatible EGen version from 3.12 to 3.14. Removed "-a" parameter from table in Clause 5.7.2. Re-formatted the tables in Clause 5.8. Used the term EGenInputFiles in the second bullet in clause 6.3.1.1. Fixed values that were miss-typed in the table in Clause 6.3.2. Added a requirement that Load Units are driven evenly to Clause 6.3.3. Checked that Clause 10 covered this new requirement. Applied some of Cecil's Clause 5 review comments. Changed cell style in the transaction database footprint tables to make each cell use the correct size font. Added CTradeCleanup to definitions section, Clause 5.2.5 and Appendix A.11. Added</p>

		custom loader information to Appendix A.6.7. Modified Clause 5, Clause 9 and Appendix A at the face-to-face meeting.
02-May-06	Draft 0.32.2c	Replaced the Grant of Rights to one suitable for public review. Updated the cross-references. Fixed the paragraph style of the first paragraph in Clause 8. Removed the yellow from Clause 6.3.2.4 (comment 1215). Replaced the sub clause numbers on EGenDriverCE, EGenDrvierMEE, and EGenDriverDM with bullets in the EGenDriver definition in Clause 1.1.
05-May-06	Draft 0.32.2d	Changed Clause 8.4.3 to say "period of 60-days". Defined Initial Database Size. In Clause 7.4 changed Trade-Request to Trade-Order. Changed bookmark cross reference to clause number cross reference in clause 7.5.2.2 and clause 7.5.5. Changed the Arbitrary Transaction definition to reference clause 7.4.1.3. Moved Trade-Update after Trade-Order to maintain the alphabetical order of the table in clause 6.1.6.2 (Transaction Inputs Mix). Un-highlighted EGen 3.14 in Clause 5.1.2.1 (comment 1225). Fixed wording in Clause 6.3.1.4 (comment 1227). Removed the level 4 clause number from the "Transaction Mix" definition in Clause 1.1 (comment 1229). Made the major clause headings consistent (comment 1219). Changed level 4 text style to remove the indent (comment 1232). Bolded defined terms in Appendix A.13 (comment 1220). Changed level 3 text 1.3.2 to level 4 text 1.3.1.1 (comment 1230). Removed numbering on paragraph following Clause 2.2 (comment 1231). Renumbered Clause 6 and Clause 5. Renumbered Clause 4 (comment 1233). Added color code for the diagrams to the front of the specification and to the diagrams (comment 1217). Summary: addressed comments 1215, 1217, 1219, 1220, 1221, 1225, 1227, 1229, 1230, 1231, 1232, 1233, 1234, 1235 and 1248.
23-May-06	Draft 0.32.2e	Addressed comments. 1228 "Durability Test for Business Recovery", 1236 "Definitions", reverse cross references, original in definitions section. 1237 "Definitions - Numbers and A", 1238 "Definitions - B", 1239 "7.5.1 end of Business Recovery", 1240 "3.3.12 Trade-Cleanup", 1242 "7.5.6.1 Add Business Recovery Time Definition", 1243 "Definitions - Customer-Emulator" 1244 "Definition - Committed Property" 1246 "Definitions - D and E", 1247 "Definitions - D", 1250 "Table formatting issues", 1251 "Clause 3.3.1.8 Harness Code for Trade-Order Frame 6", 1252 "Customer-Position Frame 3 pseudo code", 1254 "Clause 2.3 numbering", 1255 "Move clause 3.2.2 (Database-Footprint Definition)", 1256 "Clause 1.3.3.1 Market/Limit Order Percentages", 1257 "Data Type and Meta-type definitions", 1259 "Need requirement that CCE RNG seeds be unique",
02-June-06	Draft 0.32.2f	Addressed comments:- 1253 "Trade-Cleanup Cleanup - Replaces ID 1240" 1258 "Clause 2.3.7 DATETIME as Two Attributes" 1270 "Cardinality of STATUS_TYPE table" 1271 "EGen transaction names and parameters should match spec". 1272 "MaxRowsToUpdate value" 1275 "Minimum Database Cardinality of 5000 customers" also added minimum partition size of 5,000 customers to clause 6.3.2.1.

		<p>1277 "EGen requires ISO-compliant C/C++ compiler"</p> <p>1280 "Exactly 5000 Accounts per LU due to ID 1269"</p> <p>1281 "Clause 2.7.2 Over scaling the database"</p> <p>1286 "Report EGen Constant Usage in Spec"</p>
16-June-06 to 18-July-06	Draft 0.32.2g	<p>Addressed comments:-</p> <p>1216 "Trades per Load Unit Graph requirements"</p> <p>1273 "New "end_trade_dts" Input for Trade-Lookup and Trade-Update"</p> <p>1282 "Spec. must specify which parameters can be changed by Test Sponsor"</p> <p>1283 "Add EGenLogger to Clause 5, Constant inputs to Clause 6"</p> <p>1293 "Lack of EGenLogger output verification"</p> <p>1294 "Clause 2.6 Fixed/Scaling/Growing Tables"</p> <p>1299 "Trade-Result Frame 3 parameters"</p> <p>1300 "Buy on Margin % or Range"</p> <p>1301 "6.3.1.2 Comment at end Table"</p> <p>1305 "max_acct_id input to TL/TU F3"</p> <p>1308 "number noun to number-noun"</p> <p>1310 "Data-Maintenance transaction"</p> <p>The subcommittee rejected items 4, 6 and 7.</p> <p>Item 1, Trade-Order frame 2 return ap_acl instead of bad_permission</p> <p>Item 2, Data-Maintenance, add a new transaction and frame input parameter acct_id. Use acct_id instead of c_id for updating ACCOUNT_PERMISSION.</p> <p>Item 3, Data-Maintenance when updating ADDRESS, 33% of the time update a company address and the rest of the time update a customer's address.</p> <p>Item 5, replace Data-Maintenance transaction input and frame input parameter "add_flag" with the more generic "vol_incr".</p> <p>1316 "Detecting/Preventing time based games with tpsE and customer partitioning"</p> <p>1324 "EGen datatypes mentioned in spec."</p> <p>Accepted Francois's Business Cycle and Clause 6 changes.</p> <p>Applied Cecil's comments on Francois's Business Cycle (Day) changes.</p> <p>Addressing comments 1216 and 1316 involved adding EGenTester.</p> <p>Applied Wayne's editorial comments.</p>

Typographic Conventions

The following typographic conventions are used in this specification:

Convention	Description
------------	-------------

Bold	Bold type is used to highlight terms that are defined in this document
<i>Italics</i>	Italics type is used to highlight a variable that indicates some quantity whose value can be assigned in one place and referenced in many other places.
Yellow Highlight (Draft versions only)	Yellow highlight marks wording that may change in the future. For example, clause numbers and references will change.
Pink Highlight (Draft versions only)	Pink highlight marks wording that is in flux or under construction and will be changing.
UPPERCASE	Uppercase letters indicate database schema object names such as table and column names. In addition, most acronyms are in uppercase.

Diagram Color-Coding Conventions

Concept

- Customer Light Green
- Broker Pale Blue
- Market Rose

Implementation

- TPC Provided Code Light Turquoise
- Sponsor Provided Code Lavender
- Commercially Available Product Light Yellow

Table of Contents

<i>0.1</i>	<i>Introduction</i>	<i>1</i>
0.1.1	Goal of this Benchmark	1
0.1.2	Restrictions and limitations	2
<i>0.2</i>	<i>General Implementation Guidelines.....</i>	<i>2</i>
<i>0.3</i>	<i>General Measurement Guidelines.....</i>	<i>3</i>
<i>1.1</i>	<i>Definitions</i>	<i>5</i>
<i>1.2</i>	<i>Business and Application Environment.....</i>	<i>27</i>
<i>1.3</i>	<i>Model Description.....</i>	<i>29</i>
1.3.1	Entity Relationships.....	29
1.3.2	Differences between customer tiers.....	29
1.3.3	Trade types.....	29
1.3.4	Effects of trading on holdings.....	30
<i>2.1</i>	<i>Database Entities, Relationships, and Characteristics</i>	<i>31</i>
<i>2.2</i>	<i>Database Schema and Table Definitions.....</i>	<i>31</i>
2.2.1	Data Type Definitions.....	31
2.2.2	Meta-type Definitions	32
2.2.3	General Schema Items	33
2.2.4	Customer Tables	34
2.2.4.1	ACCOUNT_PERMISSION.....	34
2.2.4.2	CUSTOMER.....	34
2.2.4.3	CUSTOMER_ACCOUNT.....	35
2.2.4.4	CUSTOMER_TAXRATE.....	36

2.2.4.5	HOLDING.....	36
2.2.4.6	HOLDING_HISTORY	36
2.2.4.7	HOLDING_SUMMARY	37
2.2.4.8	WATCH_ITEM	37
2.2.4.9	WATCH_LIST.....	37
2.2.5	Broker Tables.....	38
2.2.5.1	BROKER.....	38
2.2.5.2	CASH_TRANSACTION	38
2.2.5.3	CHARGE	38
2.2.5.4	COMMISSION_RATE.....	39
2.2.5.5	SETTLEMENT	39
2.2.5.6	TRADE	40
2.2.5.7	TRADE_HISTORY	40
2.2.5.8	TRADE_REQUEST.....	41
2.2.5.9	TRADE_TYPE	41
2.2.6	Market Tables	42
2.2.6.1	COMPANY	42
2.2.6.2	COMPANY_COMPETITOR	42
2.2.6.3	DAILY_MARKET.....	43
2.2.6.4	EXCHANGE.....	43
2.2.6.5	FINANCIAL	43
2.2.6.6	INDUSTRY.....	44
2.2.6.7	LAST_TRADE	45
2.2.6.8	NEWS_ITEM.....	45
2.2.6.9	NEWS_XREF	45
2.2.6.10	SECTOR	46
2.2.6.11	SECURITY	46
2.2.7	Dimension Tables	47
2.2.7.1	ADDRESS.....	47
2.2.7.2	STATUS_TYPE.....	47
2.2.7.3	TAXRATE	47
2.2.7.4	ZIP_CODE.....	47
2.3	<i>Implementation Rules</i>	<i>48</i>
2.3.3	Table Partitioning	48
2.4	<i>Integrity Rules</i>	<i>50</i>
2.5	<i>Data Access Transparency Requirements</i>	<i>51</i>
2.6	<i>Database Size and Table Cardinality.....</i>	<i>52</i>
2.6.1	Initial Database Size Requirements	52
2.6.2	Runtime Database Size Requirements	55
3.1	<i>Introduction</i>	<i>57</i>
3.1.1	Definitions	57
3.1.2	Database-Footprint Definition	58
3.2	<i>Transaction Implementation Rules.....</i>	<i>60</i>
3.2.1	Frame Implementation.....	60
3.3	<i>The Transactions</i>	<i>61</i>
3.3.1	The Trade-Order Transaction	62
3.3.1.1	Trade-Order Transaction Parameters	63
3.3.1.2	Trade-Order Transaction Database-Footprint	64
3.3.1.3	Trade-Order Transaction Frame 1 of 6.....	65
3.3.1.4	Trade-Order Transaction Frame 2 of 6.....	67
3.3.1.5	Trade-Order Transaction Frame 3 of 6.....	68
3.3.1.6	Trade-Order Transaction Frame 4 of 6.....	75

3.3.1.7	Trade-Order Transaction Frame 5 of 6.....	78
3.3.1.8	Trade-Order Transaction Frame 6 of 6.....	78
3.3.2	The Trade-Result Transaction.....	79
3.3.2.1	Trade-Result Transaction Parameters.....	80
3.3.2.2	Trade-Result Transaction Database-Footprint.....	81
3.3.2.3	Trade-Result Transaction Frame 1 of 6.....	82
3.3.2.4	Trade-Result Transaction Frame 2 of 6.....	84
3.3.2.5	Trade-Result Transaction Frame 3 of 6.....	92
3.3.2.6	Trade-Result Transaction Frame 4 of 6.....	94
3.3.2.7	Trade-Result Transaction Frame 5 of 6.....	95
3.3.2.8	Trade-Result Transaction Frame 6 of 6.....	96
3.3.3	The Trade-Lookup Transaction.....	99
3.3.3.1	Trade-Lookup Transaction Parameters.....	99
3.3.3.2	Trade-Lookup Transaction Database-Footprint.....	101
3.3.3.3	Trade-Lookup Transaction Frame 1 of 4.....	101
3.3.3.4	Trade-Lookup Transaction Frame 2 of 4.....	104
3.3.3.5	Trade-Lookup Transaction Frame 3 of 4.....	106
3.3.3.6	Trade-Lookup Transaction Frame 4 of 4.....	109
3.3.4	The Trade-Update Transaction.....	111
3.3.4.1	Trade-Update Transaction Parameters.....	111
3.3.4.2	Trade-Update Transaction Database-Footprint.....	112
3.3.4.3	Trade-Update Transaction Frame 1 of 3.....	113
3.3.4.4	Trade-Update Transaction Frame 2 of 3.....	116
3.3.4.5	Trade-Update Transaction Frame 3 of 3.....	119
3.3.5	The Trade-Status Transaction.....	123
3.3.5.1	Trade-Status Transaction Parameters.....	123
3.3.5.2	Trade-Status Transaction Database-Footprint.....	124
3.3.5.3	Trade-Status Transaction Frame 1 of 1.....	124
3.3.6	The Customer-Position Transaction.....	126
3.3.6.1	Customer-Position Transaction Parameters.....	127
3.3.6.2	Customer-Position Database-Footprint.....	128
3.3.6.3	Customer-Position Transaction Frame 1 of 3.....	129
3.3.6.4	Customer-Position Transaction Frame 2 of 3.....	132
3.3.6.5	Customer-Position Transaction Frame 3 of 3.....	133
3.3.7	The Broker-Volume Transaction.....	134
3.3.7.1	Broker-Volume Transaction Parameters.....	134
3.3.7.2	Broker-Volume Transaction Database-Footprint.....	135
3.3.7.3	Broker Volume Transaction Frame 1 of 1.....	135
3.3.8	The Security-Detail Transaction.....	136
3.3.8.1	Security-Detail Transaction Parameters.....	136
3.3.8.2	Security-Detail Transaction Database-Footprint.....	137
3.3.8.3	Security Detail Transaction Frame 1 of 1.....	139
3.3.9	The Market-Feed Transaction.....	144
3.3.9.1	Market-Feed Transaction Parameters.....	144
3.3.9.2	Market-Feed Database-Footprint.....	145
3.3.9.3	Market-Feed Transaction Frame 1 of 1.....	146
3.3.10	The Market-Watch Transaction.....	149
3.3.10.1	Market-Watch Transaction Parameters.....	149
3.3.10.2	Market-Watch Transaction Database-Footprint.....	150
3.3.10.3	Market-Watch Transaction Frame 1 of 1.....	151
3.3.11	The Data-Maintenance Transaction.....	154
3.3.11.1	Transaction Parameters.....	155
3.3.11.2	Data-Maintenance Transaction Database-Footprint.....	156
3.3.11.3	Data-Maintenance Transaction Frame 1 of 1.....	158
3.3.12	The Trade-Cleanup Transaction.....	168
3.3.12.1	Trade-Cleanup Transaction Parameters.....	169

3.3.12.2	Trade-Cleanup Transaction Database-Footprint	169
3.3.12.3	Trade-Cleanup Transaction Frame 1 of 1	169
4.1	<i>Overview</i>	173
4.1.1	Description of the Real-World OLTP Environment	173
4.1.2	Functional Component Abstraction of the Real-World OLTP Environment	174
4.1.3	Distillation of Functional Components into the TPC-E Environment	175
4.2	<i>Driver & System Under Test (SUT) Definitions</i>	179
4.3	<i>Example Test Configuration Implementations</i>	180
4.4	<i>Further Requirements for SUT and Driver Implementations</i>	183
4.4.1	Restrictions on the Driver	183
4.4.2	Disclosure of Network Configuration	184
4.4.3	SUT Implementation Limits on Operator Intervention	184
5.1	<i>Overview</i>	185
5.2	<i>EGen Terms</i>	185
5.3	<i>Compliant EGen Versions</i>	186
5.3.5	Addressing Errors in EGen	186
5.3.6	Process for Reporting Issues with EGen	187
5.3.6.1	Portability Issues	187
5.3.6.2	Other Issues	187
5.3.7	Submitting EGen Enhancement Suggestions	187
5.4	<i>EGenProjectFiles</i>	188
5.5	<i>EGenInputFiles</i>	188
5.6	<i>EGenSourceFiles</i>	188
5.7	<i>EGenLoader</i>	188
5.8	<i>EGenDriver</i>	189
5.8.4	EGenDriverCE	189
5.8.5	EGenDriverMEE	189
5.8.6	EGenDriverDM	189
5.9	<i>EGenTxnHarness</i>	189
5.10	<i>EGenTester</i>	189
6.1	<i>Introduction</i>	190
6.1.1	Definition of Terms	190
6.2	<i>Transaction Mix</i>	190
6.2.1	Mix Control	190
6.2.2	Mix Requirements	191
6.3	<i>Input Parameters</i>	192
6.3.1	Inputs to EGenDriver Code	192
6.3.2	EGenDriverCE partitioning	194
6.4	<i>Response Time and Pacing Delays</i>	195
6.4.1	Response Time	195
6.4.2	Pacing Delay	198
6.5	<i>Test Run</i>	198
6.5.1	Definition of Terms	198
6.5.2	Database Content	199

6.5.3	Sustainable Performance.....	199
6.5.4	Steady State	200
6.5.5	Measurement Interval	200
6.5.6	Database Growth	200
6.5.7	Performance & Database Size	202
6.5.8	Throughput Rating	202
6.6	<i>Required Reporting</i>	203
6.6.1	Test Run Graph.....	203
6.6.2	Primary Metrics	203
6.6.3	EGenTester results	204
7.1	<i>ACID Properties</i>	206
7.2	<i>Atomicity Requirements</i>	206
7.2.1	Atomicity Property Definition	206
7.2.2	Atomicity Tests.....	207
7.3	<i>Consistency Requirements</i>	207
7.3.1	Consistency Property Definition.....	207
7.3.2	Consistency Conditions	207
7.3.2.1	Consistency condition 1	207
7.3.2.2	Consistency condition 2	207
7.3.2.3	Consistency condition 3	207
7.3.3	Consistency Tests	207
7.4	<i>Isolation Requirements</i>	208
7.4.1	Isolation Property Definition	208
7.4.2	Isolation Tests.....	209
7.4.2.1	P3 Test in Read-Write	209
7.4.2.2	P2 Test in Read-Write	210
7.4.2.3	P1 Test in Read-Write	211
7.4.2.4	P1 Test in Read-Only	211
7.5	<i>Durability and Data Accessibility Requirements</i>	212
7.5.1	Definition of Terms	212
7.5.2	Data Accessibility	213
7.5.2.1	Redundancy Levels	214
7.5.2.2	Durability test procedure for data accessibility	214
7.5.2.3	Reported Metrics for data accessibility	214
7.5.2.4	Data Accessibility Graph.....	215
7.5.3	Business Recovery	215
7.5.4	List of Single Failures.....	215
7.5.5	Durability test procedure when Business Recovery is needed.....	216
7.5.6	Required Reporting for Business Recovery tests.....	217
7.5.6.1	Reported Metrics	217
7.5.6.2	Business Recovery Time Graph	217
8.1	<i>Priced Configuration</i>	218
8.2	<i>On-line Storage Requirement</i>	218
8.2.1	Continuous Operation Requirement	218
8.2.2	60-Day Data Space	219
8.2.3	Archive Operation Requirement.....	219
8.2.4	Back-up Storage Requirements.....	219
8.3	<i>TPC-E Specific Pricing Requirements</i>	220
8.3.1	Additional Operational Components	220
8.3.2	Additional Software.....	220

8.4	<i>Component Substitution</i>	220
8.5	<i>Required Reporting</i>	221
9.1	<i>Full Disclosure Report Requirements</i>	222
9.1.1	General Items.....	222
9.1.2	Executive Summary Statement	222
9.1.3	Report	223
9.1.4	Supporting Files.....	224
9.2	<i>Additional Disclosure Requirements</i>	225
9.2.1	Clause 2 Database Design, Scaling & Population Related Items.....	225
9.2.2	Clause 3 Transaction Related Items.....	227
9.2.3	Clause 4 SUT, Driver, and Network Related Items	227
9.2.4	Clause 5 EGen Related Items	228
9.2.5	Clause 6 Performance Metrics and Response Time Related Items.....	228
9.2.6	Clause 7 Transaction and System Properties Related Items	230
9.2.7	Clause 8 Pricing Related Items	230
10.1	<i>General Rules</i>	231
10.1.1	Audit Requirements	231
10.2	<i>Audit Check List</i>	233
10.2.1	Auditing the Database	233
10.2.2	Auditing the Transactions	234
10.2.3	Auditing the SUT, Driver and Networks	236
10.2.4	Auditing EGen	236
10.2.5	Auditing the Execution Rules and Metrics	237
10.2.6	Auditing the ACID Tests	239
10.2.7	Auditing the Pricing	240
10.2.8	Auditing the FDR.....	240
	Appendix A. EGen User's Guide	242
A.1	<i>Overview</i>	242
A.2	<i>EGen Directory</i>	242
A.3	<i>EGenProjectFiles</i>	243
A.4	<i>EGenInputFiles</i>	243
A.5	<i>EGenSourceFiles</i>	243
A.6	<i>EGenLogger</i>	243
A.7	<i>EGenLoader</i>	243
A.8	<i>EGenDriver</i>	244
A.9	<i>Implementing a CE using EGenDriverCE</i>	245
A.10	<i>Implementing a MEE using EGenDriverMEE</i>	245
A.11	<i>Implementing a Data-Maintenance Generator using EGenDriverDM</i>	246
A.12	<i>EGenTxnHarness</i>	246
A.13	<i>Functional Implementation</i>	247
	Appendix B. Executive Summary Statement	250
B.1	<i>Layout Requirements</i>	250

B.2	<i>Sample Executive Summary Statement</i>	252
-----	---	-----

Table of Figures

<i>Figure 1.a - Business Model Transaction Flow</i>	27
<i>Figure 1.b - Application Components</i>	28
<i>Figure 3.a - Frames interfacing with the Harness and the Database</i>	57
<i>Figure 4.a - Diagram of the Real-World OLTP Environment</i>	173
<i>Figure 4.a - Abstraction of the Functional Components in an OLTP Environment</i>	174
<i>Figure 4.a - Functional Components of the Test Configuration</i>	176
<i>Figure 4.a - Defined Components of the Test Configuration</i>	179
<i>Figure 4.a - Sample Component of Physical Test Configuration</i>	180
<i>Figure 4.a - Separate Driver with combined Tier A and Tier B</i>	181
<i>Figure 4.b - Driver and Tier A combined, separate Tier B</i>	182
<i>Figure 4.c - Combined Driver, Tier A and Tier B</i>	183
<i>Figure 6.a - Transactions Selection and Processing</i>	191
<i>Figure 6.a - Measuring Response Time</i>	197
<i>Figure 6.a - Example of the Measured Throughput versus Elapsed Time Graph</i>	203
<i>Figure 9.a - Example of Measured Benchmark Configuration</i>	224
<i>Figure 10.a - High Level Overview of a Sample Implementation</i>	247

CLAUSE 0 -- PREAMBLE

0.1 Introduction

TPC Benchmark™ E (TPC-E) is an On-Line Transaction Processing (OLTP) workload. It is a mixture of read-only and update intensive transactions that simulate the activities found in complex OLTP application environments. The database schema, data population, transactions, and implementation rules have been designed to be broadly representative of modern OLTP systems. The benchmark exercises 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;
- Moderate system and application execution time;
- A balanced mixture of disk input/output and processor usage;
- Transaction integrity (ACID properties);
- A mixture of uniform and non-uniform data access through primary and secondary keys;
- Databases consisting of many tables with a wide variety of sizes, attributes, and relationships with realistic content;
- Contention on data access and update.

The TPC-E operations are modeled as follows:

- The database is continuously available 24 hours a day, 7 days a week, for data processing from multiple **sessions** and data modifications against all tables, except possibly during infrequent (e.g., once a month) maintenance **sessions**.
- Due to the worldwide nature of the application modeled by the TPC-E benchmark, any of the transactions may be executed against the database at anytime, especially in relation to each other.

0.1.1 Goal of this Benchmark

The TPC-E benchmark simulates the OLTP workload of a brokerage firm. The focus of the benchmark is the central database that executes transactions related to the firm's customer accounts. In keeping with the goal of measuring the performance characteristics of the database system, the benchmark does not attempt to measure the complex flow of data between multiple application systems that would exist in a real environment.

The mixture and variety of transactions being executed on the benchmark system is designed to capture the characteristic components of a complex system. Different transaction types are defined to simulate the interactions of the firm with its customers as well as its business partners. Different transaction types have varying run-time requirements.

The benchmark defines:

- Two types of transactions to simulate Consumer-to-Business as well as Business-to-Business activities
- Several transactions for each transaction type
- Different execution profiles for each transaction type
- A specific run-time mix for all defined transactions

For example, the database will simultaneously execute transactions generated by systems that interact with customers along with transactions that are generated by systems that interact with financial markets as well as administrative systems.

The benchmark system will interact with a set of **Driver** systems that simulate the various sources of transactions without requiring the benchmark to implement the complex environment.

The performance metric **reported** by TPC-E is a "business throughput measuring the number of completed Trade-Result transactions processed per second (see Clause 6.5.8). Multiple transactions are used to simulate the business activity of processing a trade order transaction, and each transaction is subject to a response time constraint. The performance metric for the benchmark is expressed in transactions-per-second-E (tpsE). To be compliant with the TPC-E standard, all references to tpsE results must include the tpsE rate, the associated price-per-tpsE, and the **availability date** of the **Priced Configuration** (See Clause 6.6.2 for more detail).

Although this specification defines the 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-E uses terminology and metrics that are similar to other benchmarks, originated by the TPC and others. Such similarity in terminology does not imply that TPC-E results are comparable to other benchmarks. The only benchmark results comparable to TPC-E are other TPC-E results that conform to the same revision of the TPC-E specification.

0.1.2 Restrictions and limitations

Despite the fact that this benchmark offers a rich environment that represents 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-E 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 because of these and other factors. Therefore, TPC-E 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 this specification. A **Full Disclosure Report (FDR)** of the implementation details, as specified in Clause 9.1, 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.

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-E models and represents high-volume, complex OLTP database 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 -- BENCHMARK OVERVIEW

1.1 Definitions

NUMBERS _____

60-day Period

Storage must be priced for sufficient space to store and maintain the data and indices generated during a period of 60 **Business Days** at the **Reported Throughput Rating** called the **60-day period**.

60-Day-Space

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

60-Day-Space = Initial Database Size + (60 * Data-Growth)

A _____

Accessibility

Accessibility: the ability to retrieve the contents of database tables by the database after a failure in Clause 7.5.2.

Add

The word “**Add**” indicates that a number of rows are added to the table specified by the **Database-Footprint**. Table row(s) can only be added in a **Frame** where the word “**Add**” is specified.

Arbitrary Transaction

An **Arbitrary Transaction** is a **transaction** that executes arbitrary operations against the database at a minimum isolation level of L0 (see Clause 7.4.1.3).

Attestation Letter

Attestation Letter: The **Auditor**’s opinion regarding the compliance of a **result** must be consigned in an **attestation letter** delivered directly to the **sponsor**.

Auditor

See TPC-Certified Auditor.

Availability Date

The date when all products necessary to achieve the stated performance will be available (stated as a single date on the **Executive Summary Statement**). This is known as the **Availability Date**.

B _____

BALANCE_T

BALANCE_T is defined as **SNUM(12,2)** and is used for holding aggregate account and transaction related values such as account balances, total commissions, etc.

BOOLEAN

BOOLEAN is a data type capable of holding the value zero that reflects FALSE or the value one that denotes TRUE.

Brokerage initiated

Brokerage initiated: These **transactions** simulate broker interactions with the system and are initiated by the **Customer-Emulator** component of the benchmark **Driver**.

Broker tables

Broker tables: This set includes 9 tables that contain information about the brokerage firm and the broker related data.

Business Day

Business Day: a period of eight hours of transaction processing activity.

Business Recovery

Business Recovery is the process of recovering from a **catastrophic** system failure and reaching a point where the business meets certain operational criteria.

Business Recovery Time

Business Recovery Time is the elapsed period of time between start of **Business Recovery** and end of **Business Recovery**.

C _____

Catastrophic

Catastrophic: a type of single failure where processing is interrupted without any foreknowledge given to the **SUT**. Subsequent to this interruption, all contexts for all active applications are lost and all main memory in the **SUT** is cleared.

CE

See **Customer-Emulator**.

CHAR(n)

CHAR(n) means a character string that can hold up to n single-byte characters. When storing the string, the implementer can choose whether they pad strings with spaces to always have the maximum length, or whether they don't pad strings. **CHAR** must be implemented using a **Native Data Type**.

Commit

The word “**Commit**” indicates that the specified **Frame** contains a control operation that commits the **Database-Transaction**. Committing a **Database-Transaction** can only occur in a Frame where the word “**Commit**” is specified.

Committed

Committed: 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.

Configured Customers

Configured Customers means the number of customers (with corresponding rows in the associated tables) configured at database generation.

Customer-Emulator

One key piece of a compliant TPC-E **Driver** is the **Customer-Emulator (CE)**. The **CE** is responsible for emulating customers, requesting a service of the brokerage house, providing the necessary input for the requested service, etc. Therefore, the **CE** is responsible for the following.

- Deciding which **customer initiated** or **brokerage initiated transaction** to perform next (Broker-Volume, Customer-Position, Market-Watch, Security-Detail, Trade-Lookup, Trade-Order, Trade-Update and Trade-Status).
- Generating compliant data to be used as inputs for the selected **transaction**.
- Sending the **transaction** request and associated input data to the **SUT**.
- Receiving the **transaction** response and associated output data from the **SUT**.
- Measuring the **transaction's Response Time**.

Comment: The **CE** may optionally perform additional operations as well, such as statistical accounting, data logging, etc.

Customer initiated

Customer initiated: These **transactions** simulate customer interactions with the system and are initiated by the **Customer-Emulator** component of the benchmark **Driver**.

Customer tables

Customer tables: This set includes 9 tables that contain information about the customers of the brokerage firm.

D _____

Data-Maintenance Generator

Another key piece of a compliant TPC-E **Driver** is the single instance of the **Data-Maintenance Generator (DM)**. The **DM** is responsible for:

- Generating compliant data to be used as inputs for the Data-Maintenance **transaction**
- Sending the **transaction** request and associated input data to the **SUT**
- Receiving the **transaction's** response and associated output data from the **SUT** and measuring the **transaction's Response Time**.

Comment: The **DM** may optionally perform additional operations as well, such as statistical accounting, data logging, etc. The **DM** may optionally be used to initiate a single Trade-Cleanup transaction before the start of a **Test Run**.

Database-Footprint

The **Frame Implementation** must perform each database interaction specified in the **transaction's Database-Footprint**, using the specified access method. The **Database-Footprint** of a **transaction** is the set of required database interactions to be executed by that **transaction**.

Database Interface

Database Interface – Commercially available product used by the **Frame Implementation** to communicate with the **Database Server**. It is possible that the **Database Interface** may communicate with the **Database Server** over a **Network**, but this is not a requirement.

Data-Growth

Data-Growth: the space needed in the **DBMS** data files to accommodate the increase in the **Growing Tables** resulting from executing the **transaction mix** at the **Reported Throughput** during the period of required **sustainable** performance.

Data-Growth = *Data-Space-per-Trade-Result* * **tpsE** * **Business Day** duration in seconds

Database Logic

Database Logic – **Sponsor** written **Frame implementation** logic (e.g. stored SQL procedure)

Database Management System

A **Database Management System (DBMS)** is a collection of programs that enable you to store, modify, and extract information from a database. There are many different types of **DBMSs**, ranging from small systems that run on personal computers to huge systems that run on mainframes. From a technical standpoint, **DBMSs** can differ widely. The terms relational, network, flat, and hierarchical all refer to the way a **DBMS** organizes information internally. The internal organization can affect how quickly and flexibly you can extract information. Requests for information from a database are made in the form of a query, which is a stylized question. The set of rules for constructing queries is known as a query language. The information from a database can be presented in a variety of formats. Most **DBMSs** include a report writer program that enables you to output data in the form of a report.

Database Server

- **Database Server** – Commercially available product(s). **Sponsor** provided logic may run in the context of the **Database Server** (e.g. a stored SQL procedure). An example of a **Database Server** is:
 - commercially available **DBMS** running on a
 - commercially available **Operating System** running on a
 - commercially available hardware system utilizing
 - commercially available storage

Database session

To work with a database instance, to make queries or to manage the database instance, you have to open a **database session**. This can happen as follows: The user logs on to the database with a user name and password, thus opening a **database session**. Later, the **database session** is terminated explicitly by the user or closed implicitly when the timeout value is exceeded. A database tool implicitly opens a **database session** and then closes it again.

Database-Transaction

A **Database-Transaction** is composed of one or more database interactions enclosed between a “**start**” and a “**commit**” or “**rollback**”.

DATE

DATE represents the data type of date with a granularity of a day; a time component is not required but may be implemented. The date type used must be able to support the range of January 1, 1800 to December 31, 2199, inclusive. **DATE** must be implemented using a **Native Data Type**.

DATETIME

DATETIME represents the data type for a date value that includes a time component. The date component meets all requirements of the **DATE** data type. The time component must be capable of representing the range of time values from 00:00:00 to 23:59:59. Fractional seconds may be implemented, but are not required. **DATETIME** must be implemented using a **Native Data Type**.

DBMS

DBMS – See **Database Management System**

Dimension tables

Dimension tables: This set includes 4 dimension tables that contain common information such as addresses and zip codes.

DM

See **Data-Maintenance Generator**.

Driver

To measure the performance of the OLTP system, a simple **Driver** generates transactions and their inputs, submits them to the **System Under Test**, and measures the rate of completed transactions being returned. To simplify the benchmark and focus on the core transactional performance, all application functions related to User-Interface and Display-Functions have been excluded from the benchmark. The **System Under Test** is focused on portraying the components found on the server side of a transaction monitor or application server.

Durability

Durability: the ability to preserve the effects of committed transactions and ensure database consistency after recovery from any one of the failures listed in Clauses 7.5.2 and 7.5.3.

Durable Medium

- A **durable medium** is a data storage medium that is either:
 - An inherently non-volatile medium (e.g., magnetic disk, magnetic tape, optical disk, etc.) or
 - 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.

E _____

EGen

EGen is a TPC provided software environment that must be used in a **Test Sponsor's** implementation of the TPC-E benchmark. The software environment is logically divided into three packages: **EGenProjectFiles**, **EGenInputFiles**, and **EGenSourceFiles**. The software packages provide functionality: to load the database using **EGenLoader**, and generate transactional data using **EGenDriver**.

EGenDriver

EGenDriver comprises the following parts:

- **EGenDriverCE** provides the core functionality necessary to implement a **Customer-Emulator**.
- **EGenDriverMEE** provides the core functionality necessary to implement a **Market-Exchange-Emulator**.
- **EGenDriverDM** provides the core functionality necessary to implement the **Data-Maintenance Generator**.

EGenDriver provides core transactional functionality (e.g. **transaction mix** and input generation) necessary to implement a **Driver**.

EGenDriverCE

EGenDriverCE – any and/or all instantiations of the CCE class (see **EGenSourceFiles** CE.h and CE.cpp).

EGenDriverDM

EGenDriverDM – the single instantiation of the CDM class (see **EGenSourceFiles** DM.h and DM.cpp).

EGenDriverMEE

EGenDriverMEE – any and/or all instantiations of the CMEE class (see **EGenSourceFiles** MEE.h and MEE.cpp).

EGenInputFiles

EGenInputFiles is a set of TPC provided text files containing rows of tab-separated data, which are used by various **EGen** packages as “raw” material for data generation.

EGenLoader

EGenLoader is a binary executable, generated by using the methods described in **EGenProjectFiles** with source code from **EGenSourceFiles**, including any extensions by a **Test Sponsor** (see Clause 5.7.3). When executed, **EGenLoader** uses **EGenInputFiles** to produce a set of data that represents the initial state of the TPC-E database.

EGenLogger

EGenLogger logs the initial configuration and any re-configuration of **EGenDriver** and **EGenLoader**, and compares current configuration with the TPC-E prescribed defaults.

EGenProjectFiles

EGenProjectFiles is a set of TPC provided files used to facilitate building the **EGen** packages in a **Test Sponsor's** environments.

EGenSourceFiles

EGenSourceFiles is the collection of TPC provided C++ source and header files.

EGenTables

EGenSourceFiles contain class definitions that provide abstractions of the TPC-E tables. These table classes are known collectively as **EGenTables** and they encapsulate the functionality needed to generate the data for each of the TPC-E tables.

EGenTester

EGenTester is a binary executable, generated by using methods described in **EGenProjectFiles** with source code from **EGenSourceFiles**. When executed, **EGenTester** uses **Sponsor** provided input to validate that the **Sponsor's Measurement Interval** had compliant Trade-Results per **Load Unit**.

EGenTxnHarness

EGenTxnHarness defines a set of interfaces that are used to control the execution of, and communication of inputs and outputs, of **transactions** and **frames**.

Executive Summary Statement

The term **Executive Summary Statement** refers to the Adobe Acrobat PDF file in the **ExecutiveSummaryStatement** folder in the **FDR**. The contents of the **Executive Summary Statement** are defined in Clause 9.

F _____

FDR

The **FDR** is a zip file of a directory structure containing the following:

- A **Report** in Adobe Acrobat PDF format,
- An **Executive Summary Statement** in Adobe Acrobat PDF format, and
- The **Supporting Files** consisting of various source files, scripts, and listing files. Requirements for the **FDR** file directory structure are described below.

Comment: The purpose of the **FDR** is to document how a benchmark result was implemented and executed in sufficient detail so that the result can be reproduced given the appropriate hardware and software products.

FIN_AGG_T

FIN_AGG_T is defined as **SNUM(15,2)** and is used for holding aggregated financial data such as revenue figures, valuations, and asset values.

Fixed-Space

Fixed-Space: any other space used to store static information and indices. It includes all database storage space allocated to the test database which does not qualify as either **Free-Space** or **Growing-Space**.

Fixed Tables

Fixed Tables: These tables always have the same number of rows regardless of the database size and transaction throughput. For example, **TRADE_TYPE** has five rows.

Foreign Key

A **Foreign Key** (FK) is a column or combination of columns used to establish and enforce a link between the data in two tables. A link is created between two tables by adding the column or columns that hold one table's **Primary Key** values to the other table. This column becomes a **Foreign Key** in the second table.

Frame

A **Frame** is the **sponsor** implemented transaction logic, which is invoked as a unit of execution by the **EGenTxnHarness**. The database interactions of a **transaction** are all initiated from within its **Frames**.

Frame Implementation

Frame Implementation – **Sponsor** provided functionality that accepts inputs from, and provides outputs to, **EGenTxnHarness** through a TPC defined interface. The **Frame Implementation** and all down-stream functional components are responsible for providing the appropriate functionality outlined in the **transaction** profiles (Clause 3.3).

Free-Space

Free-Space: any space allocated to the test database and available for future use. It includes all database storage space not already used to store a database entity (e.g., a row, an index, metadata) or not already used as formatting overhead by the **DBMS**.

Full Disclosure Report (FDR)

See FDR.

G _____

Growing-Space

Growing-Space: any space used to store existing rows from the **Growing Tables** (i.e., the CASH_TRANSACTION, HOLDING, HOLDING_HISTORY, SETTLEMENT, TRADE, and TRADE_HISTORY tables) and their associated indices and storage overhead. It includes all database storage space that is added to the test database as a result of inserting a new row in the **Growing Tables**, such as row data, index data and other overheads such as index overhead, page overhead, block overhead, and table overhead.

Growing Tables

Growing Tables: These tables each have an initial cardinality that has a constant relationship to the cardinality of the CUSTOMER table. However, the cardinality increases with new growth during the benchmark run at a rate that is proportional to transaction throughput rates. The HOLDING table has been included in this category. Rows are added and deleted from the HOLDING table during the benchmark run, but the average size of the table continues to grow during the benchmark at a rate that is proportional to transaction throughput rates. The TRADE_REQUEST table is also considered a Growing table, even though its average size is a fixed relationship to the transaction throughput rates and not to the cardinality of the CUSTOMER table.

H _____

I _____

IDENT_T

IDENT_T is defined as NUM(11) and is used to hold identifier fields.

Initial Database Size

Initial Database Size is measured after the database is initially loaded with the data generated by EGenLoader. **Initial Database Size** is any space allocated to the test database which is used to store a database entity (e.g. a row, an index, metadata), or used as formatting overhead by the data manager.

Initial Trade Days

The **Initial Trade Days (ITD)** is the number of **Business Days** used to populate the database. This population is made of trade data that would be generated by the **SUT** when running at the **Nominal Throughput** for the specified number of **Business Days**. The number of **Initial Trade Days** is 300.

ITD

ITD see **Initial Trade Days**.

J _____

K _____

L _____

Load Unit

The size of the CUSTOMER table can be increased in increments of 1000 customers. A set of 1000 customers is known as a **load unit**.

LOB(n)

LOB(n) is a data type capable of holding a variable length binary object of n bytes.

Log-Growth

Log-Growth: the space needed in the DBMS log files to accommodate the **Undo/Redo Log** resulting from executing the **transaction mix** at the **Reported Throughput** during the period of required **sustainable** performance.

Log-Growth = *Log-Space-per-Trade-Result* * **tpsE** * **Business Day** duration in seconds

M _____

Market-Exchange-Emulator

Another key piece of a compliant TPC-E **Driver** is the **Market-Exchange-Emulator (MEE)**. The **MEE** is responsible for emulating the stock exchanges: providing services to the brokerage house, performing requested trades, providing market activity updates, etc. Therefore, the **MEE** is responsible for the following:

- Receiving trade requests and their associated data from the **SUT**.
- Initiating Trade-Result **transactions**, sending the associated data to the **SUT** and measuring the **transaction's Response Time**.
- Initiating Market-Feed **transactions**, sending the associated data to the **SUT** and measuring the **transaction's Response Time**.

Comment: The **MEE** may optionally perform additional operations as well; such as statistical accounting, data logging, etc.

Market tables

Market tables: This set includes 11 tables that contain information about companies, markets, exchanges, and industry sectors.

Market triggered

Market triggered: These **transactions** simulate the behavior of the market and are triggered by the **Market-Exchange-Emulator** component of the benchmark **Driver**.

May

The word “**may**” in the specification means that an item is truly optional.

Measured Throughput

The **Measured Throughput** is computed as the total number of **Valid Trade-Result transactions** within the **Measurement Interval** divided by the duration of the **Measurement Interval** in seconds.

Measured throughput period

- A **Measured throughput period** is a period of time in which the following criteria are satisfied:
 - be performed with a fully scaled database and **driver** load
 - be in **Steady State**
 - satisfy **Response Time** constraints listed in Clause 6.4.1.2
 - satisfy the **Transaction Mix** requirements listed in Clause 6.2.2
 - be at or above 95% of the **Reported Throughput** with no errors
 - match all **driver** and **SUT** configuration settings used during the **Reported Throughput Measurement Interval**.

Measurement Interval

Measurement Interval: the period of time during **Steady State** chosen by the **Test Sponsor** to compute the **Reported Throughput Rating**.

MEE

See **Market-Exchange-Emulator**

Modify

The word “**Modify**” indicates that the content of a table column is modified within the **Frame**. The content of the table column can only be changed in a **Frame** where the word “**Modify**” is specified. When the original content of the table column must also be referenced or returned before it is modified, a “**Reference**” or a “**Return**” access method is also specified.

Must

The word “**must**” or the terms “required”, “requires”, “requirement” or “shall” in the specification, means that compliance is mandatory.

Must not

The phrase “**must not**” or the term “shall not” in the specification, means that this is an absolute prohibition of the specification.

N _____

Native Data Type

A **Native Data Type** is a built-in data type of the **DBMS** whose documented purpose is to store data of the type described in the specification for that attribute. For example, **DATETIME** must be implemented with a built-in data type of the **DBMS** designed to store date-time information.

Network

Network – Sponsor provided functionality that must support communication through an industry standard communications protocol using a physical means. One outstanding feature of the Connector – Network – Connector communication is that it follows the relevant standards and must imply more than just an application package. It must be possible to have concurrent use of the means by other applications. Physical transport of the data is required and the underlying means of this transport must be capable of operating over arbitrary globally geographic distances. TPC/IP over a local area network is an example of an acceptable Network implementation.

Nominal Throughput

The **Nominal Throughput** of the TPC-E benchmark is defined to be 2.00 **Transactions-Per-Second-E** (tpsE) for every 1000 customer rows in the **Configured Customers**.

Non-catastrophic

The term **non-catastrophic** as applied to a single failure is one where processing is not interrupted, but throughput may be degraded and the **SUT** may no longer be in a durable state until the **SUT** has recovered from the failure.

NUM(m[,n])

NUM(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 **NUM(m[,n])**. Omitting n, as in **NUM(m)**, indicates the same as **NUM(m,0)**. **NUM** must be implemented using a **Native Data Type**.

O _____

Operating System/OS

The term **Operating System** refers to the program that, after being initially loaded into the computer by a boot program, manages all the other programs in a computer. The **Operating System** provides a software platform on top of which all other programs run. Without the **Operating System** and the core services that it provides no other programs can run and the computer would be non-functional. The other programs are referred to as applications or application programs. The application programs make use of the **Operating System** by making requests for services through a defined application program interface (API). All major computer platforms require an **Operating System**. The functions and services supplied by an **Operating System** include but are not limited to the following:

- Manages a dedicated set of processor and memory resources.
- Maintains and manages a file system.
- Loads applications into memory.
- Ensures that the resources allocated to one application are not used by another application in an unauthorized manner.
- Determines which applications should run in what order, and how much time should be allowed to run the application before giving another application a turn to use the systems resources.
- Manages the sharing of internal memory among multiple applications.
- Handles input and output to and from attached hardware devices such as hard disks, network interface cards etc.

Some examples of **Operating Systems** are listed below:

- Windows
- Unices (Solaris, AIX)
- Linux
- MS-DOS
- Mac OS
- VMS
- Netware

P _____

Pacing Delay

The **Pacing Delay** is defined by:

$$PD_n = sT_{n+1} - eT_n$$

where:

sT_{n+1} and eT_n are measured at the **CE Driver**;

eT_n = time measured after the last byte of output data from the current **Transaction** is received by the **CE Driver** from the **SUT**; and

sT_{n+1} = time measured before the first byte of input data of the next **transaction** is sent by the **CE Driver** to the **SUT**.

Performance Metric

The TPC-E **Throughput Rating** as expressed in tpsE. This is known as the **Performance Metric**.

Priced Configuration

Priced Configuration: The components to be priced defined in the benchmark specification, including all hardware, software and maintenance.

Price/Performance Metric

The TPC-E total 3-year pricing divided by the **Throughput Rating** is price/tpsE. This is also known as the **Price/Performance metric**.

Primary Key

A **Primary Key** is a single field or combination of fields that uniquely defines a record. None of the fields that are part of the **Primary Key** can contain a null value. A table can have only one **Primary Key**.

Q _____

R _____

Ramp-down

Ramp-down: the period of time from the end of **Steady State** to the end of the **Test Run**.

Ramp-up

Ramp-up: the period of time from the start of the **Test Run** to the start of **Steady State**.

Redundancy Level One

Redundancy Level One: Guarantees access to the data on durable media when a single durable media failure occurs.

Redundancy Level Three

Redundancy Level Three: Includes **Redundancy Level Two** and guarantees access to the data on durable media when a single failure of a storage controller/interface card in the central processing complex occurs.

Redundancy Level Two

Redundancy Level Two: Includes **Redundancy Level One** and guarantees access to the data on durable media when a single failure in the processor/cache/controller of the durable media enclosure occurs.

Reference

The word “**Reference**” indicates that the table column is identified in the database and the content is accessed within the **Frame** without passing the content of the table column to the EGenTxnHarness.

Referential Integrity

Referential Integrity preserves the relationship of data between tables, by restricting actions performed on primary and **Foreign Keys** in a table. **Referential Integrity** prevents removing rows containing **Primary Keys** that are referenced by **Foreign Keys** in other tables in the database. **Referential Integrity** also prevents adding rows containing **Foreign Keys** that refer to **Primary Keys** whose rows are not already present in the database. **Referential Integrity** does not allow modifications to **Primary Key** columns of rows that are referenced by **Foreign Keys** in other tables in the database.

Remove

The word “**Remove**” indicates that a number of rows are removed from the table specified by the **Database-Footprint**. Table row(s) can only be removed in a **Frame** where the word “**Remove**” is specified. The number of rows that are removed is specified in the second column of the **Database-Footprint** with either “# row” for a fixed number of rows or “row(s)” for an unspecified number of rows.

Report

The term **Report** refers to the Adobe Acrobat PDF file in the Report folder in the **FDR**. The contents of the **Report** are defined in Clause 9.

Reported

The term **Reported** refers to an item that is part of the **FDR**.

Reported Throughput Rating/Reported Throughput

The **Reported Throughput Rating** must be measured, rather than interpolated or extrapolated, and rounded down to the nearest two decimal places. For example, suppose 105.748 tpsE is measured during a **Measurement Interval** for which all 90% **Response Time** constraints are met and 117.572 tpsE is measured during a **Measurement Interval** for which some 90% **Response Time** constraints are exceeded. Then the **Reported Throughput** is 105.74 tpsE rather than 105.75 or some interpolated value between 105.748 and 117.572.

Response Time

The **Response Time (RT)** is defined by:

$$RT_n = eT_n - sT_n$$

where:

sT_n and eT_n are measured at the **Driver**;

sT_n = time measured before the first byte of input data of the **transaction** is sent by the **Driver** to the **SUT**; and

eT_n = time measured after the last byte of output data from the **transaction** is received by the **Driver** from the **SUT**.

Comment 1: The resolution of the time stamps used for measuring **Response Time** must be at least 0.01 seconds.

Comment 2: For the purpose of calculating the **Response Time**, only **valid transactions** may be included.

Results

TPC-E **Results** are the **Performance Metric, Price/Performance Metric**.

Return

The word “**Return**” indicates that the table column is referenced and that its content is retrieved from the database and passed to the **EGenTxnHarness**. The table column must be referenced in the same **Frame** where the word “**Return**” is specified. The content of the table column can only be passed to subsequent **Frames** via the input and output parameters specified in the **Frame** parameters.

Rollback

The word “**Rollback**” indicates that the specified **Frame** contains a control operation that rolls back the **Database-Transaction**. The explicit rolling back of a **Database-Transaction** can only occur in a **Frame** where the word “**Rollback**” is specified.

RT

RT see **Response Time**.

S _____

S_COUNT_T

S_COUNT_T is defined as **NUM(12)** and is used for holding the aggregate count of shares used in many tables.

S_PRICE_T

S_PRICE_T is defined as **NUM(8,2)** and is used for holding the value of a share price.

S_QTY_T

S_QTY_T is defined as **SNUM(6)** and is used for holding the quantity of shares per individual trade.

Scale Factor

The **Scale Factor (SF)** is the number of customer rows (500) per single **Transaction-Per-Second-E** (tpsE).

Scaling Tables

Scaling Tables: These tables each have a cardinality that has a constant relationship to the cardinality of the **CUSTOMER** table. **Transactions** may update rows from these tables, but the table sizes remain constant.

Session

See **Database session**.

SF

SF see **Scale Factor**.

Should

The word “**should**” or the adjective “recommended”, mean that there might exist valid reasons in particular circumstances to ignore a particular item, but the full implication must be understood and weighed before choosing a different course.

Should not

The phrase “should not”, or the phrase “not recommended”, means that there might exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

SNUM

SNUM(m[,n]) is identical to **NUM(m[,n])** except that it can represent both positive and negative values. **SNUM** must be implemented using a **Native Data Type**.

Note: A **SNUM** data type may be used (at the **sponsor’s** discretion) anywhere a **NUM** data type is specified.

Sponsor

See **Test Sponsor**.

Start

The word “**Start**” indicates that the specified **Frame** contains a control operation that starts a **Database-Transaction**. The start of a **Database-Transaction** can only occur in a **Frame** where the word “**Start**” is specified.

Steady State

Steady State: the period of time from the end of the **Ramp-up** to the start of the **Ramp-down**.

Supporting Files

Supporting Files refers to the contents of the SupportingFiles folder in the **FDR**. The contents of this folder, consisting of various source files, scripts, and listing files, are defined in Clause 9.

Sustainable

Sustainable: the performance over a given period of time (computed as the average throughput over that time) shows no significant variations.

SUT

SUT see **System Under Test**.

System Under Test

System Under Test (SUT) – is defined to be the sum of **Tier A** and **Tier B**.

T _____

Test Sponsor

The **Test Sponsor** is the company officially submitting the **Result** with the **FDR** and will be charged the filing fee. Although multiple companies may sponsor a **Result** together, for the purposes of the TPC’s processes the **Test Sponsor** must be a single company. A **Test Sponsor** need not be a TPC member. The **Test Sponsor** is responsible for maintaining the **FDR** with any necessary updates or corrections. The **Test Sponsor** is also the name used to identify the **Result**.

Test Run

Test Run: the entire period of time during which **Drivers** submit and the **SUT** completes **transactions** other than Trade-Cleanup.

Test Run Graph

A graph of the **Measured Throughput** versus elapsed wall clock time measured in minutes must be **reported** for the entire **Test Run**. The x-axis represents the elapsed time from the **Test Run** start. The y-axis represents the throughput in tpsE. A plot interval size of 1 minute must be used. The **Ramp-up**, the **Measurement Interval** and **Steady State** must be identified on the graph.

Throughput Rating

The performance metric **reported** by TPC-E is the **Throughput Rating**. The name of the metric used for the **Throughput Rating** of the **SUT** is **tpsE**. The value of this metric is based on the **Measured Throughput** and is bound by the **Nominal Throughput** limits of the **SUT** as described in Clause 6.5.7.2.

Tier A

Tier A – is defined to be all hardware and software needed to implement the down-stream Connector, **EGenTxnHarness**, **Frame Implementation** and **Database Interface** functional components.

Tier B

Tier B – is defined to be all hardware and software needed to implement the **Database Server** functional component. This includes data storage media sufficient to satisfy the initial database population requirements of clause 2.6.1 and the **Business Day** growth requirements of clause 6.5.6.3 and clause 6.5.6.4.

TPC-Certified Auditor

The term **TPC-Certified Auditor** is used to indicate that the TPC has reviewed the qualification of the **auditor** and has certified his/her ability to verify that benchmark results are in compliance with this specification. (Additional details regarding the **auditor** certification process and the audit process can be found in Section 9 of the TPC Policy document.)

TRADE_T

TRADE_T is defined as **NUM(15)** and is used to hold trade identifier fields.

Trade identifiers have the following characteristics:

- They must be unique.
- They may be sparse.
- At load time they are generated by **EGenLoader**.
- At run time they are generated by **sponsor** provided code.
- The **EGenLoader** code will not associate trade identifiers with Date/time or customer identifier or account identifiers. No assumptions may be made about trade identifier sequencing.

Transaction(s)

The TPC-E **transactions** are at the heart of the workload. The core of each **transaction** runs on the **Database Server**, but the logic of the **transaction** interacts with several components of the benchmark environment.

A **transaction** is composed of Harness-code and of the invocation of one or more **Frames**. The Trade-Cleanup **transaction** is an exception. **Sponsors** may but do not have to run the Trade-Cleanup **transaction** from EGenTxnHarness.

Transaction mix

The TPC-E workload is made up of a set of **transactions** acting against a database following a required **transaction mix**.

Over the **Reported Measurement Interval**, the **Driver** must maintain the mix of **transactions** specified in Clause 6.2.2.1. As a natural result of using the random number selection process, the **Driver** is likely to deviate from the specified **transaction mix** percentages. The maximum deviation allowed by the benchmark is defined in Clause 6.2.2.3.

For the purpose of computing the mix, only **valid transactions** may be counted. **Transaction** errors are not allowed during the **Measurement Interval**.

Tunable Parameters

Tunable Parameters are parameters, switches or flags that can be changed to modify the behavior of the product. **Tunable Parameters** apply to both hardware and software and are not limited to those parameters intended for use by customers.

U _____

U*x

U*x is used in this specification to refer to various UNIX and Linux flavors (e.g. UNIX, Linux, AIX, Solaris).

Undo/Redo Log

Undo/Redo Log: records all changes made in data files. The **Undo/Redo Log** makes it possible to replay all the actions executed by the **Database Management System**. If something happens to one of the data files, a backed up data file can be restored and the **Undo/Redo Log** that was written since the backup can be played and applied which brings the data file to the state it had before it became unavailable.

V _____

Valid Transaction

The term **Valid Transaction** refers to any **Transaction** for which input data has been sent in full by the **Driver**, whose processing has been successfully completed on the **SUT** and whose correct output data has been received in full by the **Driver**.

Comment: Any **Transaction** that requires a rollback that runs successfully and produces the correct output is considered a **valid Transaction**.

VALUE_T

VALUE_T is defined as **SNUM(10,2)** and is used for holding non-aggregated transaction and security related values such as cost, dividend, etc.

W _____

X _____

Y _____

Z _____

1.2 Business and Application Environment

TPC Benchmark™ E is composed of a set of transactional operations designed to exercise system functionalities in a manner representative of complex OLTP application environments. These transactional operations have been given a life-like context, portraying the activity of a brokerage firm, to help users relate intuitively to the components of the benchmark. The workload is centered on the activity of processing brokerage trades and uses a schema, which is logically divided in four sets of tables.

TPC-E models the activity of brokerage firm that must manage customer accounts, execute customer trade orders, and be responsible for the interactions of customers with financial markets. TPC-E does not attempt to be a model of how to build an actual application. The following diagram illustrates the transaction flow of the business model portrayed in the benchmark:

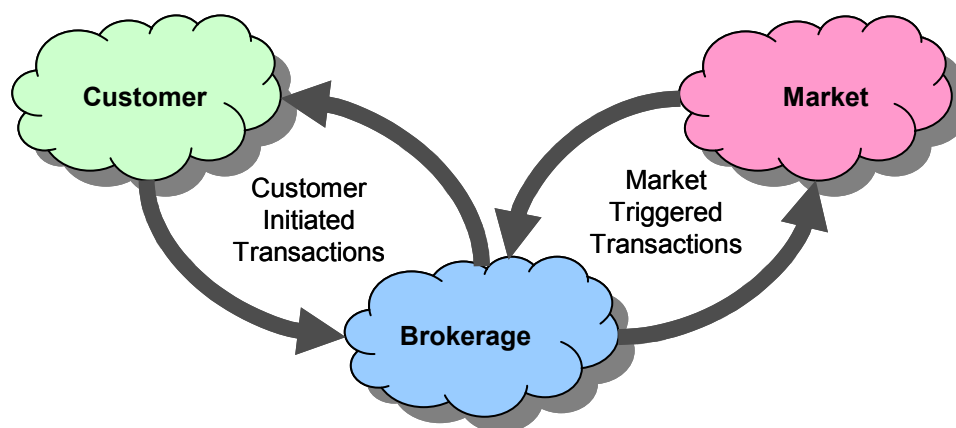


Figure 1.a - Business Model Transaction Flow

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 so that the workload can be representative of a production system. A large number of functions have to be performed to manage a production brokerage 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-E.

The Company portrayed by the benchmark is a brokerage firm with customers who generate transactions related to trades, account inquiries, and market research. The brokerage firm in turns interacts with financial markets to execute orders on behalf of the customers and updates relevant account information.

The number of customers defined for the brokerage firm can be varied to represent the workloads of different size businesses.

This benchmark is composed of a set of transactions that are executed against three sets of database tables that represent market data, customer data, and broker data. A fourth set of tables contains generic dimension data such as zip codes. The following diagram illustrates the key components of the environment:

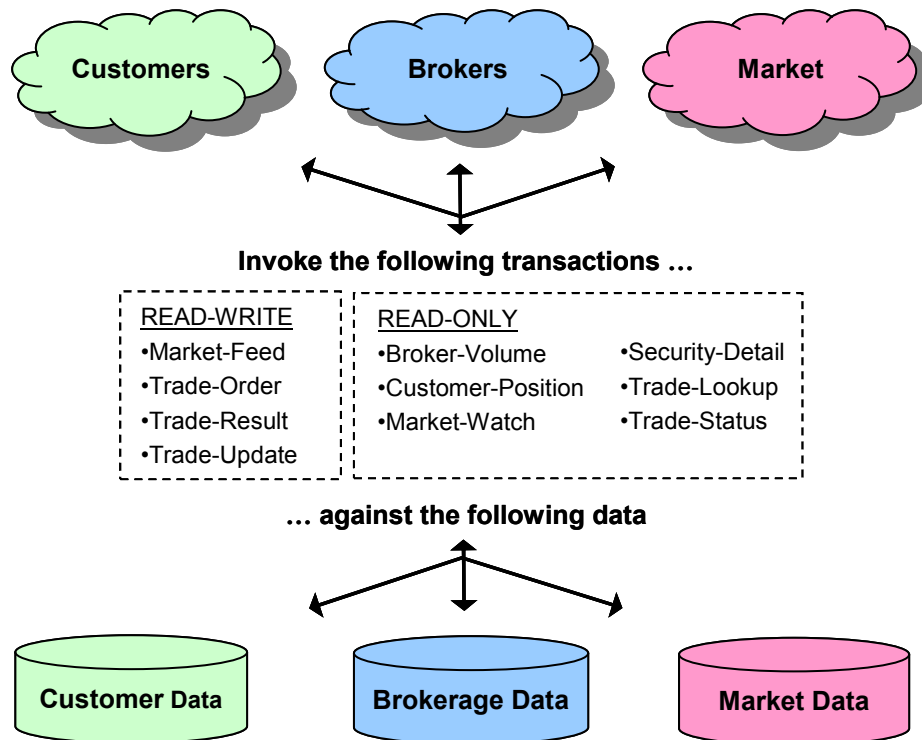


Figure 1.b - Application Components

The benchmark has been reduced to simplified form of the application environment. To measure the performance of the OLTP system, a simple **Driver** generates transactions and their inputs, submits them to the **System Under Test**, and measures the rate of completed transactions being returned. To simplify the benchmark and focus on the core transactional performance, all application functions related to User-Interface and Display-Functions have been excluded from the benchmark. The **System Under Test** is focused on portraying the components found on the sever side of a transaction monitor or application server.

1.3 Model Description

1.3.1 Entity Relationships

- 1.3.1.1 Trading in TPC-E is done by Accounts. Accounts belong to Customers. Customers are serviced by Brokers. Accounts trade Securities that are issued by Companies.
- 1.3.1.2 The total set of Securities that can be traded and the total set of Companies that issue Securities scales along with the number of Customers. For each unit of 1,000 Customers, there are 685 Securities and 500 Companies (with Companies issuing 1 to 5 Securities, mostly common shares, but some preferred as well).
- 1.3.1.3 All Companies belong to one of the 102 Industries. Each Industry belongs to one of the 12 market Sectors.
- 1.3.1.4 Each Account picks its average ten Securities to trade from across the entire range of Securities.
- 1.3.1.5 Securities to be traded can be identified by the security symbol or by the company name and security issue.

1.3.2 Differences between customer tiers

- 1.3.2.1 The basic scaling unit of a TPC-E database is a set of 1,000 Customers. 20% of each 1,000 Customers belong to Tier 1, 60% to Tier 2, and 20% to Tier 3. Tier 2 Customers trade twice as often as Tier 1 Customers. Tier 3 Customers trade three times as often as Tier 1 Customers. In general, customer trading is non-uniform by tier within each set of 1,000 Customers.
- 1.3.2.2 Tier 1 Customers have 1 to 4 Accounts (average 2.5). Tier 2 Customers have 2 to 8 Accounts (average 5.0). Tier 3 Customers have 5 to 10 Accounts (average 7.5). Overall, there is an average of five Accounts per Customer.
- 1.3.2.3 The minimum and maximum number of Securities that are traded by each Account varies by Customer Tier and by the number of Accounts for each Customer. The average number of Securities traded per Account is ten (so the average number of Securities traded per Customer is fifty). For each Account, the same set of Securities is traded for both the initial database population and for runtime.

1.3.3 Trade types

- 1.3.3.1 Trade requests come in two basic flavors: Buy (50%) and Sell (50%). Those are further broken down into Trade Types, depending on whether the request was a Market Order (60%) or a Limit Order (40%).
- 1.3.3.2 For Market Orders, the two trade types are Market-Buy (30%) and Market-Sell (30%). For Limit Orders, the three trade types are Limit-Buy (20%), Limit-Sell (10%) and Stop-Loss (10%).
- 1.3.3.3 Market-Buy and Market-Sell are trade requests to buy and sell immediately at the current market price, whatever price that may be. Limit-Buy is a request to buy only when the market price is at or below the specified limit price. Limit-Sell is a request to sell only when the market price is at or above the specified limit price. Stop-Loss is a request to sell only when (or if) the market price drops to or below the specified limit price.

1.3.3.4 If the specified limit price has not been reached when the Limit Order is requested, it is considered an Out-of-the-Money request and remains “Pending” until the specified limit price is reached. Reaching the limit price is guaranteed to occur within 15 minutes based on **EGenDriverMEE** implementation details. The act of noticing that a “Pending” limit request has reached or exceeded its specified limit price and submitting it to the market exchange to be traded is known as triggering of the pending limit order.

1.3.4 Effects of trading on holdings

1.3.4.1 Before **EGenLoader** runs, there are no trades and no positions in any security for any account. **EGenLoader** simulates running the benchmark for three hundred **Business Days** of initial trading, so that the initial database will be ready for benchmark execution.

1.3.4.2 If the first trade for a security in an account is a buy, a long position will be established (positive quantity in HOLDING row). Subsequent buys in the same account for the same security will add holding rows with positive quantities. Subsequent sells will reduce holding quantities or delete holding rows to satisfy the sell trade. All holdings may be eliminated, in which case the position becomes empty. If the sell quantity still is not satisfied, the position changes from long to short (see below).

1.3.4.3 If the first trade for a security in an account is a sell, a short position will be established (negative quantity in HOLDING row). Subsequent sells in the same account for the same security will add holding rows with negative quantities. Subsequent buys will reduce holding quantities (toward zero) or delete holding rows to satisfy the buy trade. All holdings may be eliminated, in which case the position becomes empty. If the buy quantity still is not satisfied, the position changes from short to long.

1.3.4.4 For a given account and security, holdings will be either all long (positive quantities) or all short (negative quantities).

1.3.4.5 Long positions represent shares of the security that were bought (purchased and paid for) by the customer for the account. The customer owns the shares of the security and may sell them at a later time (hopefully, for a higher price).

1.3.4.6 Short positions represent shares of the security that were borrowed from the broker (or Brokerage) and were sold by the customer for the account. In the short sale case, the customer has received the funds from that sell, but still has to cover the sell by later purchasing an equal number of shares (hopefully at a lower price) from the market and returning those shares to the broker.

CLAUSE 2 -- DATABASE DESIGN, SCALING & POPULATION

2.1 Database Entities, Relationships, and Characteristics

2.1.1 The components of the TPC-E database are defined to consist of 33 separate and individual tables. The relationships among these tables are defined in this clause and are subject to the rules specified in Clause 2.

2.1.2 The overall TPC-E schema can be organized into four sets of tables:

- **Market** tables: This set includes 11 tables that contain information about companies, markets, exchanges, and industry sectors.
- **Customer** tables: This set includes 9 tables that contain information about the customers of the brokerage firm.
- **Broker** tables: This set includes 9 tables that contain information about the brokerage firm and the broker related data.
- **Dimension** tables: This set includes 4 dimension tables that contain common information such as addresses and zip codes.

2.2 Database Schema and Table Definitions

The following sections define the name, the required structure (list of columns) of each table, the data types used and the various restrictions related to the schema.

2.2.1 Data Type Definitions

2.2.1.1 **CHAR(n)** means a character string that can hold up to n single-byte characters. When storing the string, the implementer can choose whether they pad strings with spaces to always have the maximum length, or whether they don't pad strings. **CHAR** must be implemented using a **Native Data Type**.

2.2.1.2 **NUM(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 **NUM(m[,n])**. Omitting n, as in **NUM(m)**, indicates the same as **NUM(m,0)**. **NUM** must be implemented using a **Native Data Type**.

2.2.1.3 **SNUM(m[,n])** is identical to **NUM(m[,n])** except that it can represent both positive and negative values. **SNUM** must be implemented using a **Native Data Type**.

Note: A **SNUM** data type may be used (at the **sponsor's** discretion) anywhere a **NUM** data type is specified.

2.2.1.4 **BOOLEAN** is a data type capable of holding the value zero that reflects FALSE or the value one that denotes TRUE.

2.2.1.5 **DATE** represents the data type of date with a granularity of a day; a time component is not required but may be implemented. The date type used must be able to support the range of January 1, 1800 to December 31, 2199, inclusive. **DATE** must be implemented using a **Native Data Type**.

2.2.1.6 **DATETIME** represents the data type for a date value that includes a time component. The date component meets all requirements of the **DATE** data type. The time component must be capable of representing the range of time values from 00:00:00 to 23:59:59. Fractional seconds may be implemented, but are not required. **DATETIME** must be implemented using a **Native Data Type**.

2.2.1.7 **LOB(n)** is a data type capable of holding a variable length binary object of n bytes.

2.2.2 Meta-type Definitions

The following meta-types are defined for ease of notation. These meta-types may be implemented using the underlying data type on which each is defined. There is no requirement to implement the meta-types as user-defined types in the **DBMS**. A meta-type may be implemented using a user-defined type in the **DBMS** as long as the user-defined type incorporates a **Native Data Type** where required and inherits the properties of that **Native Data Type**.

2.2.2.1 **IDENT_T** is defined as **NUM(11)** and is used to hold identifier fields.

2.2.2.2 **TRADE_T** is defined as **NUM(15)** and is used to hold trade identifier fields.

Trade identifiers have the following characteristics:

- They must be unique.
- They may be sparse.
- At load time they are generated by **EGenLoader**.
- At run time they are generated by **sponsor** provided code.
- The **EGenLoader** code will not associate trade identifiers with Date/time or customer identifier or account identifiers. No assumptions may be made about trade identifier sequencing.

2.2.2.3 **FIN_AGG_T** is defined as **SNUM(15,2)** and is used for holding aggregated financial data such as revenue figures, valuations, and asset values.

2.2.2.4 **S_PRICE_T** is defined as **NUM(8,2)** and is used for holding the value of a share price.

2.2.2.5 **S_COUNT_T** is defined as **NUM(12)** and is used for holding the aggregate count of shares used in many tables.

2.2.2.6 **S_QTY_T** is defined as **SNUM(6)** and is used for holding the quantity of shares per individual trade.

2.2.2.7 **BALANCE_T** is defined as **SNUM(12,2)** and is used for holding aggregate account and transaction related values such as account balances, total commissions, etc.

2.2.2.8 **VALUE_T** is defined as **SNUM(10,2)** and is used for holding non-aggregated transaction and security related values such as cost, dividend, etc.

2.2.3 General Schema Items

The following table lists the category, prefix and the name for all required tables in the benchmark.

Category	Table Name	Table Prefix	Definition
CUSTOMER	ACCOUNT_PERMISSION	AP_	Clause 2.2.4.1
	CUSTOMER	C_	Clause 2.2.4.2
	CUSTOMER_ACCOUNT	CA_	Clause 2.2.4.3
	CUSTOMER_TAXRATE	CX_	Clause 2.2.4.4
	HOLDING	H_	Clause 2.2.4.5
	HOLDING_HISTORY	HH_	Clause 2.2.4.6
	HOLDING_SUMMARY	HS_	Clause 2.2.4.7
	WATCH_ITEM	WI_	Clause 2.2.4.8
	WATCH_LIST	WL_	Clause 2.2.4.9
BROKER	BROKER	B_	Clause 2.2.5.1
	CASH_TRANSACTION	CT_	Clause 2.2.5.2
	CHARGE	CH_	Clause 2.2.5.3
	COMMISSION_RATE	CR_	Clause 2.2.5.4
	SETTLEMENT	SE_	Clause 2.2.5.5
	TRADE	T_	Clause 2.2.5.6
	TRADE_HISTORY	TH_	Clause 2.2.5.7
	TRADE_REQUEST	TR_	Clause 2.2.5.8
	TRADE_TYPE	TT_	Clause 2.2.5.9
MARKET	COMPANY	CO_	Clause 2.2.6.1
	COMPANY_COMPETITOR	CP_	Clause 2.2.6.2
	DAILY_MARKET	DM_	Clause 2.2.6.3
	EXCHANGE	EX_	Clause 2.2.6.4
	FINANCIAL	FI_	Clause 2.2.6.5
	INDUSTRY	IN_	Clause 2.2.6.6
	LAST_TRADE	LT_	Clause 2.2.6.7
	NEWS_ITEM	NI_	Clause 2.2.6.8
	NEWS_XREF	NX_	Clause 2.2.6.9
	SECTOR	SC_	Clause 2.2.6.10
	SECURITY	S_	Clause 2.2.6.11
DIMENSIONS	ADDRESS	AD_	Clause 2.2.7.1
	STATUS_TYPE	ST_	Clause 2.2.7.2
	TAXRATE	TX_	Clause 2.2.7.3
	ZIP_CODE	ZC_	Clause 2.2.7.4

- 2.2.3.1 The **Primary Key** references defined in this section must be maintained by the database. The **Primary Keys** are marked with PK and PK+ in the Relations filed for each table definition. PK indicates that the column is the table's **Primary Key** while PK+ indicates that the column is part of a composite (multi-column) **Primary Key**.
- 2.2.3.2 The **Foreign Key** references defined in this section must be maintained by the database. The **Foreign Keys** are marked with FK() and FK+() in the Relations field for each table definition. FK() indicates a single-column **Foreign Key** while FK+() indicates that the column is part of a composite (multi-column) **Foreign Key**. The table prefix enclosed in the parenthesis indicates the target table for the **Foreign Key** reference.
- 2.2.3.3 The check constraints defined in this section must be enforced by the database.
- 2.2.3.4 In general, for the Database Schema definition, if the Constraint column does not specify *not null*, the column may contain nulls.
- 2.2.3.5 For each table the fields can be implemented in any order, using any physical representation available from the tested system that satisfies the schema data type requirements.

2.2.4 Customer Tables

These groups of tables contain information about customer related data.

2.2.4.1 ACCOUNT_PERMISSION

This table contains information about the access the customer or an individual other than the customer has to a given customer account. Customer accounts may have trades executed on them by more than one person.

Table Prefix: AP_

Field Name	Field Type	Constraint	Relations	Comment
AP_CA_ID	IDENT_T	Not Null	PK+ FK (CA_)	Customer account identifier.
AP_ACL	CHAR(4)	Not Null		Access Control List defining the permissions the person has on the customer account.
AP_TAX_ID	CHAR(20)	Not Null	PK+	Tax identifier of the person with access to the customer account.
AP_L_NAME	CHAR(30)	Not Null		Last name of the person with access to the customer account.
AP_F_NAME	CHAR(30)	Not Null		First name of the person with access to the customer account.

2.2.4.2 CUSTOMER

This table contains information about the customers of the brokerage firm.

Table Prefix: C_

Field Name	Field Type	Constraint	Relations	Comment
C_ID	IDENT_T	Not Null	PK	Customer identifier, used internally to link customer information

C_TAX_ID	CHAR(20)	Not Null		Customer's tax identifier, used externally on communication to the customer. Is alphanumeric.
C_ST_ID	CHAR(4)	Not Null	FK (ST_)	Customer status type identifier. Identifies if this customer is active or not.
C_L_NAME	CHAR(30)	Not Null		Primary Customer's last name.
C_F_NAME	CHAR(30)	Not Null		Primary Customer's first name.
C_M_NAME	CHAR(1)			Primary Customer's middle name initial
C_GNDR	CHAR(1)			Gender of the primary customer. Valid values 'M' for male or 'F' for Female.
C_TIER	NUM(1)	Not Null in 1,2,3		Customer tier: tier 1 accounts are charged highest fees, tier 2 accounts are charged medium fees, and tier 3 accounts have the lowest fees.
C_DOB	DATE	Not Null		Customer's date of birth.
C_AD_ID	IDENT_T	Not Null	FK (AD_)	Address identifier of the customer's address.
C_CTRY_1	CHAR(3)			Country code for Customer's phone 1.
C_AREA_1	CHAR(3)			Area code for customer's phone 1.
C_LOCAL_1	CHAR(10)			Local number for customer's phone 1.
C_EXT_1	CHAR(5)			Extension number for Customer's phone 1.
C_CTRY_2	CHAR(3)			Country code for Customer's phone 2.
C_AREA_2	CHAR(3)			Area code for Customer's phone 2.
C_LOCAL_2	CHAR(10)			Local number for Customer's phone 2.
C_EXT_2	CHAR(5)			Extension number for Customer's phone 2.
C_CTRY_3	CHAR(3)			Country code for Customer's phone 3.
C_AREA_3	CHAR(3)			Area code for Customer's phone 3.
C_LOCAL_3	CHAR(10)			Local number for Customer's phone 3.
C_EXT_3	CHAR(5)			Extension number for Customer's phone 3.
C_EMAIL_1	CHAR(50)			Customer's e-mail address 1.
C_EMAIL_2	CHAR(50)			Customer's e-mail address 2.

2.2.4.3 CUSTOMER_ACCOUNT

The CUSTOMER_ACCOUNT table contains account information related to accounts of each customer.

Table Prefix: CA_

Field Name	Field Type	Constraint	Relations	Comment
CA_ID	IDENT_T	Not Null	PK	Customer account identifier.
CA_B_ID	IDENT_T	Not Null	FK (B_)	Broker identifier of the broker who manages this customer account.

CA_C_ID	IDENT_T	Not Null	FK (C_)	Customer identifier of the customer who owns this account.
CA_NAME	CHAR(50)			Name of customer account. Example, "Trish Hogan 401(k)"
CA_TAX_ST	NUM(1)	Not Null in 0,1,2		Tax status of this account: 0 means this account is not taxable, 1 means this account is taxable and tax must be withheld, 2 means this account is taxable and tax does not have to be withheld.
CA_BAL	BALANCE_T	Not Null		Account's cash balance.

2.2.4.4 CUSTOMER_TAXRATE

The table contains two indices per customer into the TAXRATE table. One index is for state/province tax; the other one is for national tax. The TAXRATE table contains the actual tax rates.

Table Prefix: CX_

Field Name	Field Type	Constraint	Relations	Comment
CX_TX_ID	CHAR(4)	Not Null	PK+ FK (TX)	Tax rate identifier.
CX_C_ID	IDENT_T	Not Null	PK+ FK (C_)	Customer identifier of a customer that must pay this tax rate.

2.2.4.5 HOLDING

The table contains information about the customer account's security holdings.

Table Prefix: H_

Field Name	Field Type	Constraint	Relations	Comment
H_T_ID	TRADE_T	Not Null	PK FK (T_)	Trade Identifier of the trade.
H_CA_ID	IDENT_T	Not Null	FK+ (HS_)	Customer account identifier.
H_S_SYMB	CHAR(15)	Not Null	FK+ (HS_)	Symbol for the security held.
H_DTS	DATETIME	Not Null		Date this security was purchased or sold.
H_PRICE	S_PRICE_T	Not Null > 0		Unit purchase price of this security.
H_QTY	S_QTY_T	Not Null		Quantity of this security held.

2.2.4.6 HOLDING_HISTORY

The table contains information about holding positions that were inserted, updated or deleted and which trades made each change.

Table Prefix: HH_

Field Name	Field Type	Constraint	Relations	Comment
HH_H_T_ID	TRADE_T	Not Null	PK+ FK (T_)	Trade Identifier of the trade that originally created the holding row. This is a Foreign Key to the TRADE table rather than the HOLDING table because the HOLDING row could be deleted.
HH_T_ID	TRADE_T	Not Null	PK+ FK (T_)	Trade Identifier of the current trade (the one that last inserted, updated or deleted the holding identified by HH_H_T_ID).
HH_BEFORE_QTY	S_QTY_T	Not Null		Quantity of this security held before the modifying trade. On initial insertion, HH_BEFORE_QTY is 0.
HH_AFTER_QTY	S_QTY_T	Not Null		Quantity of this security held after the modifying trade. If the HOLDING row gets deleted by the modifying trade, then HH_AFTER_QTY is 0.

2.2.4.7 HOLDING_SUMMARY

The table contains aggregate information about the customer account's security holdings.

Table Prefix: HS_

Field Name	Field Type	Constraint	Relations	Comment
HS_CA_ID	IDENT_T	Not Null	PK+ FK (CA_)	Customer account identifier.
HS_S_SYMB	CHAR(15)	Not Null	PK+ FK (S_)	Symbol for the security held.
HS_QTY	S_QTY_T	Not Null		Total quantity of this security held.

Comment: HOLDING_SUMMARY may be implemented as a view on HOLDING, in which case the HOLDING **Foreign Key** references to HOLDING_SUMMARY are automatically met. However, the HOLDING_SUMMARY **Foreign Key** references to CA_ and S_ must then be adopted and met by HOLDING.

2.2.4.8 WATCH_ITEM

The table contains list of securities to watch for a watch list.

Table Prefix: WI_

Field Name	Field Type	Constraint	Relations	Comment
WI_WL_ID	IDENT_T	Not Null	PK+ FK (WL_)	Watch list identifier.
WI_S_SYMB	CHAR(15)	Not Null	PK+ FK (S_)	Symbol of the security to watch.

2.2.4.9 WATCH_LIST

The table contains information about the customer who created this watch list.

Table Prefix: WL_

Field Name	Field Type	Constraint	Relations	Comment
WL_ID	IDENT_T	Not Null	PK	Watch list identifier.
WL_C_ID	IDENT_T	Not Null	FK (C_)	Identifier of customer who created this watch list.

2.2.5 Broker Tables

This group of tables contains data related to the brokerage firm and brokers.

2.2.5.1 BROKER

The table contains information about brokers.

Table Prefix: B_

Field Name	Field Type	Constraint	Relations	Comment
B_ID	IDENT_T	Not Null	PK	Broker identifier.
B_ST_ID	CHAR(4)	Not Null	FK (ST_)	Broker status type identifier; identifies if this broker is active or not.
B_NAME	CHAR(100)	Not Null		Broker's name.
B_NUM_TRADES	NUM(9)	Not Null		Number of trades this broker has executed so far.
B_COMM_TOTAL	BALANCE_T	Not Null		Amount of commission this broker has earned so far.

2.2.5.2 CASH_TRANSACTION

The table contains information about cash transactions.

Table Prefix: CT_

Field Name	Field Type	Constraint	Relations	Comment
CT_T_ID	TRADE_T	Not Null	PK FK (T_)	Trade identifier.
CT_DTS	DATETIME	Not Null		Date and time stamp of when the transaction took place.
CT_AMT	VALUE_T	Not Null		Amount of the cash transaction.
CT_NAME	CHAR(100)			Transaction name, or description: e.g. "Buy Keebler Cookies", "Cash from sale of DuPont stock".

2.2.5.3 CHARGE

The table contains information about charges for placing a trade request. Charges are based on the customer's tier and the trade type.

Table Prefix: CH_

Field Name	Field Type	Constraint	Relations	Comment
------------	------------	------------	-----------	---------

CH_TT_ID	CHAR(3)	Not Null	PK+ FK (TT_)	Trade type identifier.
CH_C_TIER	NUM(1)	Not Null in 1,2,3	PK+	Customer's tier.
CH_CHRG	VALUE_T	Not Null >= 0		Charge for placing a trade request.

2.2.5.4 COMMISSION_RATE

The commission rate depends on several factors: the tier the customer is in, the type of trade, the quantity of securities traded, and the exchange that executes the trade.

Table Prefix: **CR_**

Field Name	Field Type	Constraint	Relations	Comment
CR_C_TIER	NUM(1)	Not Null in 1,2,3	PK+	Customer's tier. Valid values 1, 2 or 3.
CR_TT_ID	CHAR(3)	Not Null	PK+ FK (TT_)	Trade Type identifier. Identifies the type of trade.
CR_EX_ID	CHAR(6)	Not Null	PK+ FK (EX_)	Exchange identifier. Identifies the exchange the trade is against.
CR_FROM_QTY	S_QTY_T	Not Null >= 0	PK+	Lower bound of quantity being traded to match this commission rate.
CR_TO_QTY	S_QTY_T	Not Null > CR_FROM_QTY		Upper bound of quantity being traded to match this commission rate.
CR_RATE	NUM(5,2)	Not Null >= 0		Commission rate. Ranges from 0.00 to 100.00. Example: 10% is 10.00

2.2.5.5 SETTLEMENT

The table contains information about how trades are settled: specifically whether the settlement is on margin or in cash and when the settlement is due.

Table Prefix: **SE_**

Field Name	Field Type	Constraint	Relations	Comment
SE_T_ID	TRADE_T	Not Null	PK FK (T_)	Trade identifier.
SE_CASH_TYPE	CHAR(40)	Not Null		Type of cash settlement involved: possible values "Margin", "Cash Account".
SE_CASH_DUE_DATE	DATE	Not Null		Date by which customer or brokerage must receive the cash; date of trade plus two days.
SE_AMT	VALUE_T	Not Null		Cash amount of settlement (can be zero).

2.2.5.6 TRADE

The table contains information about trades.

Table Prefix: T_

Field Name	Field Type	Constraint	Relations	Comment
T_ID	TRADE_T	Not Null	PK	Trade identifier.
T_DTS	DATETIME	Not Null		Date and time of trade.
T_ST_ID	CHAR(4)	Not Null	FK (ST_)	Status type identifier; identifies the status of this trade.
T_TT_ID	CHAR(3)	Not Null	FK (TT_)	Trade type identifier; identifies the type of his trade.
T_IS_CASH	BOOLEAN	Not Null in FALSE, TRUE		Is this trade a cash or margin trade
T_S_SYMB	CHAR(15)	Not Null	FK (S_)	Security symbol of the security that was traded.
T_QTY	S_QTY_T	Not Null >0		Quantity of securities traded.
T_BID_PRICE	S_PRICE_T	Not Null > 0		The requested unit price.
T_CA_ID	IDENT_T	Not Null	FK (CA_)	Customer account identifier.
T_EXEC_NAME	CHAR(64)	Not Null		Name of the person executing the trade.
T_TRADE_PRICE	S_PRICE_T			Unit price at which the security was traded.
T_CHRG	VALUE_T	>= 0		Fee charged for placing this trade request.
T_COMM	VALUE_T	Not Null >= 0		Commission earned on this trade; may be zero.
T_TAX	VALUE_T	Not Null >= 0		Amount of tax due on this trade; can be zero. Whether the tax is withheld from the settlement amount depends on the customer account tax status.
T_LIFO	BOOLEAN	Not Null in FALSE, TRUE		If this trade is closing an existing position, it must be executed against the newest to oldest account holdings of this security. Default value is 0 (FIFO).

2.2.5.7 TRADE_HISTORY

The table contains the history of each trade transaction through the various states.

Table Prefix: TH_

Field Name	Field Type	Constraint	Relations	Comment
TH_T_ID	TRADE_T	Not Null	PK+ FK (T_)	Trade identifier. This value will be used for the corresponding T_ID in the TRADE and SE_T_ID in the SETTLEMENT table if this trade request results in a trade.

TH_DTS	DATETIME	Not Null		Timestamp of when the trade history was updated.
TH_ST_ID	CHAR(4)	Not Null	PK+ FK (ST_)	Status type identifier.

2.2.5.8 TRADE_REQUEST

The table contains information about pending limit trades that are waiting for a certain security price before the trades are submitted to the market.

Table Prefix: TR_

Field Name	Field Type	Constraint	Relations	Comment
TR_T_ID	TRADE_T	Not Null	PK FK (T_)	Trade request identifier. This value will be used for the corresponding T_ID in the TRADE table and SE_T_ID in the SETTLEMENT table if this trade request results in a trade. This value is also used in TH_T_ID in the TRADE_HISTORY table.
TR_TT_ID	CHAR(3)	Not Null	FK (TT_)	Trade request type identifier; identifies the type of trade.
TR_S_SYMB	CHAR(15)	Not Null	FK (S_)	Security symbol of the security the customer wants to trade.
TR_QTY	S_QTY_T	Not Null > 0		Quantity of security the customer had requested to trade.
TR_BID_PRICE	S_PRICE_T	Not Null > 0		Price the customer wants per unit of security that they want to trade. Value of zero implies the customer wants to trade now at the market price
TR_CA_ID	IDENT_T	Not Null	FK (CA_)	Identifies the customer's account.

2.2.5.9 TRADE_TYPE

The table contains a list of valid trade types.

Table Prefix: TT_

Field Name	Field Type	Constraint	Relations	Comment
TT_ID	CHAR(3)	Not Null	PK	Trade type identifier: Values are: "TMB", "TMS", "TSL", "TLS", and "TLB".
TT_NAME	CHAR(12)	Not Null		Trade type name. Examples "Limit Buy", "Limit Sell", "Market Buy", "Market Sell", "Stop Loss".
TT_IS_SELL	BOOLEAN	Not Null in FALSE, TRUE		TRUE if this is a "Sell" type transaction. FALSE if this is a "Buy" type transaction.
TT_IS_MRKT	BOOLEAN	Not Null in FALSE, TRUE		TRUE if this is a market transaction that must be submitted to the exchange emulator immediately. FALSE if this is a limit transaction.

2.2.6 Market Tables

This group of tables contains information related to the exchanges, companies, and securities that create the Market.

2.2.6.1 COMPANY

The table contains information about all companies with publicly traded securities.

Table Prefix: CO_

Field Name	Field Type	Constraint	Relations	Comment
CO_ID	IDENT_T	Not Null	PK	Company identifier.
CO_ST_ID	CHAR(4)	Not Null	FK (ST_)	Company status type identifier. Identifies if this company is active or not.
CO_NAME	CHAR(60)	Not Null		Company name
CO_IN_ID	CHAR(2)	Not Null	FK(IN_)	Industry identifier of the industry the company is in.
CO_SP_RATE	CHAR(4)	Not Null		Company's credit rating from Standard & Poor.
CO_CEO	CHAR(100)	Not Null		Name of Company's Chief Executive Officer.
CO_AD_ID	IDENT_T	Not Null	FK (AD_)	Address identifier.
CO_DESC	CHAR(150)	Not Null		Company description.
CO_OPEN_DATE	DATE	Not Null		Date the company was founded.

2.2.6.2 COMPANY_COMPETITOR

This table contains information for the competitors of a given company and the industry in which the company competes.

Table Prefix: CP_

Field Name	Field Type	Constraint	Relations	Comment
CP_CO_ID	IDENT_T	Not Null	PK+ FK (CO_)	Company identifier.
CP_COMP_CO_ID	IDENT_T	Not Null	PK+ FK (CO_)	Company identifier of the competitor company for the specified industry.
CP_IN_ID	CHAR(2)	Not Null	PK+ FK (IN_)	Industry identifier of the industry in which the CP_CO_ID company considers that the CP_COMP_CO_ID company competes with it. This may not be either company's primary industry.

2.2.6.3 DAILY_MARKET

The table contains daily market statistics for each security, using the closing market data from the last completed trading day. EGenLoader will load this table with data for each security for the period starting 3 January 2000 and ending 31 December 2004.

Table Prefix: DM_

Field Name	Field Type	Constraint	Relations	Comment
DM_DATE	DATE	Not Null	PK+	Date of last completed trading day.
DM_S_SYMB	CHAR(15)	Not Null	PK+ FK (S_)	Security symbol of this security.
DM_CLOSE	S_PRICE_T	Not Null		Closing price for this security.
DM_HIGH	S_PRICE_T	Not Null		Day's High price for this security.
DM_LOW	S_PRICE_T	Not Null		Day's Low price for this security.
DM_VOL	S_COUNT_T	Not Null		Day's volume for this security.

2.2.6.4 EXCHANGE

The table contains information about financial exchanges.

Table Prefix: EX_

Field Name	Field Type	Constraint	Relations	Comment
EX_ID	CHAR(6)	Not Null	PK	Exchange identifier. Values are, "NYSE", "NASDAQ", "AMEX", "PCX"
EX_NAME	CHAR(100)	Not Null		Exchange name.
EX_NUM_SYMB	NUM(6)	Not Null		Number of securities traded on this exchange.
EX_OPEN	NUM(4)	Not Null		Exchange Daily start time expressed in GMT.
EX_CLOSE	NUM(4)	Not Null		Exchange Daily stop time, expressed in GMT.
EX_DESC	CHAR(150)			Description of the exchange.
EX_AD_ID	IDENT_T	Not Null	FK (AD_)	Mailing address of exchange.

2.2.6.5 FINANCIAL

The table contains information about a company's quarterly financial reports. EGenLoader will load this table with financial information for each company for the Quarters starting 1 January 2000 and ending with the quarter that starts 1 October 2004.

Table Prefix: FI_

Field Name	Field Type	Constraint	Relations	Comment
FI_CO_ID	IDENT_T	Not Null	PK+ FK (CO_)	Company identifier.
FI_YEAR	NUM(4)	Not Null	PK+	Year of the quarter end.
FI_QTR	NUM(1)	Not Null in 1,2,3,4	PK+	Quarter number that the financial information is for: valid values 1, 2, 3, 4.
FI_QTR_START_DATE	DATE	Not Null		Start date of quarter.
FI_REVENUE	FIN_AGG_T	Not Null		Reported revenue for the quarter.
FI_NET_EARN	FIN_AGG_T	Not Null		Net earnings reported for the quarter.
FI_BASIC_EPS	VALUE_T	Not Null		Basic earnings per share reported for the quarter.
FI_DILUT_EPS	VALUE_T	Not Null		Diluted earnings per share reported for the quarter.
FI_MARGIN	VALUE_T	Not Null		Profit divided by revenues for the quarter.
FI_INVENTORY	FIN_AGG_T	Not Null		Value of inventory on hand at the end of the quarter.
FI_ASSETS	FIN_AGG_T	Not Null		Value of total assets at the end of the quarter.
FI_LIABILITY	FIN_AGG_T	Not Null		Value of total liabilities at the end of the quarter.
FI_OUT_BASIC	S_COUNT_T	Not Null		Average number of common shares outstanding (basic).
FI_OUT_DILUT	S_COUNT_T	Not Null		Average number of common shares outstanding (diluted).

2.2.6.6 INDUSTRY

The table contains information about industries. Used to categorize which industries a company is in.

Table Prefix: IN_

Field Name	Field Type	Constraint	Relations	Comment
IN_ID	CHAR(2)	Not Null	PK	Industry identifier.
IN_NAME	CHAR(50)	Not Null		Industry name. Examples: "Air Travel", "Air Cargo", "Software", "Consumer Banking", "Merchant Banking", etc.
IN_SC_ID	CHAR(2)	Not Null	FK(SC_)	Sector identifier of the sector the industry is in.

2.2.6.7 LAST_TRADE

The table contains one row for each security with the latest trade price and volume for each security.

Table Prefix: LT_

Field Name	Field Type	Constraint	Relations	Comment
LT_S_SYMB	CHAR(15)	Not Null	PK FK (S_)	Security symbol.
LT_DTS	DATETIME	Not Null		Date and timestamp of when this row was last updated.
LT_PRICE	S_PRICE_T	Not Null		Latest trade price for this security.
LT_OPEN_PRICE	S_PRICE_T	Not Null		Price the security opened at today.
LT_VOL	S_COUNT_T	Not Null		Volume of trading on the market for this security so far today. Value initialized to 0.

2.2.6.8 NEWS_ITEM

The table contains information about news items of interest.

Table Prefix: NI_

Field Name	Field Type	Constraint	Relations	Comment
NI_ID	IDENT_T	Not Null	PK	News item identifier.
NI_HEADLINE	CHAR(80)	Not Null		News item headline.
NI_SUMMARY	CHAR(255)	Not Null		News item summary.
NI_ITEM	LOB(100000) or LOB_Ref	Not Null		Large object containing the news item or links to the story. LOB_Ref. = Reference to LOB(100000) object stored outside the table on the SUT
NI_DTS	DATETIME	Not Null		Date and time the news item was published.
NI_SOURCE	CHAR(30)	Not Null		Source of the news item.
NI_AUTHOR	CHAR(30)			Author of the news item. May be null if the news item came off a wire service

2.2.6.9 NEWS_XREF

The table contains a cross-reference of news items to companies that are mentioned in the news item.

Table Prefix: NX_

Field Name	Field Type	Constraint	Relations	Comment
NX_NI_ID	IDENT_T	Not Null	PK+ FK (NI_)	News item identifier.
NX_CO_ID	IDENT_T	Not Null	PK+ FK (CO_)	Company identifier of the company (or one of the companies) mentioned in the news item.

2.2.6.10 SECTOR

The table contains information about market sectors.

Table Prefix: SC_

Field Name	Field Type	Constraint	Relations	Comment
SC_ID	CHAR(2)	Not Null	PK	Sector identifier.
SC_NAME	CHAR(30)	Not Null		Sector name. Examples: "Energy", "Materials", "Industrials", "Health Care, etc.

2.2.6.11 SECURITY

This table contains information about each security traded on any of the exchanges.

Table Prefix: S_

Field Name	Field Type	Constraint	Relations	Comment
S_SYMB	CHAR(15)	Not Null	PK	Security symbol used to identify the security on "ticker".
S_ISSUE	CHAR(6)	Not Null		Security issue type. Example: "COMMON", "PERF_A", "PERF_B", etc.
S_ST_ID	CHAR(4)	Not Null	FK (ST_)	Security status type identifier. Identifies if this security is active or not.
S_NAME	CHAR(70)	Not Null		Security name.
S_EX_ID	CHAR(6)	Not Null	FK (EX_)	Exchange identifier of the exchange the security is traded on.
S_CO_ID	IDENT_T	Not Null	FK (CO_)	Company identifier of the company this security is issued by.
S_NUM_OUT	S_COUNT_T	Not Null		Number of shares outstanding for this security.
S_START_DATE	DATE	Not Null		Date security first started trading.
S_EXCH_DATE	DATE	Not Null		Date security first started trading on this exchange.
S_PE	VALUE_T	Not Null		Current share price to earnings per share ratio.
S_52WK_HIGH	S_PRICE_T	Not Null		Security share price 52-week high.
S_52WK_HIGH_DATE	DATE	Not Null		Date of security share price 52-week high.
S_52WK_LOW	S_PRICE_T	Not Null		Security share price 52-week low.
S_52WK_LOW_DATE	DATE	Not Null		Date of security share price 52-week low.
S_DIVIDEND	VALUE_T	Not Null		Annual Dividend per share amount. May be zero, is not allowed to be negative.
S_YIELD	NUM(5,2)	Not Null		Dividend to share price ratio. Value is in percent. Example 10.00 is 10%

2.2.7 Dimension Tables

This group of tables contains generic information that is referenced by multiple fact tables.

2.2.7.1 ADDRESS

This table contains address information.

Table Prefix: AD_

Field Name	Field Type	Constraint	Relations	Comment
AD_ID	IDENT_T	Not Null	PK	Address identifier.
AD_LINE1	CHAR(80)			Address Line 1.
AD_LINE2	CHAR(80)			Address Line 2.
AD_ZC_CODE	CHAR(12)	Not Null	FK (ZC_)	Zip or postal code.
AD_CTRY	CHAR(80)			Country.

2.2.7.2 STATUS_TYPE

This table contains all status values for several different status usages. Multiple tables reference this table to obtain their status values.

Table Prefix: ST_

Field Name	Field Type	Constraint	Relations	Comment
ST_ID	CHAR(4)	Not Null	PK	Status type identifier.
ST_NAME	CHAR(10)	Not Null		Status value. Examples: "Active", "Completed", "Pending", "Canceled" and "Submitted".

2.2.7.3 TAXRATE

The table contains information about tax rates.

Table Prefix: TX_

Field Name	Field Type	Constraint	Relations	Comment
TX_ID	CHAR(4)	Not Null	PK	Tax rate identifier. Format - two letters followed by one digit. Examples: 'US1', 'CA1'.
TX_NAME	CHAR(50)	Not Null		Tax rate name.
TX_RATE	NUM(6,5)	Not Null >= 0		Tax rate, between 0.00000 and 1.00000, inclusive.

2.2.7.4 ZIP_CODE

The table contains zip and postal codes, towns, and divisions that go with them.

Table Prefix: ZC_

Field Name	Field Type	Constraint	Relations	Comment
ZC_CODE	CHAR(12)	Not Null	PK	Postal code.
ZC_TOWN	CHAR(80)	Not Null		Town.
ZC_DIV	CHAR(80)	Not Null		State or province or county.

2.3 Implementation Rules

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

2.3.2 All tables must have the properly scaled number of rows as defined by the database population requirements in Clause 2.6.

2.3.3 Table Partitioning

2.3.3.1 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 **reported** in the **Report**.

2.3.3.2 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 **reported** in the **Report** (see Clause 2.5 for limitations).

2.3.3.3 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.

2.3.4 Replication is allowed for all tables. All copies of tables that are replicated must meet all requirements for atomicity, consistency, and isolation as defined in Clauses 7.2, 7.3 and 7.4. If implemented, the details of such replication must be **reported** in the **Report**.

Comment: Only one copy of a replicated table needs to meet the **Durability** requirements defined in Clause 7.5.

- 2.3.5 Attributes may be added and/or duplicated from one table to another as long as these changes do not improve performance.
- 2.3.6 Each attribute, as described in Clause 2.2 table definitions, must be logically discrete and independently accessible by the data manager. For example, ADDRESS.AD_LINE1 and ADDRESS.AD_LINE2 is not allowed to be implemented as two sub-parts of a discrete attribute ADDRESS.AD_LINE.
- 2.3.7 Each attribute, as described in Clause 2.2 table definitions, must be accessible by the data manager as a single attribute. For example, NEWS_ITEMS.NI_ITEM is not allowed to be implemented as two discrete attributes NEWS_ITEMS.NI_ITEM1 and NEWS_ITEMS.NI_ITEM2.
- 2.3.8 The **Primary Key** of each table must not directly represent the physical disk addresses of the row or any offsets thereof. The application is not allowed to 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 that have provisions for adding, deleting, and modifying records in the ordinary course of processing.
- Comment 1:** It is the intent of this clause that the application program (see Clause 1.2) 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 condition. 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.
- 2.3.9 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 non-**Growing Tables** a minimum number of rows equal to 5% of the table cardinality.
- Comment:** It is required that the space for the additional 5% table cardinality be configured for the **Test Run** and priced (as **Fixed-Space** per Clause 6.5.6.1) accordingly. For systems where space is configured and dynamically allocated at a later time, this space must be considered as allocated and included as **Fixed-Space** when priced.

- 2.3.10 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.
- 2.3.11 Any other rules specified elsewhere in this document apply to the implementation (e.g., the consistency requirements in Clause 7.3).
- 2.3.12 The implementation of LOB in the NEWS_ITEM table may be done either by specific inclusion of the LOB in the table or by use of a reference to a LOB object stored elsewhere on the **System Under Test**. Regardless of the implementation, the LOB object must satisfy the following properties:
- Changes to the data in the object must be under the same transactional control as the changes to the objects of any other type.
 - Recovery after **catastrophic** failure must be capable of restoring all objects, including of type LOB, to the same point in time.
 - The object, and any associated references to it, must be treated as a unit with respect to atomicity.

2.4 Integrity Rules

- 2.4.1 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.
- 2.4.2 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.
- 2.4.3 **Referential Integrity** (RI) must be enforced by the database for all **Foreign Key** (FK) **Primary Key** (PK) relations defined between tables.

Comment: **Referential Integrity** preserves the relationship of data between tables, by restricting actions performed on primary and **Foreign Keys** in a table. **Referential Integrity** prevents removing rows containing **Primary Keys** that are referenced by **Foreign Keys** in other tables in the database. **Referential Integrity** also prevents adding rows containing **Foreign Keys** that refer to **Primary Keys** whose rows are not already present in the database. **Referential Integrity** does not allow modifications to **Primary Key** columns of rows that are referenced by **Foreign Keys** in other tables in the database.

2.5 Data Access Transparency Requirements

Data Access Transparency is the property of the system that removes from the application program any knowledge of the location and access mechanisms of partitioned data. An implementation that 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.

- 2.5.1 Each of the tables described in Clause 2.2 must be identifiable by names that have no relationship to the partitioning of tables. All data manipulation operations in the application program (see Clause 1.2) must use only these names.
- 2.5.2 The system must prevent any data manipulation operation performed using the names described in Clause 2.5.1 that would result in a violation of the integrity rules (see Clause 2.4). For example: the system must prevent a non-TPC-E application from committing the insertion of a row in a vertically partitioned table unless all partitions of that row have been inserted.
- 2.5.3 Using the names which satisfy Clause 2.5.1, any arbitrary non-TPC-E 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 2.5.1 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-E application program uses general-purpose mechanisms to manipulate data in the database.

2.6 Database Size and Table Cardinality

The transaction load generated to service customer accounts and to interact with financial markets drives the throughput of the TPC-E benchmark. To increase the throughput, more customers and their associated data must be configured. The cardinality of the CUSTOMER table is the basis of the TPC-E database size and scaling. CUSTOMER table cardinality is determined based on the transaction throughput metric requirements defined in Clause 6.5.7.

Configured Customers means the number of customers (with corresponding rows in the associated tables) configured at database generation.

The TPC-E benchmark has three types of sizing requirements for its tables:

- **Fixed Tables:** These tables always have the same number of rows regardless of the database size and transaction throughput. For example, TRADE_TYPE has five rows.
- **Scaling Tables:** These tables each have a cardinality that has a constant relationship to the cardinality of the CUSTOMER table. **Transactions** may update rows from these tables, but the table sizes remain constant.
- **Growing Tables:** These tables each have an initial cardinality that has a constant relationship to the cardinality of the CUSTOMER table. However, the cardinality increases with new growth during the benchmark run at a rate that is proportional to transaction throughput rates. The HOLDING table has been included in this category. Rows are added and deleted from the HOLDING table during the benchmark run, but the average size of the table continues to grow during the benchmark at a rate that is proportional to transaction throughput rates. The TRADE_REQUEST table is also considered a Growing table, even though its average size is a fixed relationship to the transaction throughput rates and not to the cardinality of the CUSTOMER table.

2.6.1 Initial Database Size Requirements

- 2.6.1.1 The test database must be initially populated using data generated by **EGenLoader**. By definition, the TPC provided **EGenLoader** produces the correct number of rows for each table. The test database must be built including the initial database population and all indices/auxiliary data structures present during the **Test Run**.
- 2.6.1.2 The initial database population is based on the number of customers. The benchmark **Sponsor** selects the CUSTOMER table cardinality, based on the desired transaction throughput. Clause 6.5.7.2 defines the **Nominal Throughput** that may be **reported** for a given number of rows in the CUSTOMER table. The minimum number of rows for the CUSTOMER table is 5000. The size of the CUSTOMER table can be increased in increments of 1000 customers. A set of 1000 customers is known as a **load unit**.
- 2.6.1.3 The **Growing Tables** are populated with an initial set of rows sufficient to enable all benchmark **transactions** to run.
- 2.6.1.4 The only changes that can be made to the content of the database tables between the initial population and a valid **Test Run** is the running of **Valid transactions**, as defined in this specification.
- 2.6.1.5 The Scale Factor (**SF**) is the number of customer rows (500) per single Transaction-Per-Second-E (tpsE).

- 2.6.1.6 The **Initial Trade Days (ITD)** is the number of **Business Days** used to populate the database. This population is made of trade data that would be generated by the **SUT** when running at the **Nominal Throughput** for the specified number of **Business Days**. The number of **Initial Trade Days** is 300.
- 2.6.1.7 The number of **load units** configured must be equal to the number of **load units** actually accessed during the **Test Run**.
- 2.6.1.8 The following variables are used as an aid in defining table cardinalities:

Variable	Table	Description
<i>customers</i>	CUSTOMER	Number of rows in the CUSTOMER table.
<i>accounts</i>	CUSTOMER_ACCOUNT	Number of rows in the CUSTOMER_ACCOUNT table. Equal to 5 * <i>customers</i> .
<i>trades</i>	TRADE	Number of trade rows in the TRADE table. The trades number is equal to 17280 * <i>customers</i> (300 days of initial population at SF = 500).
<i>settled</i>	SETTLEMENT	Number of settled trade rows in the SETTLEMENT table. The settled number is equal to <i>trades</i> .
<i>companies</i>	COMPANY	Number of rows in the COMPANY table. 500 companies per load unit of 1000 customers.
<i>securities</i>	SECURITY	Number of rows in the SECURITY table. 685 securities per load unit of 1000 customers.

- 2.6.1.9 The following rules are used by **EGenLoader** to calculate the cardinalities of the **Scaling Tables** and **Growing Tables**. The **EGen** package uses random number generators to set the number of rows for relationships such as accounts per customer or securities per account and, as a result, the cardinality of some tables can only be approximated.

Table	Variable Used	Rule
ACCOUNT_PERMISSION	<i>accounts</i>	60% have just the customer as the executor 38% have the customer and 1 other executor 2% have the customer and 2 other executors Avg. is $\sim 1.42 * \text{accounts}$
ADDRESS	<i>customers</i>	$\text{companies} + \text{EXCHANGE}(4) + \text{customers}$
BROKER	<i>customers</i>	$0.01 * \text{customers}$ (or 1 broker per 100 <i>customers</i>)
CASH_TRANSACTION	<i>settled</i>	$\sim 0.92 * \text{settled}$ (84% of buys and 100% of sells are cash)
COMPANY	<i>customers</i>	$500 * (\text{customers} / 1000)$
COMPANY_COMPETITOR	<i>companies</i>	$3 * \text{companies}$
CUSTOMER_ACCOUNT	<i>customers</i>	Min=1 Max=10 Avg=5 * <i>customers</i>
CUSTOMER_TAXRATE	<i>customers</i>	$2 * \text{customers}$
DAILY_MARKET	<i>securities</i>	$\text{securities} * 1,305$ (5 years of 5-day work weeks with two leap years)
FINANCIAL	<i>companies</i>	$\text{companies} * 20$ quarters (5 years)
HOLDING	<i>settled</i>	$\sim 0.051 * \text{settled}$ (assumes ITD = 300 and SF = 500)

HOLDING_HISTORY	<i>settled</i>	~1.340 * <i>settled</i> (assumes ITD = 300 and SF = 500)
HOLDING_SUMMARY	<i>accounts</i>	~9.956 * <i>accounts</i> (assumes ITD = 300 and SF = 500)
LAST_TRADE	<i>securities</i>	1 * <i>securities</i>
NEWS_ITEM	<i>companies</i>	2 * <i>companies</i>
NEWS_XREF	<i>companies</i>	2 * <i>companies</i>
SECURITY	<i>customers</i>	685 * (<i>customers</i> /1000)
SETTLEMENT	<i>settled</i>	1 * <i>settled</i>
TRADE	<i>customers</i>	17280 * <i>customers</i> = ((ITD * 8 * 3600) / SF) * <i>customers</i>
TRADE_HISTORY	<i>settled</i>	~((2 rows per market trade) * 0.6) + ((3 rows per limit trade) * 0.4) Avg is (2.4 * <i>settled</i>)
TRADE_REQUEST		0
WATCH_LIST	<i>customers</i>	Each customer has one watch list (1 * <i>customers</i>)
WATCH_ITEM	<i>customers</i>	Avg=100 items per watch list * <i>customers</i>

2.6.1.10 The following list contains the cardinality of **Fixed Tables**.

Fixed Tables	Cardinality	Comments
CHARGE	15	5 trade types * 3 customer tiers
COMMISSION_RATE	240	4 rates * 4 exchanges * 5 trade types * 3 customer tiers
EXCHANGE	4	4 exchanges
INDUSTRY	102	102 industries
SECTOR	12	12 sectors
STATUS_TYPE	5	5 status types
TAXRATE	321	321 tax rates
TRADE_TYPE	5	5 trade types
ZIP_CODE	14,741	14,741 zip codes

2.6.1.11 The following list contains the cardinality of the **Scaling Tables**.

Scaling Tables	Cardinality	Cardinality Formula
CUSTOMER	5,000	Scaled based on transaction rate
CUSTOMER_TAXRATE	10,000	<i>customers</i> * 2
CUSTOMER_ACCOUNT	25,000	<i>accounts</i> = (Avg of 5 * <i>customers</i>)
ACCOUNT_PERMISSION	~35,500	<i>accounts</i> * (Average of ~1.42 permissions per account)
ADDRESS	7,504	<i>companies</i> + EXCHANGE (4) + <i>customers</i>
BROKER	50	<i>customers</i> * 0.01

COMPANY	2,500	500 * (<i>customers</i> /1000)
COMPANY_COMPETITOR	7,500	<i>companies</i> * 3
DAILY_MARKET	4,469,625	<i>securities</i> * 1,305
FINANCIAL	50,000	<i>companies</i> * 20
HOLDING_SUMMARY	~248,900	~9.956 * <i>accounts</i>
LAST_TRADE	3,425	<i>securities</i> * 1
NEWS_ITEM	5,000	<i>companies</i> * 2
NEWS_XREF	5,000	<i>companies</i> * 2
SECURITY	3,425	685 * (<i>customers</i> /1000)
WATCH_LIST	5,000	<i>customers</i> * 1
WATCH_ITEM	~ 500,000	<i>customers</i> * (Average of ~100 <i>securities</i> per watch list)

2.6.1.12 The following list shows the initial cardinality of the **Growing Tables**.

Growing Tables	Cardinality	Comment
CASH_TRANSACTION	~79,488,000	~0.92 * <i>settled</i> (84% of buys & 100% of sells are cash)
HOLDING	~4,406,400	~0.051 * <i>settled</i> (assumes ITD = 300 and SF = 500)
HOLDING_HISTORY	~115,776,000	~1.340 * <i>settled</i> (assumes ITD = 300 and SF = 500)
SETTLEMENT	86,400,000	1 * <i>settled</i>
TRADE	86,400,000	((ITD * 8hr/ day * 3600sec/hr * <i>customers</i>) / SF)
TRADE_HISTORY	~207,360,000	~(2.4 * <i>trades</i>)
TRADE_REQUEST	0	0

2.6.2 Runtime Database Size Requirements

2.6.2.1 The following list shows the increase in rows per second for the **Growing Tables** during runtime. The rate of growth may decline after running for a large number of days.

Table Name	Comment
CASH_TRANSACTION	~0.92 * (<i>customers</i> /SF)
HOLDING	~0.044 * (<i>customers</i> /SF)
HOLDING_HISTORY	~1.343 * (<i>customers</i> /SF)
SETTLEMENT	1 * (<i>customers</i> /SF)
TRADE	1 * (<i>customers</i> /SF)
TRADE_HISTORY	~2.4 * (<i>customers</i> /SF)
TRADE_REQUEST	~(60 to 70) * (<i>customers</i> /SF)

-
- 2.6.2.2 The test database must be built to sustain the **reported throughput rating** during a **Business Day**. This means that test database must have a **Business Day's** worth of additional space for data, index and log online. This excludes performing on the database any operation that does not occur during the **Measurement Interval**.

CLAUSE 3 -- TRANSACTIONS

3.1 Introduction

The core of each TPC-E **transaction** runs on the **Database Server**, but the logic of the **transaction** interacts with several components of the benchmark environment. This section defines all aspects of the **transactions**, including side effects on other components of the benchmark environment.

3.1.1 Definitions

3.1.1.1 A **transaction** is composed of **EGenTxnHarness** and of the invocation of one or more **Frames**. The Trade-Cleanup **transaction** is an exception. **Sponsors** may but do not have to run the Trade-Cleanup **transaction** from **EGenTxnHarness**.

3.1.1.2 The **EGenTxnHarness** is the TPC provided transaction logic, which the **sponsor** is not allowed to alter. The **EGenTxnHarness** is implemented in a manner that precludes the consolidation of multiple **Frames** within a **transaction**.

3.1.1.3 A **Frame** is the **sponsor** implemented transaction logic, which is invoked as a unit of execution by the **EGenTxnHarness**. The database interactions of a **transaction** are all initiated from within its **Frames**.

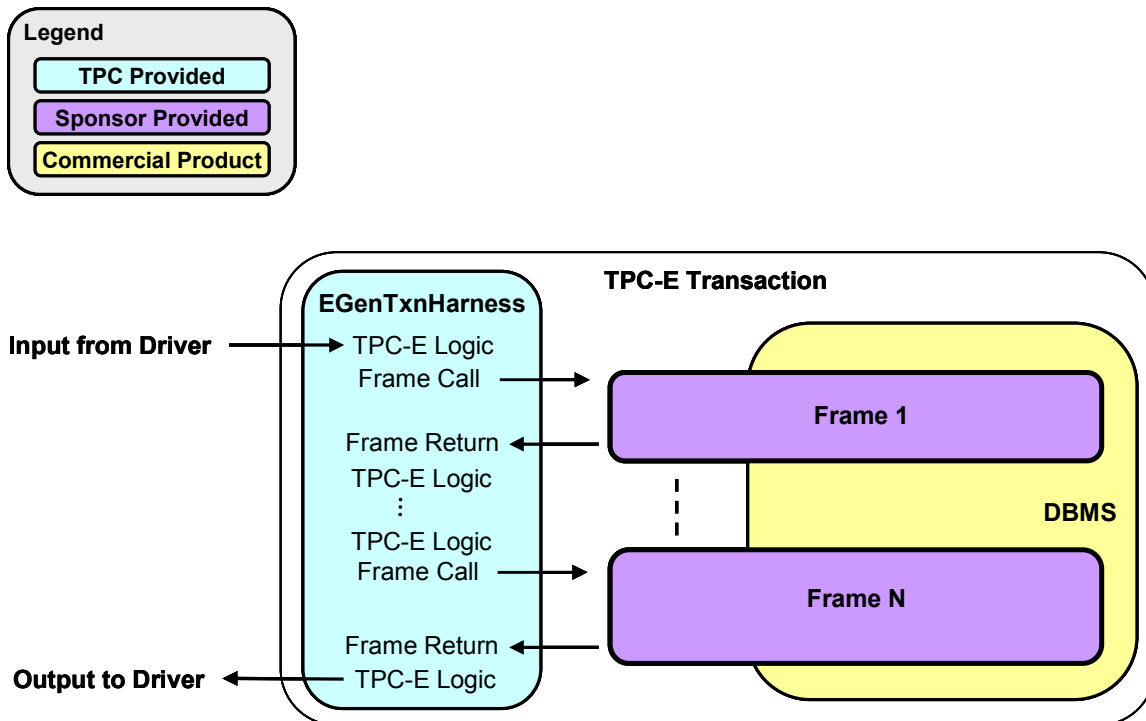


Figure 3.a - Frames interfacing with the Harness and the Database

3.1.1.4 A **Database-Transaction** is composed of one or more database interactions enclosed between a "start" and a "commit" or "rollback".

3.1.2 Database-Footprint Definition

This Clause describes the format used to specify the **Database-Footprint** of each **transaction** in this benchmark.

3.1.2.1 The **Database-Footprint** of a **transaction** is the set of required database interactions to be executed by that **transaction**.

3.1.2.2 Each **Database-Footprint** is presented in a tabular format where the columns specify the following:

- The first column denotes either one of the database tables defined in Clause 2.2 or the word “**Transaction**” that denotes the entire **transaction**. The first row following the table header defines the overall **transaction**.
- The second column denotes one of the following:
 - A specific column name of a database table as defined in Clause 2.2.
 - The string “**# rows**” that specifies the exact number of rows containing all columns of a database table. For example, “2 rows” indicates two complete rows of a database table.
 - The string “**Row(s)**” that specifies a variable number of rows containing all columns of a database table.
- The remaining columns correspond with each of the **Frames** of the **transaction** and contain the database interactions or **transaction** control operations required to be executed in that **Frame**.

3.1.2.3 The following table is an example of the **Database-Footprint** of a **transaction**.

Example Database-Footprint				
Table	Column	Frame		
		1	2*	3*
CUSTOMER_ACCOUNT	CA_BAL	Reference		
	CA_C_ID	Return		
	CA_TAX_ST	Return		
HOLDING	H_PRICE		Return	
	H_QTY		Modify	
	Row(s)		Remove *	
	1 row		Add *	
TRADE_HISTORY	1 row			Add
Transaction Control		Start	Rollback *	Commit

- For the first row of the **Database-Footprint** where the word “**Transaction**” appears, each column corresponds to one of the **transaction Frames**. The content of the columns denote which **transaction** control operations occur in that **Frame**. The possible **transaction** control operations are as follows:
 - The word “**Start**” indicates that the specified **Frame** contains a control operation that starts a **Database-Transaction**. The start of a **Database-Transaction** can only occur in a **Frame** where the word “**Start**” is specified.

- The word “**Rollback**” indicates that the specified **Frame** contains a control operation that rolls back the **Database-Transaction**. The explicit rolling back of a **Database-Transaction** can only occur in a **Frame** where the word “**Rollback**” is specified.
- The word “**Commit**” indicates that the specified **Frame** contains a control operation that commits the **Database-Transaction**. Committing a **Database-Transaction** can only occur in a **Frame** where the word “**Commit**” is specified.

Comment: Multiple **transaction** control operations may occur within the same **Frame**. For example, a **transaction** that consists of a single **Frame** would have both “**Start**” and “**Commit**” in its **Database-Footprint** column corresponding with **Frame** 1.

- For remaining rows of the **Database-Footprint** the column corresponding to each **Frame** contains the access method required for the table column listed in that row. The possible access methods are as follows:
 - The word “**Reference**” indicates that the table column is identified in the database and the content is accessed within the **Frame** without passing the content of the table column to the **EGenTxnHarness**.
 - The word “**Return**” indicates that the table column is referenced and that its content is retrieved from the database and passed to the **EGenTxnHarness**. The table column must be referenced in the same **Frame** where the word “**Return**” is specified. The content of the table column can only be passed to subsequent **Frames** via the input and output parameters specified in the **Frame** parameters.
 - The word “**Modify**” indicates that the content of a table column is modified within the **Frame**. The content of the table column can only be changed in a **Frame** where the word “**Modify**” is specified. When the original content of the table column must also be referenced or returned before it is modified, a “**Reference**” or a “**Return**” access method is also specified.
 - The word “**Add**” indicates that a number of rows are added to the table specified by the **Database-Footprint**. Table row(s) can only be added in a **Frame** where the word “**Add**” is specified. The number of rows that are added is specified in the second column of the **Database-Footprint** with either “**# row**” for a fixed number of rows or “**row(s)**” for an unspecified number of rows.
 - The word “**Remove**” indicates that a number of rows are removed from the table specified by the **Database-Footprint**. Table row(s) can only be removed in a **Frame** where the word “**Remove**” is specified. The number of rows that are removed is specified in the second column of the **Database-Footprint** with either “**# row**” for a fixed number of rows or “**row(s)**” for an unspecified number of rows.

Comment 1: An asterisk following any item in the column of a given **Frame** denotes that the transaction control, the database interactions, or the execution of the entire **Frame** is conditional. The **EGenTxnHarness** defines under which conditions the **Frame** will be executed.

Comment 2: In the example **Database-Footprint** above, the **Database-Transaction** is started in **Frame 1**. If **Frame 2** is executed the **Database-Transaction** may be rolled back. If **Frame 3** is executed the **Database-Transaction** must be committed. For the table CUSTOMER_ACCOUNT, the table column CA_BAL is referenced and the table columns CA_C_ID and CA_TAX_ST are returned in **Frame 1**. For the HOLDING table, the column H_PRICE is returned and H_QTY is modified if **Frame 2** is executed. Additionally, if **Frame 2** is executed, a number of rows are conditionally removed from the HOLDING table and 1 row is conditionally added to the HOLDING table. For the TRADE_HISTORY table, a row is added if **Frame 3** is executed.

3.2 Transaction Implementation Rules

3.2.1 Frame Implementation

- 3.2.1.1 The implementation of a **Frame** is not allowed to assume any prior knowledge of values for data elements defined in the database schema for the benchmark.

Comment: The intent of this clause is to prevent the Frames from using constant values, or other means, to circumvent database references to static or infrequently changing data elements. In general, using any private knowledge specific to the benchmark, but which is not explicitly furnished to the transaction or the Frame, is prohibited.

- 3.2.1.2 All data exchanges between **Frames** must be done by the **EGenTxnHarness** through its use of input and output parameters passed in and out of the **Frames**.

Comment 1: The intent of this clause is to prevent the **Frames** from using global variables, or other means, for storing and retrieving information across multiple invocations of the same or different **Frames** in order to avoid work intended to be done during each individual invocation.

Comment 2: The **test sponsor** may augment each **Frame** with code to unpack the input parameters received from the **EGenTxnHarness** and to pack the output parameters returned to the **EGenTxnHarness**.

- 3.2.1.3 The **Frame Implementation** must perform each database interaction specified in the **transaction's Database-Footprint**, using the specified access method.

- 3.2.1.4 The **Frame Implementation** must access any column that is marked as **Reference**. It is also free to access other columns that are not marked as **Reference**. For the other database interactions, the **Frame Implementation** must perform all the required operations and/or return all the specified column values.

- 3.2.1.5 The implementation of each **Frame** must be functionally equivalent to the pseudo-code provided for that **Frame** in Clause 3.3.

3.3 The Transactions

The TPC-E benchmark consists of eleven **transactions**, and one cleanup **transaction**. To generate a reasonably balanced workload that resembles real production environments, the **transactions** have to cover a wide variety of system functions. Ten of the **transactions** follow a specific mix to generate the desired workload while keeping the benchmark environment simple, repeatable and easy to execute. The eleventh **transaction** is not part of the **transaction Mix**, but is executed at fixed intervals. This **transaction**, called “Data-Maintenance”, simulates administrative updates to tables that are not otherwise modified by the **transactions** in the mix. A cleanup **transaction**, called “Trade-Cleanup”, is provided to clean up pending and submitted trades that may exist from an earlier run.

One of the key performance characteristics of database systems is the ratio of reads and writes generated by the workload. To emulate such a ratio, we have defined **transactions** with read-only profiles as well as **transactions** with read-write profiles. In addition, the **transactions** apply varying loads on the processor.

The variety of processor, IO, and execution frequency requirements for the **transactions** allows the benchmark to emulate a real environment with heavy processor utilization while maintaining a reasonable IO load in a simple benchmark configuration.

The **transactions** in the mix can be grouped into three categories:

- **Customer initiated:** These **transactions** simulate customer interactions with the system and are initiated by the **Customer-Emulator** component of the benchmark **Driver**.
- **Brokerage initiated:** These **transactions** simulate broker interactions with the system and are initiated by the **Customer-Emulator** component of the benchmark **Driver**.
- **Market triggered:** These **transactions** simulate the behavior of the market and are triggered by the **Market-Exchange-Emulator** component of the benchmark **Driver**.

In addition to the mix of **transactions** above, the benchmark defines a time triggered Data-Maintenance **transaction**, which is initiated at fixed time intervals as defined in Clause 6.2.2.4. Also defined is a Trade-Cleanup **transaction** (see clause 6.2.2.5), which may not be executed within a **Test Run**, but must be executed once before a **Test Run** if the database is not in its initially populated state (i.e., if any prior runs have been performed on the database).

The following summary table lists the basic characteristics of the **transactions**:

Transaction	Weight	Access	Category	Frames	Definition
Trade-Order	Heavy	Read-write	Customer initiated	6	Clause 3.3.1
Trade-Result	Heavy	Read-write	Market triggered	6	Clause 3.3.2
Trade-Lookup	Medium	Read-only	Brokerage initiated	4	Clause 3.3.3
Trade-Update	Medium	Read-write	Brokerage initiated	3	Clause 3.3.4
Trade-Status	Light	Read-only	Customer initiated	1	Clause 3.3.5
Customer-Position	Mid to Heavy	Read-only	Customer initiated	3	Clause 3.3.6
Broker-Volume	Mid to Heavy	Read-only	Brokerage initiated	1	Clause 3.3.7
Security-Detail	Medium	Read-only	Customer initiated	1	Clause 3.3.8
Market-Feed	Medium	Read-write	Market triggered	1	Clause 3.3.9
Market-Watch	Medium	Read-only	Customer initiated	1	Clause 3.3.10
Data-Maintenance	Light	Read-write	Time triggered	1	Clause 3.3.11

Trade-Cleanup	Medium	Read-write	Run once before Test Run	1	Clause 3.3.12
---------------	--------	------------	--------------------------	---	---------------

3.3.1 The Trade-Order Transaction

The Trade Order **transaction** is designed to emulate the process of ordering the trade, buy or sell, of a security by a Customer, Broker, or authorized third-party. This would be equivalent to a customer placing a trade request to the brokerage house to execute a trade on a security from their account. If the person executing the trade order is not the account owner, the **transaction** will verify that the person has the appropriate authorization to perform the trade order. The **transaction** allows the person trading to execute buys at the current market price, sells at the current market price, or limit buys and sells. The **transaction** also provides an estimate of the financial impact of the proposed trade by providing profit/loss data, tax implications, and anticipated commission fees. This allows the trader to evaluate the desirability of the proposed security trade. The trader may then decide to submit the trade or cancel it.

The Trade-Order **transaction** is invoked by the **CE** component of the **Driver**. The **transaction** starts by using the account ID passed into the **transaction** to obtain information on the customer, the customer's account, and the broker for the account. This information comes from the CUSTOMER, CUSTOMER_ACCOUNT, and BROKER tables.

Next, the **transaction** conditionally validates that the person executing the trade is authorized to perform such actions on the specified account. This information is in the ACCOUNT_PERMISSION table. If the executor is not authorized, then the **transaction** rolls back. However, during the benchmark execution, the **CE** will always generate authorized executors.

The next step is to estimate the overall financial implications of executing the trade. For limit-orders, the requested price is used in the estimation; for market orders, the requested price is set to the current market value of the security and that value is used in the estimation. Estimating includes assessing any effects the requested trade would have on existing holdings (e.g. the sale of existing long positions, or the cover of existing short positions). This information is in the HOLDING table. If a profit would be realized as a result of this trade the capital gains taxes are calculated based on information in the TAXRATE and CUSTOMER_TAXRATE tables. Administrative fees and the broker's commission for handling the trade are calculated. This information comes from the CHARGE and COMMISSION_RATE tables. If the **transaction** is being done on margin, the customer's total assets for the account are assessed based on information in the CUSTOMER_ACCOUNT, HOLDING, and LAST_TRADE tables. This information is used for recording the order into the TRADE, TRADE_REQUEST, and TRADE_HISTORY tables.

After all the above processing has completed, a small percentage of the Trade-Order **transactions** are selected to exercise the rollback functionality by rolling back all their modifications. All other Trade-Order **transactions** are committed. After a successfully committed market order, the **EGenTxnHarness** sends the order for the trade to the appropriate **MEE**.

The Trade-Order **transaction** is divided into 6 **Frames**.

3.3.1.1 Trade-Order Transaction Parameters

The inputs to the Trade-Order **transaction** are generated by the **EGen CE** code in *CETxnInputGenerator.cpp*. The data structures used to communicate the input and output parameters must match the **EGenTxnHarness** data structures defined in *TxnHarnessStructs.h*.

Trade-Order Interfaces	Module/Data Structure
CE Input generation	GenerateTradeOrderInput()
Transaction Input/Output Structure	TTradeOrderTxnInput TTradeOrderTxnOutput
Frame 1 Input/Output Structure	TTradeOrderFrame1Input TTradeOrderFrame1Output
Frame 2 Input/Output Structure	TTradeOrderFrame2Input TTradeOrderFrame2Output
Frame 3 Input/Output Structure	TTradeOrderFrame3Input TTradeOrderFrame3Output
Frame 4 Input/Output Structure	TTradeOrderFrame4Input TTradeOrderFrame4Output
Frame 5 Input/Output Structure	TTradeOrderFrame5Output
Frame 6 Input/Output Structure	TTradeOrderFrame6Output

Trade-Order Transaction Parameters:

Field	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. A single customer account id, as defined by CA_ID in CUSTOMER_ACCOUNT, is chosen at random, uniformly, from the range of customer account ids for the chosen customer.
co_name	IN	The security being traded in this transaction can be specified in one of two ways. Either by specifying the security's symbol, or by specifying the company name and the issue. If the symbol is used to specify the security, then the company name and the issue must be an empty string (i.e. ""). Otherwise the company name and the issue are both specified and the symbol must be then be an empty string (i.e. ""). For more information, see Clause 6.3.1.2.
exec_f_name	IN	First name of the person executing the trade. Note that the person executing this trade, may not be the registered owner of the account. If this is the case, the executor's permission to execute trades for this account will be verified in Frame 2. For more information, see Clause 6.3.1.2.
exec_l_name	IN	Last name of the person executing the trade. Note that the person executing this trade, may not be the registered owner of the account. If this is the case, the executor's permission to execute trades for this account will be verified in Frame 2. For more information, see Clause 6.3.1.2.
exec_tax_id	IN	Tax identifier for the person executing the trade. Note that the person executing this trade, may not be the registered owner of the account. If this is the case, the executor's permission to execute trades for this account will be verified in Frame 2. For more information, see Clause 6.3.1.2.
is_lifo	IN	If this flag is set to TRUE then this trade will process against existing holdings from oldest to newest (LIFO order). If this flag is set to FALSE, then this trade will process against existing holdings from newest to oldest (FIFO order).
issue	IN	The security being traded in this transaction can be specified in one of two ways. Either by specifying the security's symbol, or by specifying the company name and the issue. If the symbol is used to specify the security, then the company name and the issue must be an empty string (i.e. ""). Otherwise the company name and the issue are both specified and the symbol must be then be an empty string (i.e. "").

		"""). For more information, see Clause 6.3.1.2.
requested_price	IN	For a limit order, this is the requested price for triggering the trade. For a market order, the input value is undefined and this variable must be set to the current market price for the given security inside Frame 3.
roll_it_back	IN	A flag indicating whether an intentional rollback of this transaction will occur. See Clause 6.3.1.2 for details on the percentage of trades that will be intentionally rolled back.
st_pending_id	IN	Identifier for the "Pending" order status – passed in for ease of benchmarking.
st_submitted_id	IN	Identifier for the "Submitted" order status – passed in for ease of benchmarking.
symbol	IN	The security being traded in this transaction can be specified in one of two ways. Either by specifying the security's symbol, or by specifying the company name and the issue. If the symbol is used to specify the security, then the company name and the issue must be an empty string (i.e. ""). Otherwise the company name and the issue are both specified and the symbol must be then be an empty string (i.e. ""). For more information, see Clause 6.3.1.2.
trade_qty	IN	The number of shares to be traded for this order.
trade_type_id	IN	Identifier indicating the type of trade - passed in for each of benchmarking. For more information on the different types of trades generated, see Clause 6.3.1.2.
type_is_margin	IN	If this flag is set to TRUE, then the order will be done on margin. If the flag is set to FALSE, then this trade will be done with cash.
buy_value	OUT	The total dollar amount for the securities bought for a matching sell order. If trade is a buy or sell of new securities then buy_value is zero.
sell_value	OUT	The total dollar value of the securities sold for a matching buy order. If trade is buy or sell of new securities then sell_value is zero.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.
tax_amount	OUT	The estimated amount of tax that will be incurred as a result of this order. If no profit is realized, then tax_amount is zero.
trade_id	OUT	Unique trade identifier generated by the SUT for this order.

3.3.1.2 Trade-Order Transaction Database-Footprint

This **transaction** includes a mixture of Add, Reference, and Return access methods. The Trade-Order **Database-Footprint** is as follows:

Trade-Order Database-Footprint							
Table	Column	Frame					
		1	2*	3	4	5*	6*
ACCOUNT_PERMISSION	AP_ACL		Return				
	AP_CA_ID		Reference				
	AP_F_NAME		Reference				
	AP_L_NAME		Reference				
	AP_TAX_ID		Reference				
BROKER	B_NAME	Return					
CHARGE	CH_CHRG			Return			
COMMISSION_RATE	CR_RATE			Return			
COMPANY	CO_ID			Reference*			

	CO_NAME			Return*			
CUSTOMER	C_F_NAME	Return					
	C_L_NAME	Return					
	C_TIER	Return					
	C_TAX_ID	Return					
CUSTOMER_ACCOUNT	CA_BAL			Reference*			
	CA_C_ID	Return					
	CA_NAME	Return					
	CA_TAX_ST	Return					
CUSTOMER_TAXRATE	CX_TX_ID			Reference*			
HOLDING	H_PRICE			Reference			
	H_QTY			Reference			
HOLDING_SUMMARY	HS_QTY			Reference			
LAST_TRADE	LT_PRICE			Return			
SECURITY	S_CO_ID			Reference*			
	S_EX_ID			Reference			
	S_NAME			Return			
	S_SYMB			Return*			
TAXRATE	TX_RATE			Reference*			
TRADE	1 Row				Add		
TRADE_HISTORY	1 Row				Add		
TRADE_REQUEST	1 Row				Add*		
TRADE_TYPE	TT_IS_MRKT			Return			
	TT_IS_SELL			Return			
Transaction Control		Start	Rollback*			Rollback	Commit

3.3.1.3 Trade-Order Transaction Frame 1 of 6

The first **Frame** is responsible for retrieving information about the customer, customer account, and its broker.

The **EGenTxnHarness** controls the execution of **Frame 1** as follows:

```
{
    invoke (Trade-Order_Frame-1)
}
```

Trade-Order Frame 1 of 6 Parameters:

Field	Direction	Description
acct_id	IN	Identifier of the customer account involved in the transaction.
acct_name	OUT	Name of the account specified by acct_id.
broker_name	OUT	Name of the broker associated with the specified acct_id.

cust_f_name	OUT	First name of the customer who owns the specified account. This output string must not contain trailing white space.
cust_id	OUT	Unique identifier of the customer who owns the specified account.
cust_l_name	OUT	Last name of the customer who owns the specified account. This output string must not contain trailing white space.
cust_tier	OUT	The brokerage house service level tier this customer belongs to.
status	OUT	Code indicating the execution status for this frame.
tax_id	OUT	Tax identifier for the customer who owns the specified account. This output string must not contain trailing white space.
tax_status	OUT	Tax status of the customer who owns the specified account.

Trade-Order_Frame-1 pseudo-code: Get customer, customer account, and broker information

```

{
    start transaction

    // Get account, customer, and broker information
    Declare broker_id  IDENT_T
    select
        acct_name  = CA_NAME,
        broker_id  = CA_B_ID,
        cust_id    = CA_C_ID,
        tax_status = CA_TAX_ST
    from
        CUSTOMER_ACCOUNT
    where
        CA_ID = acct_id

    select
        cust_f_name = C_F_NAME,
        cust_l_name = C_L_NAME,
        cust_tier   = C_TIER,
        tax_id      = C_TAX_ID
    from
        CUSTOMER
    where
        C_ID = cust_id

    select
        broker_name = B_NAME
    from
        BROKER
    where
        B_ID=broker_id
}

```

3.3.1.4 Trade-Order Transaction Frame 2 of 6

The second **Frame** is conditionally executed when the **transaction** executor's first name, last name, and tax id do not match the customer first name, customer last name, and customer tax id returned in **Frame 1**. **Frame 2** is responsible for validating the executor's permission to order trades for the specified customer account.

The database access methods used in **Frame 2** are all **References**.

```
{
    if (exec_l_name <> cust_l_name or
        exec_f_name <> cust_f_name or
        exec_tax_id <> tax_id) then {
        invoke (Trade-Order_Frame-2)
        if (!ap_acl) then {
            // This code is here for "completeness".
            // During the benchmark execution the CE always
            // generates an executor with valid permissions on
            // the account.
            return error
        }
    }
}
```

Trade-Order Frame 2 of 6 Parameters:

Field	Direction	Description
acct_id	IN	Identifier of the customer account involved in the transaction.
exec_f_name	IN	First name of the person executing the trade.
exec_l_name	IN	Last name of the person executing the trade.
exec_tax_id	IN	Tax identifier for the person executing the trade.
ap_acl	OUT	Account permission access control list string for this executor on this customer account. If a NULL string is returned, then the executor of this transaction does not have permission to execute trades for the specified account.
status	OUT	Code indicating the execution status for this frame.

Trade-Order_Frame-2 pseudo-code : Check executor's permission

```
{
    select
        ap_acl = AP_ACL
    from
        ACCOUNT_PERMISSION
    where
        AP_CA_ID = acct_id and
```

Trade-Order_Frame-2 pseudo-code : Check executor's permission

```
AP_F_NAME = exec_f_name and
AP_L_NAME = exec_l_name and
AP_TAX_ID = exec_tax_id

if (ap_acl is NULL) then {
    rollback
}
}
```

3.3.1.5 Trade-Order Transaction Frame 3 of 6

The third **Frame** is responsible for estimating the overall impact of executing the requested trade. Profit and loss estimates are calculated and capital gains taxes are estimated based on any profits. Administrative fees and commission rates are obtained. If this is a margin trade, the customer's assets needed to cover the cost of the trade are calculated using current market values.

The database access methods used in **Frame 3** are **References** and **Returns**.

The **EGenTxnHarness** controls the execution of **Frame 3** as follows:

```
{
    invoke (Trade-Order_Frame-3)
}
```

Trade-Order Frame 3 of 6 Parameters:

Field	Direction	Description
acct_id	IN	Identifier of the customer account involved in the transaction.
cust_id	IN	The brokerage house service-level (or tier) to which this customer belongs.
cust_tier	IN	The brokerage house service level tier this customer belongs to.
is_lifo	IN	If this flag is set to TRUE then this trade will process against existing holdings from oldest to newest (LIFO order). If this flag is set to FALSE, then this trade will process against existing holdings from newest to oldest (FIFO order).
issue	IN	Specifies the particular issue of security for the given company. This value is an empty string (i.e. "") if the security is specified by symbol.
st_pending_id	IN	Identifier for the "Pending" order status - passed in for ease of benchmarking.
st_submitted_id	IN	Identifier for the "Submitted" order status - passed in for ease of benchmarking.
tax_status	IN	Tax status of the customer who owns the specified account.
trade_qty	IN	The number of shares to be traded for this order.
trade_type_id	IN	Identifier indicating the type of trade - passed in for ease of benchmarking.
type_is_margin	IN	If this flag is set to TRUE, then the order will be done on margin. If the flag is set to FALSE, then this trade will be done with cash.
co_name	IN-OUT	Name of the company for the security being traded. Otherwise, if the trade is being done based on symbol, then co_name is an empty string (i.e. "")

		and will be set appropriately inside the frame. This output string must not contain trailing white space.
requested_price	IN-OUT	For a limit order, this is the requested price for triggering the trade. For a market order, the input value is undefined and this variable must be set to the current market price for the given security.
symbol	IN-OUT	The stock symbol for the security being traded. Otherwise, if the trade is being done based on co_name and issue, then symbol is an empty string (i.e. "") and will be set appropriately inside the frame. This output string must not contain trailing white space.
buy_value	OUT	The total dollar amount for the securities bought for a matching sell order. If trade is a buy or sell of new securities then buy_value is zero.
charge_amount	OUT	The fee charged by the brokerage house for processing this trade.
comm_rate	OUT	The broker's commission rate for processing this trade.
cust_assets	OUT	If this trade is being done on margin, this will be set to the sum of the cash balance and the current market value of all holdings in the specified account.
market_price	OUT	The current market trading price of the security.
s_name	OUT	The full name of the security. This output string must not contain trailing white space.
sell_value	OUT	The total dollar value of the securities sold for a matching buy order. If trade is buy or sell of new securities then sell_value is zero.
status	OUT	Code indicating the execution status for this frame.
status_id	OUT	Identifier indicating the status of this order (either pending or submitted). This output string must not contain trailing white space.
tax_amount	OUT	The estimated amount of tax that will be incurred as a result of this order. If no profit is realized, then tax_amount is zero.
type_is_market	OUT	Flag set to TRUE for market orders and to FALSE for limit orders.
type_is_sell	OUT	Flag set to TRUE for sell orders and to FALSE for buy orders.

Trade-Order_Frame-3 pseudo-code: Estimate overall effects of the trade

```

{
  Declare co_id    IDENT_T
  Declare exch_id  CHAR(6)

  // Get information on the security
  if (symbol == "") then {
    select
      co_id = CO_ID
    from
      COMPANY
    where
      CO_NAME = co_name

    select
      exch_id = S_EX_ID,
      s_name  = S_NAME,

```

Trade-Order_Frame-3 pseudo-code: Estimate overall effects of the trade

```
        symbol  = S_SYMB
    from
        SECURITY
    where
        S_CO_ID = co_id and
        S_ISSUE = issue

} else {
    select
        co_id    = S_CO_ID,
        exch_id  = S_EX_ID,
        s_name   = S_NAME
    from
        SECURITY
    where
        S_SYMB = symbol

    select
        co_name = CO_NAME
    from
        COMPANY
    where
        CO_ID = co_id
}

// Get current pricing information for the security
select
    market_price = LT_PRICE
from
    LAST_TRADE
where
    LT_S_SYMB = symbol

// Set trade characteristics based on the type of trade.
select
    type_is_market = TT_IS_MRKT,
    type_is_sell   = TT_IS_SELL
from
    TRADE_TYPE
where
    TT_ID = trade_type_id

// If this is a limit-order, then the requested_price was passed in to us, but
// if this this a market-order, then we need to set the requested_price to the
// current market price.
if( type_is_market ) then {
    requested_price = market_price
}

// Local frame variables used when estimating impact of this trade on
```

Trade-Order_Frame-3 pseudo-code: Estimate overall effects of the trade

```
// any current holdings of the same security.
Declare hold_price S_PRICE_T
Declare hold_qty   S_QTY_T
Declare needed_qty S_QTY_T
Declare hs_qty     S_QTY_T

// Initialize variables
buy_value = 0.0
sell_value = 0.0
needed_qty = trade_qty

select
    hs_qty = HS_QTY
from
    HOLDING_SUMMARY
where
    HS_CA_ID = account and
    HS_S_SYMB = symbol

if (type_is_sell) then {
    // This is a sell transaction, so estimate the impact to any currently held
    // long positions in the security.
    //
    if (hs_qty > 0) then {
        if (is_lifo) then {
            // Estimates will be based on closing most recently acquired holdings
            // Could return 0, 1 or many rows
            declare hold_list cursor for
            select
                H_QTY,
                H_PRICE
            from
                HOLDING
            where
                H_CA_ID = acct_id and
                H_S_SYMB = symbol
            order by
                H_DTS desc
        } else {
            // Estimates will be based on closing oldest holdings
            // Could return 0, 1 or many rows
            declare hold_list cursor for
            select
                H_QTY,
                H_PRICE
            from
                HOLDING
            where
                H_CA_ID = acct_id and
                H_S_SYMB = symbol
```

Trade-Order_Frame-3 pseudo-code: Estimate overall effects of the trade

```
        order by
            H_DTS asc
    }

    // Estimate, based on the requested price, any profit that may be realized
    // by selling current holdings for this security. The customer may have
    // multiple holdings for this security (representing different purchases of
    // this security at different times and therefore, most likely, different
    // prices).
    open hold_list
    do until (needed_qty = 0 or end_of_hold_list) {
        fetch from
            hold_list
        into
            hold_qty,
            hold_price
        if (hold_qty > needed_qty) then {
            // Only a portion of this holding would be sold as a result of the
            // trade.
            buy_value += needed_qty * hold_price
            sell_value += needed_qty * requested_price
            needed_qty = 0
        } else {
            // All of this holding would be sold as a result of this trade.
            buy_value += hold_qty * hold_price
            sell_value += hold_qty * requested_price
            needed_qty = needed_qty - hold_qty
        }
    }
    close hold_list
}

// NOTE: If needed_qty is still greater than 0 at this point, then the
// customer would be liquidating all current holdings for this security, and
// then short-selling this remaining balance for the transaction.
} else {

    // This is a buy transaction, so estimate the impact to any currently held
    // short positions in the security. These are represented as negative H_QTY
    // holdings. Short positions will be covered before opening a long position in
    // this security.
    if (hs_qty < 0) then { // Existing short position to buy
        if (is_lifo) then {
            // Estimates will be based on closing most recently acquired holdings
            // Could return 0, 1 or many rows
            declare hold_list cursor for
                select
                    H_QTY,
                    H_PRICE
                from
                    HOLDING
```

Trade-Order_Frame-3 pseudo-code: Estimate overall effects of the trade

```
        where
            H_CA_ID = acct_id and
            H_S_SYMB = symbol
        order by
            H_DTS desc
    } else {
        // Estimates will be based on closing oldest holdings
        // Could return 0, 1 or many rows
        declare hold_list cursor for
            select
                H_QTY,
                H_PRICE
            from
                HOLDING
            where
                H_CA_ID = acct_id and
                H_S_SYMB = symbol
            order by
                H_DTS asc
    }

    // Estimate, based on the requested price, any profit that may be realized
    // by covering short positions currently held for this security. The customer
    // may have multiple holdings for this security (representing different
    // purchases of this security at different times and therefore, most
    // likely, different prices).
    open hold_list
    do until (needed_qty = 0 or end_of_hold_list) {
        fetch from
            hold_list
        into
            hold_qty,
            hold_price
        if (hold_qty + needed_qty < 0) then {
            // Only a portion of this holding would be covered (bought back) as
            // a result of this trade.
            sell_value += needed_qty * hold_price
            buy_value  += needed_qty * requested_price
            needed_qty = 0
        } else {
            // All of this holding would be covered (bought back) as
            // a result of this trade.
            // NOTE: Local variable hold_qty is made positive for easy
            // calculations
            hold_qty  = -hold_qty
            sell_value += hold_qty * hold_price
            buy_value  += hold_qty * requested_price
            needed_qty = needed_qty - hold_qty
        }
    }
}
```

Trade-Order_Frame-3 pseudo-code: Estimate overall effects of the trade

```
        close hold_list
    }
    // NOTE: If needed_qty is still greater than 0 at this point, then the
    // customer would cover all current short positions for this security,
    // (if any) and then open a new long position for the remaining balance
    // of this transaction.
}

// Estimate any capital gains tax that would be incurred as a result of this
// transaction.
tax_amount = 0.0
if ((sell_value > buy_value) and
    ((tax_status == 1) or (tax_status == 2))) then {
    //
    // Customer's can be (are) subject to more than one tax rate.
    // For example, a state tax rate and a federal tax rate. Therefore,
    // get all tax rates the customer is subject to, and estimate overall
    // amount of tax that would result from this order.
    //
    Declare tax_rates    S_PRICE_T
    select
        tax_rates = sum(TX_RATE)
    from
        TAXRATE
    where
        TX_ID in (
            select
                CX_TX_ID
            from
                CUSTOMER_TAXRATE
            where
                CX_C_ID = cust_id)
    tax_amount = (sell_value - buy_value) * tax_rates
}

// Get administrative fees (e.g. trading charge, commission rate)
select
    comm_rate = CR_RATE
from
    COMMISSION_RATE
where
    CR_C_TIER = cust_tier and
    CR_TT_ID = trade_type_id and
    CR_EX_ID = exch_id and
    CR_FROM_QTY <= trade_qty and
    CR_TO_QTY >= trade_qty
select
    charge_amount = CH_CHRG
from
    CHARGE
```

Trade-Order_Frame-3 pseudo-code: Estimate overall effects of the trade

```
where
    CH_C_TIER = cust_tier and
    CH_TT_ID = trade_type_id

// Compute assets on margin trades
Declare acct_bal    BALANCE_T
Declare hold_assets S_PRICE_T

cust_assets = 0.0
if (type_is_margin) then {
    select
        acct_bal = CA_BAL
    from
        CUSTOMER_ACCOUNT
    where
        CA_ID = acct_id

    // Should return 0 or 1 row
    select
        hold_assets = sum(HS_QTY * LT_PRICE)
    from
        HOLDING_SUMMARY,
        LAST_TRADE
    where
        HS_CA_ID = acct_id and
        LT_S_SYMB = HS_S_SYMB

    if (hold_assets is NULL)    /* account currently has no holdings */
        cust_assets = acct_bal
    else
        cust_assets = hold_assets + acct_bal
}

// Set the status for this trade
if (type_is_market) then {
    status_id = st_submitted_id
} else {
    status_id = st_pending_id
}
}
```

3.3.1.6 Trade-Order Transaction Frame 4 of 6

The fourth **Frame** is responsible for creating an audit trail record of the order and assigning a unique trade ID to it.

The database access methods used in **Frame 4** are all **Adds**.

```

{
    // Estimate the total commission amount for this trade.
    comm_amount = (comm_rate / 100) * trade_qty * requested_price
    exec_name = exec_f_name + " " + exec_l_name
    is_cash = !(type_is_margin)
    invoke (Trade-Order_Frame-4)
}

```

Trade-Order Frame 4 of 6 Parameters:

Field	Direction	Description
acct_id	IN	Identifier of the customer account involved in the transaction.
charge_amount	IN	The fee charged by the brokerage house for processing this trade.
comm_amount	IN	The broker's commission for processing this trade.
exec_name	IN	First and last name of the person executing this trade.
is_cash	IN	If this flag is set to TRUE, then this trade will be done with cash. If this flag is set to FALSE, then this trade will be done on margin.
is_lifo	IN	If this flag is set to TRUE then this trade will process against existing holdings from oldest to newest (LIFO order). If this flag is set to FALSE, then this trade will process against existing holdings from newest to oldest (FIFO order).
requested_price	IN	For a limit trade, this is the requested price for triggering action. For a market order, this has been set by the harness code to the current market price for the given security.
status_id	IN	Identifier indicating the status of this order (either pending or submitted).
symbol	IN	The stock symbol for the security being traded.
trade_qty	IN	The number of shares to be traded for this order.
trade_type_id	IN	Identifier indicating the type of trade to be executed.
type_is_market	IN	Flag set to TRUE for market orders and to FALSE for limit orders.
status	OUT	Code indicating the execution status for this frame.
trade_id	OUT	Unique trade identifier generated by the SUT for this order.

Trade-Order_Frame-4 pseudo-code: Record the trade request by making all related updates

```

{
    // Get the timestamp and unique trade ID for this trade.
    Declare now_dts    DATETIME
    get_current_dts ( now_dts )
    get_new_trade_id ( trade_id )

    // Record trade information in TRADE table.
    insert into
        TRADE (
            T_ID, T_DTS, T_ST_ID, T_TT_ID, T_IS_CASH,
            T_S_SYMB, T_QTY, T_BID_PRICE, T_CA_ID, T_EXEC_NAME,

```

Trade-Order_Frame-4 pseudo-code: Record the trade request by making all related updates

```
T_TRADE_PRICE, T_CHRG, T_COMM, T_TAX, T_LIFO
)
values (
    trade_id,           // T_ID
    now_dts,            // T_DTS
    status_id,          // T_ST_ID
    trade_type_id,      // T_TT_ID
    is_cash,            // T_IS_CASH
    symbol,              // T_S_SYMB
    trade_qty,          // T_QTY
    requested_price,     // T_BID_PRICE
    acct_id,            // T_CA_ID
    exec_name,          // T_EXEC_NAME
    NULL,               // T_TRADE_PRICE
    charge_amount,      // T_CHRG
    comm_amount         // T_COMM
    0,                  // T_TAX
    is_lifo             // T_LIFO
)
// Record pending trade information in TRADE_REQUEST table if this trade is a
// limit trade
if (!type_is_market) {
    insert into
        TRADE_REQUEST (
            TR_T_ID, TR_TT_ID, TR_S_SYMB,
            TR_QTY, TR_BID_PRICE, TR_CA_ID
        )
    values (
        trade_id,       // TR_T-ID
        trade_type_id,  // TR_TT_ID
        symbol,          // TR_S_SYMB
        trade_qty,       // TR_QTY
        requested_price, // TR_BID_PRICE
        acct_id          // TR_CA_ID
    )
}

// Record trade information in TRADE_HISTORY table.
insert into
    TRADE_HISTORY (
        TH_T_ID, TH_DTS, TH_ST_ID
    )
values (
    trade_id,          // TH_T_ID
    now_dts,           // TH_DTS
    status_id          // TH_ST_ID
)
}
```

Trade-Order_Frame-4 pseudo-code: Record the trade request by making all related updates

3.3.1.7 Trade-Order Transaction Frame 5 of 6

The fifth **Frame** is conditionally executed when the parameter `roll_it_back` is set to `TRUE`. This **Frame** is responsible for intentionally rolling back all database updates from this **transaction**, occasionally exercising the rollback functionality.

There are no database access methods used in **Frame 5**. This **Frame** is only using **transaction** control operations.

The **EGenTxnHarness** controls the execution of **Frame 5** as follows:

```
{
    if (roll_it_back) then {
        invoke (Trade-Order_Frame-5)
        exit // Rest of transaction and send_to_market are skipped
    }
}
```

Trade-Order Frame 5 of 6 Parameters:

Field	Direction	Description
status	OUT	Code indicating the execution status for this frame.

Trade-Order_Frame-5 pseudo-code: Rollback database transaction

```
{
    // Intentional rollback of transaction caused by driver (CE).
    rollback transaction
}
```

3.3.1.8 Trade-Order Transaction Frame 6 of 6

The sixth **Frame** is conditionally executed when parameter `roll_it_back` is set to `FALSE`. This **Frame** is responsible for committing all database updates from this **transaction**.

There are no database access methods used in **Frame 6**. This **Frame** is only using **transaction** control operations.

The **EGenTxnHarness** controls the execution of **Frame 6** as follows:

```

{
    invoke (Trade-Order_Frame-6)

    if (type_is_market) then {
        eAction = eMEEProcessOrder
    }
    else {
        eAction = eMEESetLimitOrderTrigger
    }

    // Send the trade to the Market-Exchange-Emulator (MEE)
    send_to_market (
        requested_price,
        symbol,
        trade_id,
        trade_qty,
        trade_type_id,
        eAction
    )
}

```

Trade-Order Frame 6 of 6 Parameters:

Field	Direction	Description
status	OUT	Code indicating the execution status for this frame.

Trade-Order Frame 6: Commit database transaction

```

{
    commit transaction
}

```

3.3.2 The Trade-Result Transaction

The Trade-Result **transaction** represents the completion of a stock market trade. This is similar to when a brokerage house receives the final price of a trade from the stock market and replaces its estimates for the broker commission, and other similar quantities with the actual numbers. The customer's holdings are updated to reflect that the trade has completed and historical information about the trade is recorded for later reference.

The Trade-Result **transaction** is submitted when the **MEE** component of the **Driver** completes a trade and sends the result back to the **SUT**. The **transaction** starts by using the trade ID passed into the **transaction** to obtain information about the trade from the **TRADE** table. The information gathered includes the account ID of the customer account the trade is for. The account ID is used to lookup account information from the **CUSTOMER_ACCOUNT** table.

Next, the **transaction** gets a timestamp (trade_dts) from the database.

If the trade is a sell, the account's existing holdings in the security being traded are sold first. This results in updates and/or deletes from the **HOLDING** table and inserts into the **HOLDING_HISTORY** table. If the account did not have enough of the security to cover the quantity of the sale, the uncovered quantity is added as a negative position to the **HOLDING** table and a row is inserted into the **HOLDING_HISTORY** table. The trade_dts timestamp is used as the value for **H_DTS** for the insert into the **HOLDING** table.

If the trade is a buy, then check if the account has any negative holdings for the security being traded. A negative holding indicates a previous short sell. The buy trade will cover the short sell. If the buy quantity is greater than or equal to the negative holdings, then delete the holdings and insert into the **HOLDING_HISTORY** table. If the buy quantity is less than the negative holdings, update the negative holding quantity to the difference between it and the buy quantity. Insert a row into the **HOLDING_HISTORY** table.

Continue to work on a buy trade. If this account has no more negative positions for this security, and all the buy quantity has not been used up, the customer is buying new holdings. A new **HOLDING** record will be created using trade_dts for **H_DTS**. A record will be inserted into the **HOLDING_HISTORY** table.

Next, the **transaction** conditionally computes the amount of tax due by the customer as a result of the trade. The computation is only made if the trade resulted in a gain and the account's tax status is 1 or 2. The calculation is performed by first getting the sum of the tax rates the customer is subject to, from the **CUSTOMER_TAXRATE** table. The amount of gain from the trade is multiplied by the customer's tax rate to find the tax_amount.

The next step is to get the commission rate for the trade from the **COMMISSION_RATE** table.

The next **Frame** records the results of the trade and the broker's commission. The **TRADE** table is updated with the commission, completed status and the actual trade price. A row is inserted into the **TRADE_HISTORY** table. The **BROKER** table is updated with the trade commission and one is added to the number of trades year to date.

The last **Frame** settles the trade by inserting a row into the **SETTLEMENT** table. If the trade is for cash a row is also inserted into the **CASH_TRANSACTION** table and the account balance is updated in the **CUSTOMER_ACCOUNT** table.

The Trade-Result **transaction** is divided into 6 **Frames**.

3.3.2.1 Trade-Result Transaction Parameters

The inputs to the Trade-Result **transaction** are generated by the **EGen MEE** code in **MEE.cpp**. The data structures used to communicate the input and output parameters must match the **EGen EGenTxnHarness** data structures defined in **TxnHarnessStructs.h**.

Trade-Result Interfaces	Module/Data Structure
MEE Input generation	CMEESUTInterface::TradeResult()
Transaction Input/Output Structure	TTradeResultTxnInput TTradeResultTxnOutput
Frame 1 Input/Output Structure	TTradeResultFrame1Input

	TTradeResultFrame1Output
Frame 2 Input/Output Structure	TTradeResultFrame2Input TTradeResultFrame2Output
Frame 3 Input/Output Structure	TTradeResultFrame3Input TTradeResultFrame3Output
Frame 4 Input/Output Structure	TTradeResultFrame4Input TTradeResultFrame4Output
Frame 5 Input/Output Structure	TTradeResultFrame5Input TTradeResultFrame5Output
Frame 6 Input/Output Structure	TTradeResultFrame6Input TTradeResultFrame6Output

Trade-Result Transaction Parameters:

Field	Direction	Description
trade_id	IN	The Trade ID for the trade to be settled. Trade ID is the primary key of the TRADE table.
trade_price	IN	The price of the trade.
st_completed_id	IN	The index ID value into STATUS_TYPE for “Completed” status.
acct_bal	OUT	Customer account’s cash balance after the trade was completed.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.

3.3.2.2 Trade-Result Transaction Database-Footprint

This **transaction** includes a mixture of Reference, Return, Modify, Remove and Add operations. The Trade-Result **Database-Footprint** is as follows:

Trade-Result Database-Footprint							
Table	Column	Frame					
		1	2	3*	4	5	6
BROKER	B_COMM_TOTAL					Reference Modify	
	B_NUM_TRADES					Reference Modify	
CASH_TRANSACTION	1 row						Add *
COMMISSION_RATE	CR_RATE				Return		
CUSTOMER	C_TIER				Reference		
CUSTOMER_ACCOUNT	CA_BAL						Return Reference* Modify*
	CA_B_ID		Return				
	CA_C_ID		Return				
	CA_TAX_ST		Return				
CUSTOMER_TAXRATE	CX_TX_ID			Reference			
HOLDING	H_PRICE		Reference				

	H_QTY		Reference Modify*				
	row(s)		Remove*				
	1 row		Add*				
HOLDING_SUMMARY	HS_QTY	Reference	Modify*				
	1 row		Remove*				
	1 row		Add*				
HOLDING_HISTORY	Row(s)		Add				
SECURITY	S_EX_ID				Reference		
	S_NAME				Reference		
SETTLEMENT	1 row						Add
TAX_RATE	TX_RATE			Reference			
TRADE	T_CA_ID	Return					
	T_CHRG	Return					
	T_COMM					Modify	
	T_IS_CASH	Return					
	T_LIFO	Return					
	T_QTY	Return					
	T_S_SYMB	Return					
	T_ST_ID					Modify	
	T_TAX			Modify			
	T_TRADE_PRICE					Modify	
	T_TT_ID	Return					
TRADE_HISTORY	1 row					Add	
TRADE_TYPE	TT_IS_MRKT	Return					
	TT_IS_SELL	Return					
	TT_NAME	Return					
Transaction Control		Start					Commit

3.3.2.3 Trade-Result Transaction Frame 1 of 6

The first **Frame** is responsible for retrieving information about the customer and its trade.

The database access methods used in **Frame 1** are all **Returns**.

The **EGenTxnHarness** controls the execution of **Frame 1** as follows:

```
{
    invoke (Trade-Result_Frame-1)
}
```

Trade-Result Frame 1 of 6 Parameters:

Field	Direction	Description
-------	-----------	-------------

trade_id	IN	The trade ID for the trade to be settled passed to the transaction by the Market-Exchange-Emulator.
acct_id	OUT	Customer account ID of the customer account involved in Trade-Result transaction.
charge	OUT	Fee charged for placing this trade request.
hs_qty	OUT	Current quantity of shares of the security being traded, that the customer holds in their account.
is_lifo	OUT	If this trade is closing an existing position, then is_lifo boolean indicates if trade is to be executed against the newest to oldest customer holdings (TRUE), or in FIFO order of oldest to newest (FALSE).
status	OUT	Code indicating the execution status for this frame.
symbol	OUT	Seven character identifier of security that is being traded. This output string must not contain trailing white space.
trade_is_cash	OUT	Boolean indicating trade is for cash, (TRUE), or on margin, (FALSE).
trade_qty	OUT	Quantity of securities traded
type_id	OUT	Trade type identifier, (T_TT_ID). This output string must not contain trailing white space.
type_is_market	OUT	Boolean indicating trade type is a market transaction, (TRUE), limit trade (FALSE)
type_is_sell	OUT	Boolean indicating if this is a sell trade, (TRUE), or a buy trade, (FALSE)
type_name	OUT	Trade type name

Trade-Result_Frame-1 pseudo-code: Get info on the trade and the customer's account

```

{
  start transaction

  select
    acct_id      = T_CA_ID,
    type_id      = T_TT_ID,
    symbol       = T_S_SYMB,
    trade_qty    = T_QTY,
    charge       = T_CHRG,
    is_lifo      = T_LIFO,
    trade_is_cash = T_IS_CASH
  from
    TRADE
  where
    T_ID = trade_id

  select
    type_name      = TT_NAME,
    type_is_sell   = TT_IS_SELL,
    type_is_market = TT_IS_MRKT
  from
    TRADE_TYPE

```

Trade-Result_Frame-1 pseudo-code: Get info on the trade and the customer's account

```
where
    TT_ID = type_id

select
    hs_qty = HS_QTY
from
    HOLDING_SUMMARY
where
    HS_CA_ID = acct_id and
    HS_S_SYMB = symbol

if (hs_qty is NULL) then    // no prior holdings exist
    hs_qty = 0
}
```

3.3.2.4 Trade-Result Transaction Frame 2 of 6

The second **Frame** is responsible for modifying the customer's holdings to reflect the result of a buy or a sell trade.

The database access methods used in **Frame 2** are a mixture of **References**, **Modifies**, **Removes** and **Adds**.

The EGenTxnHarness controls the execution of **Frame 2** as follows:

```
{
    invoke (Trade-Result_Frame-2)
}
```

Trade-Result Frame 2 of 6 Parameters:

Field	Direction	Description
acct_id	IN	Customer account ID of the customer account involved in the Trade-Result transaction obtained in Frame 1
hs_qty	IN	Current quantity of shares of the security being traded, that the customer holds in their account.
is_lifo	IN	If this trade is closing an existing position, then is_lifo boolean indicates if trade is to be executed against the newest to oldest customer holdings (TRUE), or in FIFO order of oldest to newest (FALSE).
symbol	IN	Seven character security identifier obtained in Frame 1
trade_id	IN	The trade ID for the trade to be settled passed to the transaction by the Market- Exchange-Emulator. Used for insert(s) into the HOLDING and HOLDING_HISTORY tables.
trade_price	IN	The price of the trade passed to the Trade-Result Transaction by the Market-Exchange-Emulator.
trade_qty	IN	Quantity of securities traded obtained form Frame 1
type_is_sell	IN	Boolean obtained in Frame 1 indicating if this is a sell trade, (TRUE), or a buy trade, (FALSE)

broker_id	OUT	ID of the broker who executed the trade.
buy_value	OUT	The total dollar amount for the securities bought for a matching sell order. If trade is a buy or sell of new securities then buy_value is zero.
cust_id	OUT	Customer ID of the customer who owns the customer account involved in the trade.
sell_value	OUT	The total dollar value of the securities sold for a matching buy order. If trade is buy or sell of new securities then sell_value is zero.
status	OUT	Code indicating the execution status for this frame.
tax_status	OUT	Customer account tax status
trade_dts	OUT	Date and time of trade result generated by the SUT.

Trade-Result_Frame-2 pseudo-code: Update the customer's holdings for buy or sell

```

{
    // Local Frame Variables
    Declare hold_id    IDENT_T
    Declare hold_price S_PRICE_T
    Declare hold_qty   S_QTY_T
    Declare needed_qty S_QTY_T
    get_current_dts ( trade_dts )

    // Initialize variables
    buy_value = 0.0
    sell_value = 0.0
    needed_qty = trade_qty

    select
        broker_id = CA_B_ID,
        cust_id   = CA_C_ID,
        tax_status = CA_TAX_ST
    from
        CUSTOMER_ACCOUNT
    where
        CA_ID = acct_id

    // Determine if sell or buy order
    if (type_is_sell) then {

        if (hs_qty == 0) then    // no prior holdings exist, but one will be inserted
            insert into
                HOLDING_SUMMARY (
                    HS_CA_ID,
                    HS_S_SYMB,
                    HS_QTY
                )
            values (
                acct_id,

```

Trade-Result_Frame-2 pseudo-code: Update the customer's holdings for buy or sell

```
        symbol,
        -trade_qty
    )
else
    if (hs_qty != trade_qty) then
        update
            HOLDING_SUMMARY
        set
            HS_QTY = hs_qty - trade_qty
        where
            HS_CA_ID = acct_id and
            HS_S_SYMB = symbol

// Sell Trade:

// First look for existing holdings, H_QTY > 0
if (hs_qty > 0) {
    if (is_lifo) then {
        // Could return 0, 1 or many rows
        declare hold_list cursor for
            select
                H_T_ID,
                H_QTY,
                H_PRICE
            from
                HOLDING
            where
                H_CA_ID = acct_id and
                H_S_SYMB = symbol
            order by
                H_DTS desc
    } else {
        // Could return 0, 1 or many rows
        declare hold_list cursor for
            select
                H_T_ID,
                H_QTY,
                H_PRICE
            from
                HOLDING
            where
                H_CA_ID = acct_id and
                H_S_SYMB = symbol
            order by
                H_DTS asc
    }
    // Liquidate existing holdings. Note that more than
    // 1 HOLDING record can be deleted here since customer
    // may have the same security with differing prices.
```

Trade-Result_Frame-2 pseudo-code: Update the customer's holdings for buy or sell

```
open hold_list
do until (needed_qty = 0 or end_of_hold_list) {
  fetch from
    hold_list
  into
    hold_id,
    hold_qty,
    hold_price
  if (hold_qty > needed_qty) then {
    //Selling some of the holdings
    insert into
      HOLDING_HISTORY (
        HH_H_T_ID,
        HH_T_ID,
        HH_BEFORE_QTY,
        HH_AFTER_QTY
      )
    values (
      hold_id,           // H_T_ID of original trade
      trade_id,          // T_ID current trade
      hold_qty,          // H_QTY now
      hold_qty - needed_qty // H_QTY after update
    )

    update
      HOLDING
    set
      H_QTY = hold_qty - needed_qty
    where
      current of hold_list

    buy_value += needed_qty * hold_price
    sell_value += needed_qty * trade_price
    needed_qty = 0
  } else {
    // Selling all holdings
    insert into
      HOLDING_HISTORY (
        HH_H_T_ID,
        HH_T_ID,
        HH_BEFORE_QTY,
        HH_AFTER_QTY
      )
    values (
      hold_id,          // H_T_ID original trade
      trade_id,         // T_ID current trade
      hold_qty,         // H_QTY now
```

Trade-Result_Frame-2 pseudo-code: Update the customer's holdings for buy or sell

```

        0          // H_QTY after delete
    )

    delete from
        HOLDING
    where
        current of hold_list

    buy_value += hold_qty * hold_price
    sell_value += hold_qty * trade_price
    needed_qty = needed_qty - hold_qty
}
}
close hold_list
}

// Sell Short:
// If needed_qty > 0 then customer has sold all existing
// holdings and customer is selling short. A new HOLDING
// record will be created with H_QTY set to the negative
// number of needed shares.
if (needed_qty > 0) then {
    insert into
        HOLDING_HISTORY (
            HH_H_T_ID,
            HH_T_ID,
            HH_BEFORE_QTY,
            HH_AFTER_QTY
        )
    values (
        trade_id,          // T_ID current is original trade
        trade_id,          // T_ID current trade
        0,                 // H_QTY before
        (-1) * needed_qty  // H_QTY after insert
    )

    insert into
        HOLDING (
            H_T_ID,
            H_CA_ID,
            H_S_SYMB,
            H_DTS,
            H_PRICE,
            H_QTY
        )
    values (
        trade_id,          // H_T_ID
        acct_id,           // H_CA_ID
        symbol,            // H_S_SYMB

```

Trade-Result_Frame-2 pseudo-code: Update the customer's holdings for buy or sell

```
        trade_dts,          // H_DTS
        trade_price,        // H_PRICE
        (-1) * needed_qty   /* H_QTY
    )
else
    if (hs_qty = trade_qty) then
        delete from
            HOLDING_SUMMARY
        where
            HS_CA_ID  = acct_id and
            HS_S_SYMB = symbol
    }
} else {    // The trade is a BUY
    if (hs_qty == 0) then    // no prior holdings exist, but one will be inserted
        insert into
            HOLDING_SUMMARY (
                HS_CA_ID,
                HS_S_SYMB,
                HS_QTY
            )
        values (
            acct_id,
            symbol,
            trade_qty
        )
    else    // hs_qty != 0
        if (-hs_qty != trade_qty) then
            update
                HOLDING_SUMMARY
            set
                HS_QTY = hs_qty + trade_qty
            where
                HS_CA_ID  = acct_id and
                HS_S_SYMB = symbol

// Short Cover:
// First look for existing negative holdings, H_QTY < 0,
// which indicates a previous short sell. The buy trade
// will cover the short sell.
if (hs_qty < 0) then {
    if (is_lifo) then {
        // Could return 0, 1 or many rows
        declare hold_list cursor for
            select
                H_T_ID,
                H_QTY,
                H_PRICE
            from
                HOLDING
```

Trade-Result_Frame-2 pseudo-code: Update the customer's holdings for buy or sell

```
        where
            H_CA_ID = acct_id and
            H_S_SYMB = symbol
        order by
            H_DTS desc
    } else {
        // Could return 0, 1 or many rows
        declare hold_list cursor for
            select
                H_T_ID,
                H_QTY,
                H_PRICE
            from
                HOLDING
            where
                H_CA_ID = acct_id and
                H_S_SYMB = symbol
            order by
                H_DTS asc
    }
    // Buy back securities to cover a short position.
    open hold_list
    do until (needed_qty = 0 or end_of_hold_list) {
        fetch from
            hold_list
        into
            hold_id,
            hold_qty,
            hold_price
        if (hold_qty + needed_qty < 0) then {
            // Buying back some of the Short Sell
            insert into
                HOLDING_HISTORY (
                    HH_H_T_ID,
                    HH_T_ID,
                    HH_BEFORE_QTY,
                    HH_AFTER_QTY
                )
            values (
                hold_id,                // H_T_ID original trade
                trade_id,               // T_ID current trade
                hold_qty,               // H_QTY now
                hold_qty + needed_qty  // H_QTY after update
            )

            update
                HOLDING
            set
```

Trade-Result_Frame-2 pseudo-code: Update the customer's holdings for buy or sell

```
H_QTY = hold_qty + needed_qty
where
    current of hold_list

sell_value += needed_qty * hold_price
buy_value += needed_qty * trade_price
needed_qty = 0
} else {
    // Buying back all of the Short Sell
    insert into
        HOLDING_HISTORY (
            HH_H_T_ID,
            HH_T_ID,
            HH_BEFORE_QTY,
            HH_AFTER_QTY
        )
    values (
        hold_id,      // H_T_ID original trade
        trade_id,     // T_ID current trade
        hold_qty,     // H_QTY now
        0             // H_QTY after delete
    )

    delete from
        HOLDING
    where
        current of hold_list

    // Make hold_qty positive for easy calculations
    hold_qty = -hold_qty
    sell_value += hold_qty * hold_price
    buy_value += hold_qty * trade_price
    needed_qty = needed_qty - hold_qty
}
}
close hold_list
}

// Buy Trade:
// If needed_qty > 0, then the customer has covered all
// previous Short Sells and the customer is buying new
// holdings. A new HOLDING record will be created with
// H_QTY set to the number of needed shares.
if (needed_qty > 0) then {
    insert into
        HOLDING_HISTORY (
            HH_H_T_ID,
            HH_T_ID,
            HH_BEFORE_QTY,
```

Trade-Result_Frame-2 pseudo-code: Update the customer's holdings for buy or sell

```
        HH_AFTER_QTY
    )
values (
    trade_id,      // T_ID current is original trade
    trade_id,      /* T_ID current trade
    0,             // H_QTY before
    needed_qty     // H_QTY after insert
)

insert into
    HOLDING (
        H_T_ID,
        H_CA_ID,
        H_S_SYMB,
        H_DTS,
        H_PRICE,
        H_QTY
    )
values (
    trade_id      // H_T_ID
    acct_id,      // H_CA_ID
    symbol,       // H_S_SYMB
    trade_dts,    // H_DTS
    trade_price,  // H_PRICE
    needed_qty    // H_QTY
)
}
else
if (-hs_qty = trade_qty) then
    delete from
        HOLDING_SUMMARY
    where
        HS_CA_ID  = acct_id and
        HS_S_SYMB = symbol
}
}
```

3.3.2.5 Trade-Result Transaction Frame 3 of 6

The third **Frame** is responsible for computing the amount of tax due by the customer as a result of the trade. **Frame 3** is only executed if the customer is liquidating existing holdings, and the liquidation has resulted in a gain, and the customer's tax status is either 1 or 2. The amount of tax due is recorded in the TRADE table.

Comment: The parameter tax_amount is used by the **EGenTxnHarness** to compute the value of the parameter se_amount just before **Frame 6**. Thus, the parameter tax_amount is initialized to zero and is passed in and out of **Frame 3**.

The database access methods used in **Frame 3** are a mixture of **References** and **Modifies**.

The EGenTxnHarness controls the execution of **Frame 3** as follows:

```
{
    tax_amount = 0.0
    if ((tax_status == 1 or tax_status == 2)
        and (sell_value > buy_value)) then
    {
        invoke (Trade-Result_Frame-3)
    }
}
```

Trade-Result Frame 3 of 6 Parameters:

Field	Direction	Description
buy_value	IN	The total dollar amount for the securities bought for a matching sell order.
cust_id	IN	Customer ID of the customer involved in the Trade-Result transaction, which was obtained in Frame 1.
sell_value	IN	The total dollar value of the securities sold for a matching buy order.
trade_id	IN	The Trade ID for the trade to be settled passed to the transaction by the Market-Exchange-Emulator.
status	OUT	Code indicating the execution status for this frame.
tax_amount	OUT	Tax_amount is initialized to 0.0 by the EGen code and modified by Frame 3.

Trade-Result_Frame-3 pseudo-code: Compute and record the tax liability

```
{
    // Local Frame variables
    Declare tax_rates    S_PRICE_T
    select
        tax_rates = sum(TX_RATE)
    from
        TAXRATE
    where
        TX_ID in ( select
                    CX_TX_ID
                  from
                    CUSTOMER_TAXRATE
                  where
                    CX_C_ID = cust_id)

    tax_amount = (sell_value - buy_value) * tax_rates

    update
        TRADE
    set
        T_TAX = tax_amount
    where
```

Trade-Result_Frame-3 pseudo-code: Compute and record the tax liability

```
T_ID = trade_id  
  
}
```

3.3.2.6 Trade-Result Transaction Frame 4 of 6

The fourth **Frame** is responsible for computing the commission for the broker who executed the trade.

The database access methods used in **Frame 4** are all **References**.

The **EGenTxnHarness** controls the execution of **Frame 4** as follows:

```
{  
    invoke (Trade-Result_Frame-4)  
}
```

Trade-Result Frame 4 of 6 Parameters:

Field	Direction	Description
cust_id	IN	Customer ID of the customer involved in the Trade-Result transaction, which was obtained in Frame 1.
symbol	IN	Seven character security identifier, which was obtained in Frame 1
trade_qty	IN	Quantity of securities traded, which was obtained in Frame 1
type_id	IN	Trade type identifier, which was obtained in Frame 1
comm_rate	OUT	The broker commission rate. Ranges from 0.00 to 100.00.
s_name	OUT	Name of security traded
status	OUT	Code indicating the execution status for this frame.

Trade-Result_Frame-4: Compute and record the broker's commission

```
{  
    select  
        s_ex_id = S_EX_ID,  
        s_name  = S_NAME  
    from  
        SECURITY  
    where  
        S_SYMB = symbol  
  
    select  
        c_tier = C_TIER  
    from  
        CUSTOMER
```

Trade-Result_Frame-4: Compute and record the broker's commission

```
where
    C_ID = cust_id

// Only want 1 commission rate row
select first 1 row
    comm_rate = CR_RATE
from
    COMMISSION_RATE
where
    CR_C_TIER = c_tier and
    CR_TT_ID = type_id and
    CR_EX_ID = s_ex_id and
    CR_FROM_QTY <= trade_qty and
    CR_TO_QTY >= trade_qty
}
```

3.3.2.7 Trade-Result Transaction Frame 5 of 6

The fifth **Frame** is responsible for recording the result of the trade and the broker's commission.

The database access methods used in **Frame 5** are a mixture of **Modifies**, **Adds** and **Removes**.

The **EGenTxnHarness** controls the execution of **Frame 5** as follows:

```
{
    comm_amount = (comm_rate / 100) * (trade_qty * trade_price)
    invoke (Trade-Result_Frame-5)
}
```

Trade-Result Frame 5 of 6 Parameters:

Field	Direction	Description
broker_id	IN	Broker ID, which was obtained in Frame 1.
comm_amount	IN	The broker commission amount, computed by the EGen code
st_completed_id	IN	The index ID value into STATUS_TYPE for “Completed” status.
trade_dts	IN	Trade date and time provided by the output of Frame 2.
trade_id	IN	The Trade ID for the trade to be settled passed to the transaction by the Market-Exchange-Emulator.
trade_price	IN	Trade price provided by the Market-Exchange-Emulator.
status	OUT	Code indicating the execution status for this frame.

Trade-Result_Frame-5 pseudo-code: Record the trade result and the broker's commission

Trade-Result_Frame-5 pseudo-code: Record the trade result and the broker's commission

```
{
  update
    TRADE
  set
    T_COMM      = comm_amount,
    T_DTS       = trade_dts,
    T_ST_ID     = st_completed_id,
    T_TRADE_PRICE = trade_price
  where
    T_ID = trade_id

  insert into
    TRADE_HISTORY (
      TH_T_ID,
      TH_DTS,
      TH_ST_ID
    )
  values (
    trade_id,
    trade_dts,
    st_completed_id
  )

  update
    BROKER
  set
    B_COMM_TOTAL = B_COMM_TOTAL + comm_amount,
    B_NUM_TRADES = B_NUM_TRADES + 1
  where
    B_ID = broker_id
}
```

3.3.2.8 Trade-Result Transaction Frame 6 of 6

The sixth **Frame** is responsible for settling the trade.

The database access methods used in **Frame 6** are a mixture **Adds** and **Modifies**.

The **EGenTxnHarness** controls the execution of **Frame 6** as follows:

```

{
    due_date = (trade_date + 2 days)
    if (type_is_sell) then
    {
        se_amount = (trade_qty * trade_price) - charge - comm_amount
    } else {
        se_amount = -((trade_qty * trade_price) + charge + comm_amount)
    }
    if (tax_status == 1) then
    {
        se_amount = se_amount - tax_amount
    }
    invoke (Trade-Result_Frame-6)
}

```

Trade-Result Frame 6 of 6 Parameters:

Field	Direction	Description
acct_id	IN	Customer account ID of the customer involved in the Trade-Result transaction, which was obtained in Frame 1.
due_date	IN	Date and time when trade is due to be settled.
s_name	IN	Name of security traded, which was obtained in Frame 4
se_amount	IN	The trade settlement amount.
trade_dts	IN	Date and time of trade result generated by the SUT, and output in Frame 2.
trade_id	IN	The trade ID for the trade to be settled, passed to the transaction by the Market-Exchange-Emulator.
trade_is_cash	IN	Boolean obtained in Frame 1 indicating trade is for cash (TRUE) or on margin (FALSE).
trade_qty	IN	Quantity of securities traded, which was obtained from Frame 1
type_name	IN	Trade type name, which was obtained in Frame 1.
acct_bal	OUT	Customer account's cash balance (needed for one of the isolation tests)
status	OUT	Code indicating the execution status for this frame.

Trade-Result_Frame-6 pseudo-code: Settle the trade

```

{
    // Local Frame Variables
    Declare cash_type char(40)
    if (trade_is_cash) then
        cash_type = "Cash Account"
    else
        cash_type = "Margin"

```

Trade-Result_Frame-6 pseudo-code: Settle the trade

```
insert into
    SETTLEMENT (
        SE_T_ID,
        SE_CASH_TYPE,
        SE_CASH_DUE_DATE,
        SE_AMT
    )
values (
    trade_id,
    cash_type,
    due_date,
    se_amount
)

if (trade_is_cash) then {
    update
        CUSTOMER_ACCOUNT
    set
        CA_BAL = CA_BAL + se_amount
    where
        CA_ID = acct_id

    insert into
        CASH_TRANSACTION (
            CT_DTS,
            CT_T_ID,
            CT_AMT,
            CT_NAME
        )
    values (
        trade_dts,
        trade_id,
        se_amount,
        type_name + " " + trade_qty + " shares of " + s_name
    )
}

select
    acct_bal = CA_BAL
from
    CUSTOMER_ACCOUNT
where
    CA_ID = acct_id

commit transaction
}
```

3.3.3 The Trade-Lookup Transaction

The Trade-Lookup **transaction** is designed to emulate the information retrieval by either a customer or a broker to satisfy their questions regarding a particular account, a group of trade transaction identifiers or a particular security. This is analogous to a customer reviewing their transactions for a period of time beyond the most recent account statement, a broker gathering information prior to analyzing past performance of individual securities, customers, brokers, and overall market analysis. This **transaction** primarily looks at historical data and retrieves all pertinent information regarding trades for a customer's account, trade transaction identifiers, or a security.

The Trade-Lookup **transaction** is invoked by the **CE** component of the **Driver**. The **transaction** has 4 mutually exclusive **frames**. Each of the **frames** employs a different technique for identifying a particular set of trades and then looking up information for those particular trades.

The first **Frame** accepts an array of trade IDs and an array length as inputs. Information for the identified trades is returned from the TRADE, TRADE_HISTORY and SETTLEMENT tables, as well as from the CASH_TRANSACTION table for cash (non-margin) trades.

The second **Frame** accepts a customer account ID, a start timestamp, end timestamp and a number of trades (N) as inputs. It finds the first N trades by the specified customer account at or after the specified start timestamp but at or before the specified end timestamp. Information from the TRADE, TRADE_HISTORY and SETTLEMENT tables is returned, as well as from the CASH_TRANSACTION table for cash (non-margin) trades.

The third **Frame** accepts a security symbol, a start timestamp, end timestamp and a number of trades (N) as inputs. It then finds the first N trades for the given security starting from the specified point in time and ending at the specified end timestamp. Information from the TRADE, TRADE_HISTORY and SETTLEMENT tables is returned, as well as from the CASH_TRANSACTION table for cash (non-margin) trades.

The fourth **Frame** accepts a customer account ID and a timestamp as inputs. The first trade for this customer account on or after the specified timestamp is identified. Then a maximum of 20 entries in the HOLDING_HISTORY table for this trade ID are returned.

3.3.3.1 Trade-Lookup Transaction Parameters

The inputs to the Trade-Lookup **transaction** are generated by the **EGen CE** code in *CETxnInputGenerator.cpp*. The data structures used to communicate the input and output parameters must match the **EGenTxnHarness** data structures defined in *TxnHarnessStructs.h*.

Trade-Lookup Interfaces	Module/Data Structure
CE Input generation	GenerateTradeLookupInput()
Transaction Input/Output Structure	TTradeLookupTxnInput TTradeLookupTxnOutput
Frame 1 Input/Output Structure	TTradeLookupFrame1Input TTradeLookupFrame1Output
Frame 2 Input/Output Structure	TTradeLookupFrame2Input TTradeLookupFrame2Output
Frame 3 Input/Output Structure	TTradeLookupFrame3Input TTradeLookupFrame3Output
Frame 4 Input/Output Structure	TTradeLookupFrame4Input TTradeLookupFrame4Output

Trade-Lookup Transaction Parameters:

Field	Direction	
acct_id	IN	Customer account ID. Used when frame_to_execute is 2 or 4, otherwise set to 0.
end_trade_dts	IN	For Frames 1 and 4, this parameter is ignored, so it can be set to an empty date. Used in Frame 2 as the end point in time for identifying a particular trade. Used in Frame 3 as the end point in time for identifying trades for a particular symbol.
frame_to_execute	IN	Identifies which of the mutually exclusive frames to execute.
max_acct_id	IN	Used in Frame 3 to identify the maximum customer account ID, otherwise set to 0.
max_trades	IN	Used in Frames 1, 2 and 3 for the number of trades to find otherwise set to 0. The default value for max_trades for each frame is set in the TTradeLookupSettings structure in DriverParameterSettings.h
start_trade_dts	IN	For Frame 1, this parameter is ignored, so it can be set to an empty date. Used in Frame 2 as the point in time for identifying a particular trade. Non-uniform over pre-populated interval. Used in Frame 3 as the point in time for identifying trades for a particular symbol. Uniform over pre-populated interval. Used in Frame 4 as the point in time for identifying a particular trade. Uniform over pre-populated interval.
symbol	IN	Used in Frame 3 as the security symbol for which to find trades. Uniformly chosen over all securities. For the other frames symbol is set to the empty string.
trade_id[]	IN	Array of non-uniform randomly chosen trade IDs used by Frame 1 to identify a set of particular trades. For the other frames array elements are set to 0. For Frame 1, max_trades indicates how many elements are to be used in the array.
frame_executed	OUT	Confirmation of which frame was executed.
is_cash[]	OUT	Indicates whether the trades used in Frame 1, 2 or 3 were cash transactions.
is_market[]	OUT	Indicates whether the trades used in Frame 1 were market order trades.
num_found	OUT	Number of trade rows found for frames 1, 2, 3, or number of holding history rows found for frame 4.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.
trade_list[]	OUT	List of trade IDs found in Frames 2 and 3.

3.3.3.2 Trade-Lookup Transaction Database-Footprint

The Trade-Lookup **Database-Footprint** is as follows:

Trade-Lookup Database-Footprint					
Table	Column	Frame			
		1*	2*	3*	4*
CASH_TRANSACTION	CT_AMT	Return*	Return*	Return*	
	CT_DTS	Return*	Return*	Return*	
	CT_NAME	Return*	Return*	Return*	
HOLDING_HISTORY	Row(s)				Return
SETTLEMENT	SE_AMT	Return	Return	Return	
	SE_CASH_DUE_DATE	Return	Return	Return	
	SE_CASH_TYPE	Return	Return	Return	
TRADE	T_BID_PRICE	Return	Return		
	T_CA_ID			Return	
	T_DTS		Reference	Reference	Reference
	T_EXEC_NAME	Return	Return	Return	
	T_ID		Return	Return	Return
	T_IS_CASH	Return	Return	Return	
	T_QTY			Return	
	T_S_SYMB			Reference	
	T_ST_ID		Reference	Reference	
	T_TRADE_PRICE	Return	Return	Return	
	T_TT_ID			Return	
TRADE_HISTORY	TH_DTS	Return	Return	Return	
	TH_ST_ID	Return	Return	Return	
TRADE_TYPE	TT_IS_MRKT	Return			
Transaction Control	Start Commit	Start Commit	Start Commit	Start Commit	Start Commit

3.3.3.3 Trade-Lookup Transaction Frame 1 of 4

The first **Frame** is responsible for retrieving information about the specified array of trade IDs.

The EGenTxnHarness controls the execution of **Frame 1** as follows:

```

{
    if( frame_to_execute == 1 )
    {
        invoke (Trade-Lookup_Frame-1)
        frame_executed = 1
    }
    [...]

```

Trade-Lookup Frame 1 of 4 Parameters:

Field	Direction	Description
max_trades	IN	Number of valid array elements in trade_id[]. The default value (20) is set in TTradeLookupSettings.MaxRowsFrame1 in DriverParameterSettings.h.
trade_id[]	IN	The array of trade IDs picked non-uniformly over the set of pre-populated trades.
bid_price[]	OUT	The requested unit price.
cash_transaction_amount[]	OUT	Amount of the cash transaction.
cash_transaction_dts[]	OUT	Date and time stamp of when the transaction took place.
cash_transaction_name[]	OUT	Description of the cash transaction.
exec_name[]	OUT	Name of the person who executed the trade.
is_cash[]	OUT	Flag that is non-zero for a cash trade, zero for a margin trade.
is_market[]	OUT	Flag that is non-zero for a market trade, zero for a limit trade.
num_found	OUT	Number of trade rows returned, usually the same as max_trades.
settlement_amount[]	OUT	Cash amount of settlement.
settlement_cash_due_date[]	OUT	Date by which customer or brokerage must receive the cash.
settlement_cash_type[]	OUT	Type of cash settlement involved: cash or margin.
status	OUT	Code indicating the execution status for this frame.
trade_history_dts[][3]	OUT	Array of timestamps of when the trade history was updated.
trade_history_status_id[][3]	OUT	Array of status type identifiers.
trade_price[]	OUT	Unit price at which the security was traded.

Trade-Lookup_Frame-1 pseudo-code: Get trade information for each trade ID in the trade_id array

```

{
    declare i int
    start transaction

    num_found = max_trades

    for (i = 0; i++; i < max_trades) do {
        // Get trade information
        // Should only return one row for each trade
        select
            bid_price[i]   = T_BID_PRICE,
            exec_name[i]   = T_EXEC_NAME,
            is_cash[i]     = T_IS_CASH,
            is_market[i]   = TT_IS_MRKT,
            trade_price[i] = T_TRADE_PRICE
        from

```

Trade-Lookup_Frame-1 pseudo-code: Get trade information for each trade ID in the trade_id array

```
TRADE,
TRADE_TYPE
where
  T_ID = trade_id[i] and
  T_TT_ID = TT_ID

// Get settlement information
// Should only return one row for each trade
select
  settlement_amount[i]      = SE_AMT,
  settlement_cash_due_date[i] = SE_CASH_DUE_DATE,
  settlement_cash_type[i]    = SE_CASH_TYPE
from
  SETTLEMENT
where
  SE_T_ID = trade_id[i]

// get cash information if this is a cash transaction
// Should only return one row for each trade that was a cash transaction
if (is_cash[i]) then {
  select
    cash_transaction_amount[i] = CT_AMT,
    cash_transaction_dts[i]    = CT_DTS,
    cash_transaction_name[i]   = CT_NAME
  from
    CASH_TRANSACTION
  where
    CT_T_ID = trade_id[i]
}

// read trade_history for the trades
// Should return 2 to 3 rows per trade
select
  trade_history_dts[i][]      = TH_DTS,
  trade_history_status_id[i][] = TH_ST_ID
from
  TRADE_HISTORY
where
  TH_T_ID = trade_id[i]
order by
  TH_DTS
} // end for loop

commit transaction
}
```

3.3.3.4 Trade-Lookup Transaction Frame 2 of 4

The second **Frame** returns information for the first N (max_trades) trades executed for the specified customer account at or after the specified start time but before or at a specified end time. If the specified start time (start_trade_dts) is too close to the end of the historical trading data, it is possible that fewer than N trades may be present.

The **EGenTxnHarness** controls the execution of **Frame 2** as follows:

```
[...]
    else if( frame_to_execute == 2 )
    {
        invoke (Trade-Lookup_Frame-2)
        frame_executed = 2
    }
[...]
```

Trade-Lookup Frame 2 of 4 Parameters:

Field	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. A single customer account id, as defined by CA_ID in CUSTOMER_ACCOUNT, is chosen at random, uniformly, from the range of customer account id's for the chosen customer.
end_trade_dts	IN	Point in time at which to stop searching for N trades.
max_trades	IN	Maximum number of trades to return. The default value (20) is set in TTradeLookupSettings.MaxRowsFrame2 in DriverParameterSettings.h.
start_trade_dts	IN	Point in time from which to search for N trades.
bid_price[]	OUT	The requested unit price.
cash_transaction_amount[]	OUT	Amount of the cash transaction.
cash_transaction_dts[]	OUT	Date and time stamp of when the transaction took place.
cash_transaction_name[]	OUT	Description of the cash transaction.
exec_name[]	OUT	Name of the person who executed the trade.
is_cash[]	OUT	Flag that is non-zero for a cash trade, zero for a margin trade.
num_found	OUT	Number of trade rows returned (may be less than max_trades).
settlement_amount[]	OUT	Cash amount of settlement.
settlement_cash_due_date[]	OUT	Date by which customer or brokerage must receive the cash.
settlement_cash_type[]	OUT	Type of cash settlement involved: cash or margin.
status	OUT	Code indicating the execution status for this frame.
trade_history_dts[][3]	OUT	Array of timestamps of when the trade history was updated.
trade_history_status_id[][3]	OUT	Array of status type identifiers.
trade_list[]	OUT	Trade ID actually used for retrieving data.
trade_price[]	OUT	Unit price at which the security was traded.

Trade-Lookup_Frame-2 pseudo-code : Get trade information for the first N trades of a given customer account from a given point in time.

```
{
    declare i int
    start transaction

    // Get trade information
    // Should return between 0 and max_trades rows
    select first max_trades rows
        bid_price[]    = T_BID_PRICE,
        exec_name[]    = T_EXEC_NAME,
        is_cash[]      = T_IS_CASH,
        trade_list[]   = T_ID,
        trade_price[]  = T_TRADE_PRICE
    from
        TRADE
    where
        T_CA_ID = acct_id and
        T_ST_ID = "CMPT" and
        T_DTS >= start_trade_dts and
        T_DTS <= end_trade_dts
    order by
        T_DTS asc

    num_found    = row_count

    // Get extra information for each trade in the trade list.
    for (i = 0; i < num_found; i++) {
        // Get settlement information
        // Should return only one row for each trade
        select
            settlement_amount[i]    = SE_AMT,
            settlement_cash_due_date[i] = SE_CASH_DUE_DATE,
            settlement_cash_type[i]   = SE_CASH_TYPE
        from
            SETTLEMENT
        where
            SE_T_ID = trade_list[i]

        // get cash information if this is a cash transaction
        // Should return only one row for each trade that was a cash transaction
        if (is_cash[i]) then {
            select
                cash_transaction_amount[i] = CT_AMT,
                cash_transaction_dts[i]    = CT_DTS
                cash_transaction_name[i]    = CT_NAME
            from
                CASH_TRANSACTION
```

Trade-Lookup_Frame-2 pseudo-code : Get trade information for the first N trades of a given customer account from a given point in time.

```

        where
            CT_T_ID = trade_list[i]
        }

        // read trade_history for the trades
        // Should return 2 to 3 rows per trade
        select
            trade_history_dts[i][]      = TH_DTS,
            trade_history_status_id[i][] = TH_ST_ID
        from
            TRADE_HISTORY
        where
            TH_T_ID = trade_list[i]
        order by
            TH_DTS

    } // end for loop

    commit transaction
}

```

3.3.3.5 Trade-Lookup Transaction Frame 3 of 4

The third **Frame** returns up to N (max_trades) trades for a given security on or after a specified start point in time but at or before a specified end time. If the specified time is too close to the end of the historical trade data, it is possible that fewer than N trades may be present for the given security.

The **EGenTxnHarness** controls the execution of **Frame 3** as follows:

```

[... ]
    else if( frame_to_execute == 3 )
    {
        invoke (Trade-Lookup_Frame-3)
        frame_executed = 3
    }
}

```

Trade-Lookup Frame 3 of 4 Parameters:

Field	Direction	Description
end_trade_dts	IN	Point in time at which to end the search.
max_acct_id	IN	Maximum customer account ID.
max_trades	IN	Maximum number of trades to find. The default value (20) is set in TTradeLookupSettings.MaxRowsFrame3 in DriverParameterSettings.h.
start_trade_dts	IN	Point in time from which to start search.

symbol	IN	Security for which to find trades.
acct_id[]	OUT	Array of accounts for which the trades were done.
cash_transaction_amount[]	OUT	Amount of the cash transaction.
cash_transaction_dts[]	OUT	Date and time stamp of when the transaction took place.
cash_transaction_name[]	OUT	Description of the cash transaction.
exec_name[]	OUT	Array of name of the person who executed each of the trades.
is_cash[]	OUT	Flag that is non-zero for a cash trade, zero for a margin trade.
num_found	OUT	Number of TRADE rows returned.
price[]	OUT	Array of the price that was paid in each trade.
quantity[]	OUT	Array of the quantity of security bought in each trade.
settlement_amount[]	OUT	Cash amount of settlement.
settlement_cash_due_date[]	OUT	Date by which the customer or brokerage must receive the cash.
settlement_cash_type[]	OUT	Type of cash settlement involved: cash or margin.
status	OUT	Code indicating the execution status for this frame.
trade_dts[]	OUT	Array of the timestamps for when the trade was requested.
trade_history_dts[][3]	OUT	Array of timestamps of when the trade history was updated.
trade_history_status_id[][3]	OUT	Array of status type identifiers.
trade_list[]	OUT	Array of T_IDs found.
trade_type[]	OUT	Array of the trade type for each trade.

Trade-Lookup_Frame-3 pseudo-code: Get a list of N trades executed for a certain security starting from a given point in time.

```

{
  declare i int
  start transaction

  // Should return between 0 and max_trades rows.
  select first max_trades rows
    acct_id[]    = T_CA_ID,
    exec_name[]  = T_EXEC_NAME,
    is_cash[]    = T_IS_CASH,
    price[]      = T_TRADE_PRICE,
    quantity[]   = T_QTY,
    trade_dts[]  = T_DTS,
    trade_list[] = T_ID,
    trade_type[] = T_TT_ID
  from
    TRADE
  where
    T_S_SYMB = symbol and
    T_ST_ID  = "CMPT" and
    T_DTS   >= start_trade_dts and

```

Trade-Lookup_Frame-3 pseudo-code: Get a list of N trades executed for a certain security starting from a given point in time.

```
T_DTS <= end_trade_dts
// The max_acct_id "where" clause is a hook used for engineering purposes
// only and is not required for benchmark publication purposes.
// T_CA_ID <= max_acct_id
order by
    T_DTS asc

num_found = row_count

// Get extra information for each trade in the trade list.
for (i = 0; i < num_found; i++) {
    // Get settlement information
    // Should return only one row for each trade
    select
        settlement_amount[i]          = SE_AMT,
        settlement_cash_due_date[i]    = SE_CASH_DUE_DATE,
        settlement_cash_type[i]        = SE_CASH_TYPE
    from
        SETTLEMENT
    where
        SE_T_ID = trade_list[i]

    // get cash information if this is a cash transaction
    // Should return only one row for each trade that was a cash transaction
    if (is_cash[i]) then {
        select
            cash_transaction_amount[i] = CT_AMT,
            cash_transaction_dts[i]    = CT_DTS
            cash_transaction_name[i]   = CT_NAME
        from
            CASH_TRANSACTION
        where
            CT_T_ID = trade_list[i]
    }

    // read trade_history for the trades
    // Should return 2 to 3 rows per trade
    select
        trade_history_dts[i][]         = TH_DTS,
        trade_history_status_id[i][]   = TH_ST_ID
    from
        TRADE_HISTORY
    where
        TH_T_ID = trade_list[i]
    order by
        TH_DTS asc

} // end for loop

commit transaction
```

Trade-Lookup_Frame-3 pseudo-code: Get a list of N trades executed for a certain security starting from a given point in time.

}

3.3.3.6 Trade-Lookup Transaction Frame 4 of 4

The fourth **Frame** identifies the first trade for the specified customer account on or after the specified time. Up to the first 20 rows in the HOLDING_HISTORY with a matching trade ID are then returned. If the specified time is too close to the end of the historical trade data, it is possible that no matching trade may be found for the specified customer account.

The EGenTxnHarness controls the execution of **Frame 4** as follows:

```
[...]
    else if( frame_to_execute == 4 )
    {
        invoke (Trade-Lookup_Frame-4)
        frame_executed = 4
    }
[...]
```

Trade-Lookup Frame 4 of 4 Parameters:

Field	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. A single customer account id, as defined by CA_ID in CUSTOMER_ACCOUNT, is chosen at random, uniformly, from the range of customer account id's for the chosen customer.
start_trade_dts	IN	Point in time from which to search for a trade.
holding_history_id[20]	OUT	Array of trade identifiers of the trades that originally created each of the returned holding rows.
holding_history_trade_id[20]	OUT	Array of trade identifiers of the trades that modified each of the returned holding rows.
num_found	OUT	Number of HOLDING_HISTORY rows returned.
quantity_after[20]	OUT	Array of quantities of the security that was held after the holding was modified.
quantity_before[20]	OUT	Array of quantities of the security that was held before the holding was modified.
status	OUT	Code indicating the execution status for this frame.
trade_id	OUT	ID of first trade found for customer at or after the specified time. This is the ID that is used for the look up in HOLDING_HISTORY.

Trade-Lookup_Frame-4 pseudo-code: Return HOLDING_HISTORY information for a particular trade ID.

Trade-Lookup_Frame-4 pseudo-code: Return HOLDING_HISTORY information for a particular trade ID.

```
{
    start transaction

    select first 1 row
        trade_id = T_ID
    from
        TRADE
    where
        T_CA_ID = acct_id and
        T_DTS >= start_trade_dts
    order by
        T_DTS asc

    // The trade_id is used in the subquery to find the original trade_id
    // (HH_H_T_ID), which then is used to list all the entries.

    // Should return 0 to 20 rows.
    select first 20 rows
        holding_history_id[]      = HH_H_T_ID,
        holding_history_trade_id[] = HH_T_ID,
        quantity_before[]         = HH_BEFORE_QTY,
        quantity_after[]          = HH_AFTER_QTY
    from
        HOLDING_HISTORY
    where
        HH_H_T_ID in
            (select
                HH_H_T_ID
            from
                HOLDING_HISTORY
            where
                HH_T_ID = trade_id)

    num_found = row_count

    commit transaction
}
```

3.3.4 The Trade-Update Transaction

The Trade-Update **transaction** is designed to emulate information retrieval and possibly modification by either a customer or a broker to satisfy questions regarding a particular account, a group of trade transaction identifiers or a particular security. This is analogous to a customer reviewing transactions for a period of time beyond the most recent account statement, or a broker gathering information prior to analyzing past performance of individual securities, customers, brokers, or the overall markets. This **transaction** primarily looks at historical data and retrieves all pertinent information regarding trades for a customer's account, trade transaction identifiers, or a security. Minor corrections may also be made to some of the retrieved data.

The Trade-Update **transaction** is invoked by the **CE** component of the **Driver**. The **transaction** has 3 mutually exclusive **frames**. Each of the **frames** employs a different technique for identifying a particular set of trades, looking up information from several tables associated with those particular trades, and modifying data in one of those tables.

The first **Frame** accepts an array of trade IDs and an array length as inputs. Information for the identified trades is returned from the TRADE, TRADE_HISTORY and SETTLEMENT tables, as well as possibly from the CASH_TRANSACTION table for cash (non-margin) trades. Some rows from the TRADE table may also be modified.

The second **Frame** accepts a customer account ID, a start timestamp, end timestamp and a number of trades (N) as inputs. It finds the first N trades by the specified customer account at or after the specified start timestamp but before the specified end timestamp. Information from the TRADE, TRADE_HISTORY and SETTLEMENT tables is returned, as well as from the CASH_TRANSACTION table for cash (non-margin) trades. Some rows from the SETTLEMENT table may also be modified.

The third **Frame** accepts a security symbol, a start timestamp, end timestamp and a number of trades (N) as inputs. It then finds the first N trades for the given security starting from the specified start point in time and ending at the specified end timestamp. Information from the TRADE, TRADE_HISTORY and SETTLEMENT tables is returned, as well as from the CASH_TRANSACTION table for cash (non-margin) trades. Some rows from the CASH_TRANSACTION table may also be modified.

3.3.4.1 Trade-Update Transaction Parameters

The inputs to the Trade-Update **transaction** are generated by the **EGen CE** code in *CETxnInputGenerator.cpp*. The data structures used to communicate the input and output parameters must match the **EGenTxnHarness** data structures defined in *TxnHarnessStructs.h*.

Trade-Update Interfaces	Module/Data Structure
CE Input generation	GenerateTradeUpdateInput()
Transaction Input/Output Structure	TTradeUpdateTxnInput TTradeUpdateTxnOutput
Frame 1 Input/Output Structure	TTradeUpdateFrame1Input TTradeUpdateFrame1Output
Frame 2 Input/Output Structure	TTradeUpdateFrame2Input TTradeUpdateFrame2Output
Frame 3 Input/Output Structure	TTradeUpdateFrame3Input TTradeUpdateFrame3Output

Trade-Update Transaction Parameters:

Field	Direction	Description
acct_id	IN	Customer account ID. Used when frame_to_execute is 2, otherwise set to 0.
end_trade_dts	IN	Used in Frame 2 as the end point in time for identifying a particular trade for an account. Used in Frame 3 as the end point in time for identifying trades for a particular symbol. For Frame 1, this parameter is ignored, so it is set to an empty date.
frame_to_execute	IN	Identifies which of the mutually exclusive frames to execute.
max_acct_id	IN	Maximum account identifier, used in Frame 3, otherwise set to 0.
max_trades	IN	Maximum number of trades to find. The default value (20) is defined in the TTradeUpdateSettings structure in DriverParameterSettings.h.
max_updates	IN	Maximum number of trades to be modified. The default value (20) is defined in the TTradeUpdateSetting structure in DriverParameterSettings.h.
start_trade_dts	IN	Used in Frame 2 as the point in time for identifying a particular trade for an account. Non-uniform over pre-populated interval. Used in Frame 3 as the point in time for identifying trades for a particular symbol. Uniform over pre-populated interval. For Frame 1, this parameter is ignored, so it is set to an empty date.
symbol	IN	Used in Frame 3 as the security symbol for which to find trades. Uniformly chosen over all securities. For the other frames, symbol is set to the empty string.
trade_id[]	IN	Array of non-uniform randomly chosen trade IDs used by Frame 1 to identify a set of particular trades. For the other frames, array elements are set to 0. For Frame 1, max_trades indicates how many elements are to be used in the array.
frame_executed	OUT	Confirmation of which frame was executed.
is_cash[]	OUT	Indicates whether the trades were cash transactions.
is_market[]	OUT	Indicates whether the trades used in Frame 1 were market order trades.
num_found	OUT	Number of trade rows found for frames 1, 2 and 3.
num_updated	OUT	Number of trade rows modified for frames 1, 2 and 3.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.
trade_list[]	OUT	List of trade IDs found in Frames 2 and 3.

3.3.4.2 Trade-Update Transaction Database-Footprint

The Trade-Update **Database-Footprint** is as follows:

Trade-Update Database-Footprint				
Table	Column	Frame		
		1*	2*	3*
CASH_TRANSACTION	CT_AMT	Return*	Return*	Return*
	CT_DTS	Return*	Return*	Return*
	CT_NAME	Return*	Return*	Modify* Return*
SECURITY	S_NAME			Return

SETTLEMENT	SE_AMT	Return	Return	Return
	SE_CASH_DUE_DATE	Return	Return	Return
	SE_CASH_TYPE	Return	Modify Return	Return
TRADE	T_BID_PRICE	Return	Return	
	T_CA_ID			Return
	T_DTS		Reference	Reference
	T_EXEC_NAME	Modify Return	Return	Return
	T_ID		Return	Return
	T_IS_CASH	Return	Return	Return
	T_QTY			Return
	T_S_SYMB			Reference
	T_ST_ID		Reference	Reference
	T_TRADE_PRICE	Return	Return	Return
	T_TT_ID			Return
TRADE_HISTORY	TH_DTS	Return	Return	Return
	TH_ST_ID	Return	Return	Return
TRADE_TYPE	TT_IS_MRKT	Return		
	TT_NAME			Return
Transaction Control		Start Commit	Start Commit	Start Commit

3.3.4.3 Trade-Update Transaction Frame 1 of 3

The first **Frame** is responsible for retrieving information about the specified array of trade IDs and modifying some data from the TRADE table.

The EGenTxnHarness controls the execution of **Frame 1** as follows:

```
{
    if( frame_to_execute == 1 )
    {
        invoke (Trade-Update_Frame-1)
        frame_executed = 1
    }
    [...]
}
```

Trade-Update Frame 1 of 3 Parameters:

Field	Direction	Description
max_trades	IN	Number of valid array elements in trade_id[]. The default value (20) is set in TTradeUpdateSettings.MaxRowsFrame1 in DriverParameterSettings.h.
max_updates	IN	Maximum number of TRADE rows to modify. The default value (20)

		is set in TTradeUpdateSettings.MaxRowsToUpdateFrame1 in DriverParameterSettings.h.
trade_id[]	IN	The array of trade IDs picked non-uniformly over the set of pre-populated trades.
bid_price[]	OUT	The requested unit price.
cash_transaction_amount[]	OUT	Amount of the cash transaction.
cash_transaction_dts[]	OUT	Date and time stamp of when the transaction took place.
cash_transaction_name[]	OUT	Description of the cash transaction.
exec_name[]	OUT	Name of the person who executed the trade.
is_cash[]	OUT	Flag that is non-zero for a cash trade, zero for a margin trade.
is_market[]	OUT	Flag that is non-zero for a market trade, zero for a limit trade.
num_found	OUT	Number of trade rows returned, usually the same as max_trades.
num_updated	OUT	Number of TRADE rows that were modified.
settlement_amount[]	OUT	Cash amount of settlement.
settlement_cash_due_date[]	OUT	Date by which customer or brokerage must receive the cash.
settlement_cash_type[]	OUT	Type of cash settlement involved: cash or margin.
status	OUT	Code indicating the execution status for this frame.
trade_history_dts[][3]	OUT	Array of timestamps of when the trade history was updated.
trade_history_status_id[][3]	OUT	Array of status type identifiers.
trade_price[]	OUT	Unit price at which the security was traded.

Trade-Update_Frame-1 pseudo-code: Get trade information for each trade ID in the trade_id array and modify some of the TRADE rows.

```

{
    declare i int
    declare ex_name char(64)
    start transaction

    num_found = max_trades
    num_updated = 0

    for (i = 0; i++; i < max_trades) do {
        // Get trade information
        if (num_updated < max_updates) then {
            // Modify the TRADE row for this trade.

            select
                ex_name = T_EXEC_NAME
            from
                TRADE
            where
                T_ID = trade_id[i]
        }
    }
}

```

Trade-Update_Frame-1 pseudo-code: Get trade information for each trade ID in the trade_id array and modify some of the TRADE rows.

```
        if (ex_name like "% X %") then
            select ex_name = REPLACE (ex_name, " X ", " ")
        else
            select ex_name = REPLACE (ex_name, " ", " X ")

    update
        TRADE
    set
        T_EXEC_NAME = ex_name
    where
        T_ID = trade_id[i]

    num_updated = num_updated + row_count
}

// Will only return one row for each trade
select
    bid_price[i]    = T_BID_PRICE,
    exec_name[i]    = T_EXEC_NAME,
    is_cash[i]      = T_IS_CASH,
    is_market[i]    = TT_IS_MRKT,
    trade_price[i]  = T_TRADE_PRICE
from
    TRADE,
    TRADE_TYPE
where
    T_ID = trade_id[i] and
    T_TT_ID = TT_ID

// Get settlement information
// Will only return one row for each trade
select
    settlement_amount[i]      = SE_AMT,
    settlement_cash_due_date[i] = SE_CASH_DUE_DATE,
    settlement_cash_type[i]    = SE_CASH_TYPE
from
    SETTLEMENT
where
    SE_T_ID = trade_id[i]

// get cash information if this is a cash transaction
// Will only return one row for each trade that was a cash transaction
if (is_cash[i]) then {
    select
        cash_transaction_amount[i] = CT_AMT,
        cash_transaction_dts[i]    = CT_DTS,
        cash_transaction_name[i]   = CT_NAME
    from
        CASH_TRANSACTION
    where
```

Trade-Update_Frame-1 pseudo-code: Get trade information for each trade ID in the trade_id array and modify some of the TRADE rows.

```

        CT_T_ID = trade_id[i]
    }
    // read trade_history for the trades
    // Will return 2 or 3 rows per trade
    select
        trade_history_dts[i][]      = TH_DTS,
        trade_history_status_id[i][] = TH_ST_ID
    from
        TRADE_HISTORY
    where
        TH_T_ID = trade_id[i]
    order by
        TH_DTS
    } // end for loop

    commit transaction
}

```

3.3.4.4 Trade-Update Transaction Frame 2 of 3

The second **Frame** returns information for up to the first N trades executed for the specified customer account at or after the specified start time but before or at the specified end time and modifies some rows of the SETTLEMENT table. If the specified start time is too close to the end of the historical trade data, then less than N trades may be present for the specified customer account, in which case, fewer rows may be modified.

The EGenTxnHarness controls the execution of **Frame 2** as follows:

```

[... ]
    else if( frame_to_execute == 2 )
    {
        invoke (Trade-Update_Frame-2)
        frame_executed = 2
    }
[... ]

```

Trade-Update Frame 2 of 3 Parameters:

Field	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. A single customer account id, as defined by CA_ID in CUSTOMER_ACCOUNT, is chosen at random, uniformly, from the range of customer account id's for the chosen customer.
end_trade_dts	IN	Point in time at which to stop the search for N trades.
max_trades	IN	Maximum number of trades to return. The default value (20) is set in TTradeUpdateSettings.MaxRowsFrame2 in DriverParameterSettings.h.

max_updates	IN	Maximum number of SETTLEMENT rows to modify. The default value (20) is set in TTradeUpdateSettings.MaxRowsToUpdateFrame2 in DriverParameterSettings.h.
start_trade_dts	IN	Point in time from which to search for N trades.
bid_price[]	OUT	The requested unit price.
cash_transaction_amount[]	OUT	Amount of the cash transaction.
cash_transaction_dts[]	OUT	Date and time stamp of when the transaction took place.
cash_transaction_name[]	OUT	Description of the cash transaction.
exec_name[]	OUT	Name of the person who executed the trade.
is_cash[]	OUT	Flag that is non-zero for a cash trade, zero for a margin trade.
num_found	OUT	Number of trade rows returned.
num_updated	OUT	Number of SETTLEMENT rows that were modified.
settlement_amount[]	OUT	Cash amount of settlement.
settlement_cash_due_date[]	OUT	Date by which customer or brokerage must receive the cash.
settlement_cash_type[]	OUT	Type of cash settlement involved: cash or margin.
status	OUT	Code indicating the execution status for this frame.
trade_history[][3]	OUT	Array of timestamps of when the trade history was updated.
trade_history_status_id[][3]	OUT	Array of status type identifiers.
trade_list[]	OUT	Trade ID actually used for retrieving data.
trade_price[]	OUT	Unit price at which the security was traded.

Trade-Update_Frame-2 pseudo-code : Get trade information for the first N trades of a given customer account from a given point in time and modify some of the SETTLEMENT rows.

```

{
    declare i int
    declare cash_type char(40)
    start transaction

    // Get trade information
    // Will return between 0 and max_trades rows
    select first max_trades rows
        bid_price[]    = T_BID_PRICE,
        exec_name[]    = T_EXEC_NAME,
        is_cash[]      = T_IS_CASH,
        trade_list[]   = T_ID,
        trade_price[]  = T_TRADE_PRICE
    from
        TRADE
    where
        T_CA_ID = acct_id and

```

Trade-Update_Frame-2 pseudo-code : Get trade information for the first N trades of a given customer account from a given point in time and modify some of the SETTLEMENT rows.

```
T_ST_ID = "CMPT" and
T_DTS >= start_trade_dts and
T_DTS <= end_trade_dts
order by
    T_DTS asc

num_found    = row_count
num_updated  = 0

// Get extra information for each trade in the trade list.
for (i = 0; i < num_found; i++) {
    if (num_updated < max_updates) then {
        // Modify the SETTLEMENT row for this trade.
        select
            cash_type = SE_CASH_TYPE
        from
            SETTLEMENT
        where
            SE_T_ID = trade_list[i]

        if (is_cash[i]) then {
            if (cash_type == "Cash Account") then
                cash_type = "Cash"
            else
                cash_type = "Cash Account"
        }
        else
            if (cash_type == "Margin Account") then
                cash_type = "Margin"
            else
                cash_type = "Margin Account"
        }

        update
            SETTLEMENT
        set
            SE_CASH_TYPE = cash_type
        where
            SE_T_ID = trade_list[i]

        num_updated = num_updated + row_count
    }

    // Get settlement information
    // Will return only one row for each trade
    select
        settlement_amount[i]          = SE_AMT,
        settlement_cash_due_date[i]    = SE_CASH_DUE_DATE,
        settlement_cash_type[i]        = SE_CASH_TYPE
```

Trade-Update_Frame-2 pseudo-code : Get trade information for the first N trades of a given customer account from a given point in time and modify some of the SETTLEMENT rows.

```

from
    SETTLEMENT
where
    SE_T_ID = trade_list[i]

// get cash information if this is a cash transaction
// Should return only one row for each trade that was a cash transaction
if (is_cash[i]) then {
    select
        cash_transaction_amount[i] = CT_AMT,
        cash_transaction_dts[i]    = CT_DTS
        cash_transaction_name[i]   = CT_NAME
    from
        CASH_TRANSACTION
    where
        CT_T_ID = trade_list[i]
}

// read trade_history for the trades
// Will return 2 or 3 rows per trade
select
    trade_history_dts[i][]      = TH_DTS,
    trade_history_status_id[i][] = TH_ST_ID
from
    TRADE_HISTORY
where
    TH_T_ID = trade_list[i]
order by
    TH_DTS

} // end for loop

commit transaction
}

```

3.3.4.5 Trade-Update Transaction Frame 3 of 3

The third **Frame** returns N trades for a given security on or after a specified start point in time and modifies some data from the CASH_TRANSACTION table. If the specified start time is too close to the end of historical trade data, then it is possible that less than N trades for the security may be present, in which case fewer rows may be modified.

The EGenTxnHarness controls the execution of **Frame 3** as follows:

```
[...]
    else if( frame_to_execute == 3 )
    {
        invoke (Trade-Update_Frame-3)
        frame_executed = 3
    }
}
```

Trade-Update Frame 3 of 3 Parameters:

Field	Direction	Description
end_trade_dts	IN	Point in time at which to stop search.
max_acct_id	IN	Maximum customer account identifier.
max_trades	IN	Number of trades to find. The default value (20) is set in TTradeUpdateSettings.MaxRowsFrame3 in DriverParameterSettings.h.
max_updates	IN	Number of CASH_TRANSACTION rows to modify. The default value (20) is set in TTradeUpdateSettings.MaxRowsToUpdateFrame3 in DriverParameterSettings.h.
start_trade_dts	IN	Point in time from which to start search.
symbol	IN	Security for which to find trades.
acct_id[]	OUT	Array of accounts for which the trades were done.
cash_transaction_amount[]	OUT	Amount of the cash transaction.
cash_transaction_dts[]	OUT	Date and time stamp of when the transaction took place.
cash_transaction_name[]	OUT	Description of the cash transaction.
exec_name[]	OUT	Array of name of the person who executed each of the trades.
is_cash[]	OUT	Flag that is non-zero for a cash trade, zero for a margin trade.
num_found	OUT	Number of TRADE rows returned.
num_updated	OUT	Number of CASH_TRANSACTION rows modified.
price[]	OUT	Array of the price that was paid in each trade.
quantity[]	OUT	Array of the quantity of security bought in each trade.
s_name[]	OUT	Name of the security traded.
settlement_amount[]	OUT	Cash amount of settlement.
settlement_cash_due_date[]	OUT	Date by which the customer or brokerage must receive the cash.
settlement_cash_type[]	OUT	Type of cash settlement involved: cash or margin.
status	OUT	Code indicating the execution status for this frame.
trade_dts[]	OUT	Array of the timestamps for when the trade was requested.
trade_history_dts[][3]	OUT	Array of timestamps of when the trade history was updated.
trade_history_status_id[][3]	OUT	Array of status type identifiers.
trade_list[]	OUT	Array of T_IDs found.
type_name[]	OUT	Array of the trade type name for each trade.
trade_type[]	OUT	Array of the trade type for each trade.

Trade-Update_Frame-3 pseudo-code: Get a list of N trades executed for a certain security starting from a given point in time and modify some of the CASH_TRANSACTION rows.

```

{
    declare i int
    declare ct_name char(100)
    start transaction
    // Will return between 0 and max_trades rows.
    select first max_trades rows
        acct_id[]    = T_CA_ID,
        exec_name[]  = T_EXEC_NAME,
        is_cash[]    = T_IS_CASH,
        price[]      = T_TRADE_PRICE,
        quantity[]   = T_QTY,
        s_name[]     = S_NAME,
        trade_dts[]  = T_DTS,
        trade_list[] = T_ID,
        trade_type[] = T_TT_ID,
        type_name[]  = TT_NAME
    from
        TRADE,
        TRADE_TYPE,
        SECURITY
    where
        T_S_SYMB = symbol and
        T_ST_ID = "CMPT" and
        T_DTS >= start_trade_dts and
        T_DTS <= end_trade_dts and
        TT_ID = T_TT_ID and
        S_SYMB = T_S_SYMB
        // The max_acct_id "where" clause is a hook used for engineering purposes
        // only and is not required for benchmark publication purposes.
        // and
        //T_CA_ID <= max_acct_id
    order by
        T_DTS asc

    num_found = row_count
    num_updated = 0

    // Get extra information for each trade in the trade list.
    for (i = 0; i < num_found; i++) {
        // Get settlement information
        // Will return only one row for each trade
        select
            settlement_amount[i]      = SE_AMT,
            settlement_cash_due_date[i] = SE_CASH_DUE_DATE,
            settlement_cash_type[i]     = SE_CASH_TYPE

```

Trade-Update_Frame-3 pseudo-code: Get a list of N trades executed for a certain security starting from a given point in time and modify some of the CASH_TRANSACTION rows.

```
from
    SETTLEMENT
where
    SE_T_ID = trade_list[i]

// get cash information if this is a cash transaction
// Will return only one row for each trade that was a cash transaction
if (is_cash[i]) then {
    if (num_updated < max_updates) then {
        // Modify the CASH_TRANSACTION row for this trade.
        select
            ct_name = CT_NAME
        from
            CASH_TRANSACTION
        where
            CT_T_ID = trade_list[i]

        if (ct_name like "% shares of %") then
            ct_name = type_name[i] + " " + quantity[i] + " Shares of " + s_name[i]
        else
            ct_name = type_name[i] + " " + quantity[i] + " shares of " + s_name[i]

        update
            CASH_TRANSACTION
        set
            CT_NAME = ct_name
        where
            CT_T_ID = trade_list[i]

        num_updated = num_updated + row_count
    }

    select
        cash_transaction_amount[i] = CT_AMT,
        cash_transaction_dts[i]    = CT_DTS
        cash_transaction_name[i]   = CT_NAME
    from
        CASH_TRANSACTION
    where
        CT_T_ID = trade_list[i]
}

// read trade_history for the trades
// Will return 2 or 3 rows per trade
select
    trade_history_dts[i][]      = TH_DTS,
    trade_history_status_id[i][] = TH_ST_ID
from
    TRADE_HISTORY
```

Trade-Update_Frame-3 pseudo-code: Get a list of N trades executed for a certain security starting from a given point in time and modify some of the CASH_TRANSACTION rows.

```

    where
        TH_T_ID = trade_list[i]
    order by
        TH_DTS asc

} // end for loop

commit transaction

}

```

3.3.5 The Trade-Status Transaction

The Trade-Status **transaction** is invoked by the **CE** component of the **Driver**. This **transaction** returns the status of fifty trades. A list of all trades for a given customer's account is retrieved and the fifty most recently placed trades, ordered by date and time are selected. For each trade the security's name and exchange are retrieved, along with the customer's name and the broker's name.

The Trade-Status **transaction** consists of a single **Frame**.

3.3.5.1 Trade-Status Transaction Parameters

The inputs to the Trade-Status **transaction** are generated by the **EGen CE** code in *CETxnInputGenerator.cpp* and the data structures used to communicate the input and output parameters must match the **EGenTxnHarness** data structures defined in TxnHarnessStructs.h.

Trade-Status Interfaces	Module/Data Structure
CE Input generation	GenerateTradeStatusInput()
Transaction Input/Output Structure	TTradeStatusTxnInput TTradeStatusTxnOutput
Frame 1 Input/Output Structure	TTradeStatusFrame1Input TTradeStatusFrame1Output

Trade-Status Transaction Parameters:

Field	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. A single customer account id, as defined by CA_ID in CUSTOMER_ACCOUNT, is chosen at random, uniformly, from the range of customer account id's for the chosen customer.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.
status_name[]	OUT	A list of character strings, each character string as defined by ST_NAME in STATUS_TYPE, representing the current status of a trade.
trade_id[]	OUT	A list of numbers, each number as defined by T_ID in TRADE, assigned by the brokerage or exchange to identify the specific trade being requested.

3.3.5.2 Trade-Status Transaction Database-Footprint

The Trade-Status **Database-Footprint** is as follows:

Trade-Status Database-Footprint		
Table	Column	Frame
		1
BROKER	B_NAME	Return
CUSTOMER	C_F_NAME	Return
	C_L_NAME	Return
EXCHANGE	EX_NAME	Return
SECURITY	S_NAME	Return
STATUS_TYPE	ST_NAME	Return
TRADE	T_CHRG	Return
	T_DTS	Return
	T_EXEC_NAME	Return
	T_ID	Return
	T_QTY	Return
	T_S_SYMB	Return
TRADE_TYPE	TT_NAME	Return
Transaction Control		Start Commit

3.3.5.3 Trade-Status Transaction Frame 1 of 1

The database access methods used in **Frame 1** are all **Returns**.

The **EGenTxnHarness** controls the execution of **Frame 1** as follows:

```
{  
    invoke (Trade-Status_Frame-1)  
}
```

Trade-Status Frame 1 of 1 Parameters:

Field	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. A single customer account id, as defined by CA_ID in CUSTOMER_ACCOUNT, is chosen at random, uniformly, from the range of customer account ids for the chosen customer.
broker_name	OUT	A character string, as defined by B_NAME in BROKER, representing the name of the broker who executes transactions on behalf of the customer
charge[]	OUT	A list of numbers, each number as defined by T_CHRG in TRADE, representing the cost of executing the trade as charged by the broker.
cust_f_name	OUT	A character string, as defined by C_F_NAME in CUSTOMER, representing the first name of the customer who owns the account (acct_id).
cust_l_name	OUT	A character string, as defined by C_L_NAME in CUSTOMER, representing the last name of the customer who owns the account (acct_id).

ex_name[]	OUT	A list of character strings, each character string as defined by EX_NAME in EXCHANGE, representing the name of the security exchange where the security is traded.
exec_name[]	OUT	A list of character strings, each character string as defined by T_EXEC_NAME in TRADE, representing the name of the person who initiated the trade on behalf of the customer (c_f_name, c_l_name).
s_name[]	OUT	A list of character strings, each character string as defined by S_NAME in SECURITY, representing the name of the security as listed with the exchange.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.
status_name[]	OUT	A list of character strings, each character string as defined by ST_NAME in STATUS_TYPE, representing the current status of the trade.
symbol []	OUT	A list of character strings, each character string as defined by S_SYMB in SECURITY, representing the specific security, as listed with the exchange, being traded in the trade.
trade_dts[]	OUT	A list of dates and times, each data and time as defined by T_DTS in TRADE, at which the Trade-Request was processed by the broker.
trade_id[]	OUT	A list of numbers, each number as defined by T_ID in TRADE, assigned by the brokerage or exchange to identify the specific trade being requested.
trade_qty[]	OUT	A list of numbers, each number as defined by T_QTY in TRADE, representing the quantity of the security being traded in the Trade-Request.
type_name[]	OUT	A list of character strings, each character string as defined by TT_NAME in TRADE_TYPE, representing the type of trade being executed on behalf of the customer.

Trade-Status_Frame-1 pseudo-code: Retrieve information on the 50 most recent trades

```

{
  start transaction
  // Only want 50 rows, the 50 most recent trades for this customer account
  select first 50 row
    trade_id[]      = T_ID,
    trade_dts[]     = T_DTS,
    status_name[]   = ST_NAME,
    type_name[]     = TT_NAME,
    symbol[]        = T_S_SYMB,
    trade_qty[]     = T_QTY,
    exec_name[]     = T_EXEC_NAME,
    charge[]        = T_CHRG,
    s_name[]        = S_NAME,
    ex_name[]       = EX_NAME
  from
    TRADE,
    STATUS_TYPE,
    TRADE_TYPE,
    SECURITY,
    EXCHANGE

```

Trade-Status_Frame-1 pseudo-code: Retrieve information on the 50 most recent trades

```
where
    T_CA_ID = acct_id and
    ST_ID = T_ST_ID and
    TT_ID = T_TT_ID and
    S_SYMB = T_S_SYMB and
    EX_ID = S_EX_ID
order by
    T_DTS desc

select
    cust_l_name = C_L_NAME,
    cust_f_name = C_F_NAME,
    broker_name = B_NAME
from
    CUSTOMER_ACCOUNT,
    CUSTOMER,
    BROKER
where
    CA_ID = acct_id and
    C_ID = CA_C_ID and
    B_ID = CA_B_ID

commit transaction
}
```

3.3.6 The Customer-Position Transaction

The Customer-Position **transaction** is invoked by the **CE** component of the **Driver**. This **transaction** takes a customer tax ID or customer ID as input and is responsible for calculating the current market value of each one of that customer's accounts. The **transaction** returns detailed information about the customer along with a list of accounts and their associated cash and asset values. If the `get_history` parameter is set to `TRUE`, the **transaction** will also return the customer's trade history.

The Customer-Position **transaction** is divided into 3 **Frames**.

3.3.6.1 Customer-Position Transaction Parameters

The inputs to the Customer Position **transaction** are generated by the **EGen CE** code in *CETxnInputGenerator.cpp* and the data structures used to communicate the input and output parameters must match the **EGenTxnHarness** data structures defined in *TxnHarnessStructs.h*.

Customer-Position Interfaces	Module/Data Structure
CE Input generation	GenerateCustomerPositionInput()
Transaction Input/Output Structure	TCustomerPositionTxnInput TCustomerPositionTxnOutput
Frame 1 Input/Output Structure	TCustomerPositionFrame1Input TCustomerPositionFrame1Output
Frame 2 Input/Output Structure	TCustomerPositionFrame2Input TCustomerPositionFrame2Output
Frame 3 Input/Output Structure	TCustomerPositionFrame3Output

Customer-Position Transaction Parameters:

Field	Direction	Description
acct_id_idx	IN	Index to one of the customers accounts. This indexed account will be used in frame 2 if get_history is TRUE.
cust_id	IN	Customer id or 0, selected by the driver.
get_history	IN	Selected by the driver to be TRUE or FALSE.
tax_id	IN	Customer tax id or empty string selected by the driver.
acct_id[max_acct_len]	OUT	Array of customer account IDs.
acct_len	OUT	Number of customer accounts (max_acct_len (10) or less)
asset_total[max_acct_len]	OUT	Array of asset totals for each customer account.
c_ad_id	OUT	Customer address identifier.
c_area_1	OUT	Area code for customer's first phone number.
c_area_2	OUT	Area code for customer's second phone number.
c_area_3	OUT	Area code for customer's third phone number.
c_ctype_1	OUT	Country code for customer's first phone number.
c_ctype_2	OUT	Country code for customer's second phone number.
c_ctype_3	OUT	Country code for customer's third phone number.
c_dob	OUT	Customer date of birth.
c_email_1	OUT	Customer's first email address.
c_email_2	OUT	Customer's second email address.
c_ext_1	OUT	Customer's extension for the first phone number.
c_ext_2	OUT	Customer's extension for the second phone number.
c_ext_3	OUT	Customer's extension for the third phone number.
c_fname	OUT	Customer first name.

c_gndr	OUT	Customer gender.
c_l_name	OUT	Customer last name.
c_local_1	OUT	Customer's first phone number.
c_local_2	OUT	Customer's second phone number.
c_local_3	OUT	Customer's third phone number.
c_m_name	OUT	Customer middle name.
c_st_id	OUT	Customer Status id.
c_tier	OUT	Customer tier.
cash_bal[max_acct_len]	OUT	Array of cash balances for each customer account.
hist_dts[max_hist_len]	OUT	Date for each transaction date from the transaction history
hist_len	OUT	Number of records from the transaction history
qty[max_hist_len]	OUT	Number of shares involved in each event from history
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.
symbol[max_hist_len]	OUT	Security involved in each event from history.
trade_id[max_hist_len]	OUT	Trade ID for each event from history.
trade_status[max_hist_len]	OUT	Trade Status fro each event from history.

3.3.6.2 Customer-Position Database-Footprint

The Customer-Position **Database-Footprint** is as follows:

Customer-Position Database-Footprint				
Table Name	Column	Frame		
		1	2*	3*
CUSTOMER	C_AD_ID	Return		
	C_AREA_1	Return		
	C_AREA_2	Return		
	C_AREA_3	Return		
	C_CTRY_1	Return		
	C_CTRY_2	Return		
	C_CTRY_3	Return		
	C_DOB	Return		
	C_EMAIL_1	Return		
	C_EMAIL_2	Return		
	C_EXT_1	Return		
	C_EXT_2	Return		
	C_EXT_3	Return		
	C_F_NAME	Return		

	C_GNDR	Return		
	C_L_NAME	Return		
	C_LOCAL_1	Return		
	C_LOCAL_2	Return		
	C_LOCAL_3	Return		
	C_M_NAME	Return		
	C_ST_ID	Return		
	C_TIER	Return		
CUSTOMER_ACCOUNT	CA_BAL	Return		
	CA_ID	Return		
HOLDING_SUMMARY	HS_QTY	Reference		
LAST_TRADE	LT_PRICE	Reference		
STATUS_TYPE	ST_NAME		Return	
TRADE_HISTORY	TH_DTS		Return	
TRADE	T_ID		Return	
	T_QTY		Return	
	T_S_SYMB		Return	
Transaction Control		Start	Commit	Commit

3.3.6.3 Customer-Position Transaction Frame 1 of 3

If the `cust_id` input parameter is set to 0, the **Frame** must use the `tax_id` input parameter to search the CUSTOMER table and find the ID of the customer. The **Frame** retrieves the detailed customer information and finds the cash balance for each of the customer's accounts as well as the total value of the holdings in each account. In addition to the detailed customer information, the **Frame** returns a list of accounts and their associated cash balance and asset value sorted by asset value.

The database access methods used in **Frame 1** are **Reference** and **Return**.

The **EGenTxnHarness** controls the execution of **Frame 1** as follows:

```
{
    invoke (Customer-Position_Frame-1)
}
```

Customer-Position Frame 1 of 3 Parameters:

Field	Direction	Description
<code>cust_id</code>	IN/OUT	Customer id or 0, selected by the driver.
<code>tax_id</code>	IN	Customer tax id or empty string selected by the driver.
<code>acct_id[max_acct_len]</code>	OUT	Array of customer account IDs.
<code>acct_len</code>	OUT	Number of customer accounts (<code>max_acct_len</code> (10) or less).
<code>asset_total[max_acct_len]</code>	OUT	Array of asset totals for each customer account.
<code>c_ad_id</code>	OUT	Customer address identifier.

c_area_1	OUT	Area code for customer's first phone number.
c_area_2	OUT	Area code for customer's second phone number.
c_area_3	OUT	Area code for customer's third phone number.
c_ctype_1	OUT	Country code for customer's first phone number.
c_ctype_2	OUT	Country code for customer's second phone number.
c_ctype_3	OUT	Country code for customer's third phone number.
c_dob	OUT	Customer date of birth.
c_email_1	OUT	Customer's first email address.
c_email_2	OUT	Customer's second email address.
c_ext_1	OUT	Customer's extension for the first phone number.
c_ext_2	OUT	Customer's extension for the second phone number.
c_ext_3	OUT	Customer's extension for the third phone number.
c_f_name	OUT	Customer first name.
c_gndr	OUT	Customer gender.
c_l_name	OUT	Customer last name.
c_local_1	OUT	Customer's first phone number.
c_local_2	OUT	Customer's second phone number.
c_local_3	OUT	Customer's third phone number.
c_m_name	OUT	Customer middle name.
c_st_id	OUT	Customer Status id.
c_tier	OUT	Customer tier.
cash_bal[max_acct_len]	OUT	Array of cash balances for each customer account.
status	OUT	Frame status.

Customer-Position_Frame-1 pseudo-code: Get the customer's total assets

```

{
  start transaction
  if (cust_id == null_cust_id) then {
    select
      cust_id = C_ID
    from
      CUSTOMER
    where
      C_TAX_ID = tax_id
  }

  select
    c_st_id   = C_ST_ID,
    c_l_name  = C_L_NAME,
    c_f_name  = C_F_NAME,

```

Customer-Position_Frame-1 pseudo-code: Get the customer's total assets

```

    c_m_name   = C_M_NAME,
    c_gndr     = C_GNDR,
    c_tier     = C_TIER,
    c_dob      = C_DOB,
    c_ad_id    = C_AD_ID,
    c_ctry_1   = C_CTRY_1,
    c_area_1   = C_AREA_1,
    c_local_1  = C_LOCAL_1,
    c_ext_1    = C_EXT_1,
    c_ctry_2   = C_CTRY_2,
    c_area_2   = C_AREA_2,
    c_local_2  = C_LOCAL_2,
    c_ext_2    = C_EXT_2,
    c_ctry_3   = C_CTRY_3,
    c_area_3   = C_AREA_3,
    c_local_3  = C_LOCAL_3,
    c_ext_3    = C_EXT_3,
    c_email_1  = C_EMAIL_1,
    c_email_2  = C_EMAIL_2
from
    CUSTOMER
where
    C_ID = cust_id

// Should return 1 to max_acct_len (10).
select first max_acct_len rows
    acct_id[]      = CA_ID,
    cash_bal[]     = CA_BAL,
    assets_total[] = ifnull((sum(HS_QTY * LT_PRICE)),0)
from
    CUSTOMER_ACCOUNT left outer join
    HOLDING_SUMMARY on HS_CA_ID = CA_ID,
    LAST_TRADE
where
    CA_C_ID   = cust_id and
    LT_S_SYMB = HS_S_SYMB
group by
    CA_ID, CA_BAL
order by
    3 asc

acct_len = row_count
}
```

3.3.6.4 Customer-Position Transaction Frame 2 of 3

This **Frame** is only executed if the **transaction** parameter `get_history` value is set to `TRUE`. Using the customer ID the **Frame** must search the `TRADE` and `TRADE_HISTORY` tables to find up to 30 history rows that correspond with the 10 most recent trades executed by the customer. For each event the **Frame** must return the `T_ID`, `T_S_SYMB`, `T_QTY`, `TH_DTS`, and `ST_NAME` for all events in a descending order of date found in `TH_DTS`. This **Frame** completes the work and commits the **transaction**

The database access methods used in **Frame 2** are all **Returns**.

The **EGenTxnHarness** controls the execution of **Frame 2** as follows:

```
{
    if (get_history == 1) then
    {
        frame2.acct_id = frame1.acct_id[acct_id_idx]
        invoke (Customer-Position_Frame-2)
        exit
    }
}
```

Customer-Position Frame 2 of 3 Parameters:

Field	Direction	Description
acct_id	IN	Customer account identifier
hist_dts[max_hist_len]	OUT	Date for each transaction date from the transaction history
hist_len	OUT	Number of records from the transaction history, at most max_hist_len which is 30.
qty[max_hist_len]	OUT	Number of shares involved in each event from history
status	OUT	Frame Status.
symbol[max_hist_len]	OUT	Security involved in each event from history.
trade_id[max_hist_len]	OUT	Trade ID for each event from history.
trade_status[max_hist_len]	OUT	Trade Status fro each event from history.

Customer-Position_Frame-2 pseudo-code: Get the customer's trade history

```
{
    // Should return 1 to 30 rows.
    select first 30 rows
        trade_id[]      = T_ID,
        symbol[]        = T_S_SYMB,
        qty[]           = T_QTY,
        trade_status[]  = ST_NAME,
        hist_dts[]      = TH_DTS
    from
```

Customer-Position_Frame-2 pseudo-code: Get the customer's trade history

```
(select first 10 rows
  T_ID as ID
  from
    TRADE
  where
    T_CA_ID = acct_id
  order by T_DTS desc) as T,
TRADE,
TRADE_HISTORY,
STATUS_TYPE
where
  T_ID = ID and
  TH_T_ID = T_ID and
  ST_ID = TH_ST_ID
order by
  TH_DTS desc

hist_len = row_count

commit transaction
}
```

3.3.6.5 Customer-Position Transaction Frame 3 of 3

This **Frame** is only executed if **get_history transaction** input parameter is set to FALSE. The **Frame** simply commits the **transaction** started in **Frame 1** and returns the status.

There are no database access methods used in **Frame 3**. This **Frame** is only using **transaction** control operations.

The **EGenTxnHarness** controls the execution of **Frame 3** as follows:

```
{
  if (get_history <> 1)
  {
    invoke (Customer-Position_Frame-3)
  }
}
```

Customer-Position Frame 3 of 3 Parameters:

Field	Direction	Description
status	OUT	Frame Status.

Customer-Position_Frame-3: End database transaction

Customer-Position_Frame-3: End database transaction

```
{  
    commit transaction  
}
```

3.3.7 The Broker-Volume Transaction

The Broker-Volume **transaction** is invoked by the **CE** component of the **Driver**. This **transaction** produces a descending list of the total potential volume generated by a list of brokers for all trade requests in a given sector. The **transaction** searches the **TRADE_REQUEST** table and finds all pending orders associated with the list of brokers for stocks of a specific sector. The value of each order is calculated based on bid price and quantity of shares and added to the running total for each broker. The **transaction** returns the list of brokers with their associated total volume sorted in descending volume order. In addition, the **transaction** returns the size of the output list and error status information.

The Broker-Volume **transaction** consists of a single **Frame**.

3.3.7.1 Broker-Volume Transaction Parameters

The inputs to the Broker-Volume **Transaction** are generated by the **EGen CE** code in *CETxnInputGenerator.cpp* and the data structures used to communicate the input and output parameters must match the **EGenTxnHarness** data structures defined in *TxnHarnessStructs.h*.

Broker-Volume Interfaces	Module/Data Structure
CE Input generation	GenerateBrokerVolumeInput()
Transaction Input/Output Structure	TBrokerVolumeTxnInput TBrokerVolumeTxnOutput
Frame 1 Input/Output Structure	TBrokerVolumeTxnInput TBrokerVolumeFrame1Output

Broker-Volume Transaction Parameters:

Field	Direction	Description
broker_list[]	IN	A list of twenty to forty distinct broker name strings as defined by B_NAME in BROKER table. Names are randomly selected from the broker range, with, uniform distribution. The list size is determined by the first null input name in the broker_list array. If customer partitioning is being used, 50% of the time the broker range is the range of brokers for the current customer partition. The rest of the time the broker range is the entire broker range. If no customer partitioning is used, all brokers are in the broker range.
sector_name	IN	A randomly selected sector name string as defined in SC_NAME in SECTOR table using uniform distribution.
list_len	OUT	Number of items in the list being returned.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.
volume[]	OUT	A list of numbers, sorted in descending order, representing the sum of all trade request values (TR_QTY * TR_BID_PRICE) in the TRADE_REQUEST table for stocks in a given sector grouped by broker names provided by broker_list. The list size is determined by list_len parameter.

3.3.7.2 Broker-Volume Transaction Database-Footprint

This **transaction** is read-only and makes no changes to the database. The Broker-Volume Database-Footprint is as follows:

Broker-Volume Database-Footprint		
Table	Column	Frame
		1
BROKER	B_NAME	Return
TRADE_REQUEST	TR_BID_PRICE	Reference
	TR_QTY	Reference
Transaction Control		Start Commit

3.3.7.3 Broker Volume Transaction Frame 1 of 1

The database access methods used in **Frame 1** are all **Returns**.

The **EGenTxnHarness** controls the execution of **Frame 1** as follows:

```
{  
    invoke (Broker-Volume_Frame-1)  
}
```

Broker-Volume Frame 1 of 1 Parameters:

Field	Direction	Description
broker_list[]	IN	A list of twenty to forty distinct broker name strings as defined by B_NAME in BROKER table. Names are randomly selected from the broker range, with, uniform distribution. The list size is determined by the first null input name in the broker_list array. If customer partitioning is being used, 50% of the time the broker range is the range of brokers for the current customer partition. The rest of the time the broker range is the entire broker range. If no customer partitioning is used, all brokers are in the broker range.
sector_name	IN	A randomly selected sector name string as defined in SC_NAME in SECTOR table using uniform distribution.
broker_name[]	OUT	A list of broker name strings sorted in descending order of the "volume" associated with the broker. The list size is determined by list_len parameter.
list_len	OUT	Number of items in the list being returned.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.
volume[]	OUT	A list of numbers, sorted in descending order, representing the sum of all trade request values (TR_QTY * TR_BID_PRICE) in the TRADE_REQUEST table for stocks in a given sector grouped by broker names provided by broker_list. The list size is determined by list_len parameter.

Broker-Volume_Frame-1 pseudo-code: Broker Volume

```
{
    start transaction
    // Should return 0 to 40 rows
    select
        broker_name[] = B_NAME,
        volume[]      = sum(TR_QTY * TR_BID_PRICE)
    from
        TRADE_REQUEST,
        SECTOR,
        INDUSTRY,
        COMPANY,
        BROKER,
        SECURITY,
        CUSTOMER_ACCOUNT
    where
        TR_CA_ID = CA_ID and
        CA_B_ID = B_ID and
        TR_S_SYMB = S_SYMB and
        S_CO_ID = CO_ID and
        CO_IN_ID = IN_ID and
        SC_ID = IN_SC_ID and
        B_NAME in (broker_list) and
        SC_NAME = sector_name
    group by
        B_NAME
    order by
        2 DESC

    list_len = row_count
    commit transaction
}
```

3.3.8 The Security-Detail Transaction

The Security-Detail **transaction** is invoked by the **CE** component of the **Driver**. This **transaction** returns all available information related to a given security. A security symbol is passed to the **transaction** as an input parameter. Based on the symbol, the **transaction** searches the **SECURITY**, **COMPANY**, **COMPANY_COMPETITOR**, **EXCHANGE**, **FINANCIAL**, **DAILY_MARKET**, **LAST_TRADE**, **NEWS_ITEM** and **NEW_XFER** tables to collect detailed information about the security.

The Security-Detail **transaction** consists of a single **Frame**.

3.3.8.1 Security-Detail Transaction Parameters

The inputs to the Security-Detail **transaction** are generated by the **EGen CE** code in *CETxnInputGenerator.cpp* and the data structures used to communicate the input and output parameters must match the **EGenTxnHarness** data structures defined in *TxnHarnessStructs.h*.

Security-Detail Interfaces	Module/Data Structure
CE Input generation	GenerateSecurityDetailInput()
Transaction Input/Output Structure	TSecurityDetailTxnInput TSecurityDetailTxnOutput
Frame 1 Input/Output Structure	TSecurityDetailFrame1Input TSecurityDetailFrame1Output

Security-Detail Transaction Parameters:

Field	Direction	Description
access_lob_flag	IN	If TRUE, access the complete news articles for the company. If FALSE, access just the news headlines and summaries.
max_rows_to_return	IN	An integer value, randomly selected between 5 and 20 with a uniform distribution. This value determines how many rows must be returned from the DAILY_MARKET table for this security.
start_day	IN	A date randomly selected from a uniform distribution of dates between 3 January 2000 and max_rows_to_return days before 1 January 2005. The DAILY_MARKET table contains data for the period 3 January 2000 to 31 December 2004. The transaction will return max_rows_to_return worth of rows from the DAILY_MARKET table for this security beginning with the row for start_day.
symbol	IN	Security symbol, randomly selected from a uniform distribution.
last_vol	OUT	Volume of last trade
news_len	OUT	Number of news items returned in news array.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.

3.3.8.2 Security-Detail Transaction Database-Footprint

The Security-Detail **Database-Footprint** is as follows:

Security-Detail Database-Footprint		
Table	Column	Frame
		1
ADDRESS	AD_CTRY	Return
	AD_LINE1	Return
	AD_LINE2	Return
	AD_ZC_CODE	Return
COMPANY	CO_CEO	Return
	CO_DESC	Return
	CO_NAME	Return
	CO_OPEN_DATE	Return
	CO_SP_RATE	Return
	CO_ST_ID	Return
	CP_CO_ID	Reference
	CP_COMP_CO_ID	Reference

	CP_IN_ID	Reference
DAILY_MARKET	DM_CLOSE	Return
	DM_DATE	Return
	DM_HIGH	Return
	DM_LOW	Return
	DM_VOL	Return
EXCHANGE	EX_CLOSE	Return
	EX_DESC	Return
	EX_NAME	Return
	EX_NUM_SYMB	Return
	EX_OPEN	Return
FINANCIAL	FI_ASSETS	Return
	FI_BASIC_EPS	Return
	FI_DILUT_EPS	Return
	FI_INVENTORY	Return
	FI_LIABILITY	Return
	FI_MARGIN	Return
	FI_NET_EARN	Return
	FI_OUT_BASIC	Return
	FI_OUT_DILUT	Return
	FI_QTR	Return
	FI_QTR_START_DATE	Return
	FI_REVENUE	Return
	FI_YEAR	Return
INDUSTRY	IN_NAME	Return
LAST_TRADE	LT_OPEN_PRICE	Return
	LT_PRICE	Return
	LT_VOL	Return
NEWS_ITEM	NI_AUTHOR	Return
	NI_DTS	Return
	NI_HEADLINE	Return*
	NI_ITEM	Return*
	NI_SOURCE	Return
	NI_SUMMARY	Return*
NEWS_XREF	NX_CO_ID	Reference
	NX_NI_ID	Reference
SECURITY	S_52_WK_HIGH	Return
	S_52_WK_HIGH_DATE	Return
	S_52_WK_LOW	Return

	S_52_WK_LOW_DATE	Return
	S_DIVIDEND	Return
	S_NAME	Return
	S_NUM_OUT	Return
	S_PE	Return
	S_START_DATE	Return
	S_YIELD	Return
ZIP_CODE	ZC_DIV	Return
	ZC_TOWN	Return
Transaction Control		Start Commit

3.3.8.3 Security Detail Transaction Frame 1 of 1

The database access methods used in **Frame 1** are **Returns** and References.

The **EGenTxnHarness** controls the execution of **Frame 1** as follows:

```
{
    invoke (Security-Detail_Frame-1)
}
```

Security-Detail Frame 1 of 1 Parameters:

Field	Direction	Description
access_lob_flag	IN	If TRUE, access the complete news articles for the company. If FALSE, access just the news headlines and summaries.
max_rows_to_return	IN	An integer value, randomly selected between 5 (iSecurityDetailMinRows) and 20 (iSecurityDetailMaxRows) with a uniform distribution. This value determines how many rows must be returned from the DAILY_MARKET table for this security.
start_day	IN	A date randomly selected from a uniform distribution of dates between 3 January 2000 and max_rows_to_return before 31 December 2004. The DAILY_MARKET table contains data for the period 3 January 2000 to 31 December 2004. The transaction will return max_rows_to_return worth of rows from the DAILY_MARKET table for this security beginning with the row for start_day.
symbol	IN	Security symbol, randomly selected from a uniform distribution.
52_wk_high	OUT	Number showing 52 week high value for the security.
52_wk_high_date	OUT	Date showing when the 52_wk_high happened.
52_wk_low	OUT	Number showing 52 week low value for the security.
52_wk_low_date	OUT	Date showing when 52_wk_low happened.
ceo_name	OUT	CEO name, based on a list of distinct first and last names.
co_ad_etry	OUT	Company country, USA or Canada
co_ad_div	OUT	Company county or state or province
co_ad_line1	OUT	Line 1 from a real company address
co_ad_line2	OUT	Line 2 from a real company address

co_ad_town	OUT	Company town
co_ad_zip	OUT	Company ZIP or postal code. Contains partly realistic US or Canadian ZIP codes
co_desc	OUT	Short description of the company. Readable English text.
co_name	OUT	Company name
co_st_id	OUT	Contains the value 'ST1'
cp_co_name[max_comp_len]	OUT	Array of strings containing the company names of competitors for this securities' company. EGen loads the COMPANY_COMPETITOR table with 3 competitors for each company, so max_comp_len is 3.
cp_in_name[max_comp_len]	OUT	Array of strings containing the name of the industries in which competitors compete with this securities' company. EGen loads the COMPANY_COMPETITOR table with 3 competitors for each company, so max_comp_len is 3.
day[max_day_len]	OUT	Array of numbers containing daily data. max_day_len is a constant set to 20.
day_len	OUT	Elements in the Day array
divid	OUT	Number containing security dividend
ex_ad_ctry	OUT	Exchange country
ex_ad_div	OUT	Exchange county or town or province
ex_ad_line1	OUT	Line 1 from real exchange address
ex_ad_line2	OUT	Line 2 from real exchange address
ex_ad_town	OUT	Exchange town
ex_ad_zip	OUT	Exchange ZIP code
ex_close	OUT	Time the exchange closes, 2 possible values.
ex_date	OUT	Date listed on exchange. Not earlier than Start_date
ex_desc	OUT	Description of the exchange
ex_name	OUT	Name of the exchange. 4 values
ex_num_symb	OUT	Number of securities traded
ex_open	OUT	Time the exchange opens
fin[max_fin_len]	OUT	Array of numbers with financial data. max_fin_len (20) is a constant set in the EGen code.
fin_len	OUT	Length of the array
last_open	OUT	Price of security at last exchange open
last_price	OUT	Price for security
last_vol	OUT	Volume of last trade
news[max_news_len]	OUT	Array of news items about the security's company. max_new_len (2) is a constant set in the EGen code.
news_len	OUT	Number of news items returned in news array.
num_out	OUT	Number of outstanding shares. Valid range is 4,000,000 to 9,500,000,000.
open_date	OUT	Date the company opened. Valid range is 01/01/1800 to build date

pe_ratio	OUT	Price/earning ratio. A random value between 1.00 and 120.00
s_name	OUT	Security name, 6850 distinct values
sp_rate	OUT	Standards & Poor rating for the company, one of 39 values.
start_date	OUT	Date of trade started. Range id between 01/01/1900 and build date.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.
yield	OUT	Number containing yield for the security

Security-Detail_Frame-1 pseudo-code: Get all details about the security

```
DoSecDetailFrame1( SecDetailFrame1 *pSecDetailFrame1 )
```

```
{
  start transaction
  status = "success"
  select
    s_name          = S_NAME,
    co_id           = CO_ID,
    co_name         = CO_NAME,
    sp_rate         = CO_SP_RATE,
    ceo_name        = CO_CEO,
    co_desc         = CO_DESC,
    open_date       = CO_OPEN_DATE,
    co_st_id        = CO_ST_ID,
    co_ad_line1     = CA.AD_LINE1,
    co_ad_line2     = CA.AD_LINE2,
    co_ad_town      = ZCA.ZC_TOWN,
    co_ad_div       = ZCA.ZC_DIV,
    co_ad_zip       = CA.AD_ZC_CODE,
    co_ad_ctry      = CA.AD_CTRY,
    num_out         = S_NUM_OUT,
    start_date      = S_START_DATE,
    exch_date       = S_EXCH_DATE,
    pe_ratio        = S_PE,
    52_wk_high      = S_52WK_HIGH,
    52_wk_high_date = S_52WK_HIGH_DATE,
    52_wk_low       = S_52WK_LOW,
    52_wk_low_date  = S_52WK_LOW_DATE,
    divid          = S_DIVIDEND,
    yield          = S_YIELD,
    ex_ad_div       = ZEA.ZC_DIV,
    ex_ad_ctry      = EA.AD_CTRY,
    ex_ad_line1     = EA.AD_LINE1,
    ex_ad_line2     = EA.AD_LINE2,
    ex_ad_town      = ZEA.ZC_TOWN,
    ex_ad_zip       = EA.AD_ZC_CODE,
    ex_close        = EX_CLOSE,
    ex_desc         = EX_DESC,
    ex_name         = EX_NAME,
```

Security-Detail_Frame-1 pseudo-code: Get all details about the security

```
        ex_num_symb      = EX_NUM_SYMB,
        ex_open          = EX_OPEN
from
    SECURITY,
    COMPANY,
    ADDRESS CA,
    ADDRESS EA,
    ZIP_CODE ZCA,
    ZIP_CODE ZEA,
    EXCHANGE
where
    S_SYMB = symbol and
    CO_ID = S_CO_ID and
    CA.AD_ID = CO_AD_ID and
    EA.AD_ID = EX_AD_ID and
    EX_ID = S_EX_ID and
    ca.ad_zc_code = zca.zc_code and
    ea.ad_zc_code = zea.zc_code

// Should return max_comp_len (3) rows
select first max_comp_len rows
    cp_co_name[] = CO_NAME,
    cp_in_name[] = IN_NAME
from
    COMPANY_COMPETITOR, COMPANY, INDUSTRY
where
    CP_CO_ID = co_id and
    CO_ID = CP_COMP_CO_ID and
    IN_ID = CP_IN_ID

// Should return max_fin_len rows
select first max_fin_len (20) rows
    fin[].year          = FI_YEAR,
    fin[].qtr           = FI_QTR,
    fin[].strart_date   = FI_QTR_START_DATE,
    fin[].rev           = FI_REVENUE,
    fin[].net_earn      = FI_NET_EARN,
    fin[].basic_eps     = FI_BASIC_EPS,
    fin[].dilut_eps     = FI_DILUT_EPS,
    fin[].margin        = FI_MARGIN,
    fin[].invent        = FI_INVENTORY,
    fin[].assets        = FI_ASSETS,
    fin[].liab          = FI_LIABILITY,
    fin[].out_basic     = FI_OUT_BASIC,
    fin[].out_dilut     = FI_OUT_DILUT
from
    FINANCIAL
where
    FI_CO_ID = co_id
order by
```

Security-Detail_Frame-1 pseudo-code: Get all details about the security

```

    FI_YEAR asc,
    FI_QTR

fin_len = row_count

// Should return max_rows_to_return rows
// max_rows_to_return is between 5 and 20
select first max_rows_to_return rows
    day[].date = DM_DATE,
    day[].close = DM_CLOSE,
    day[].high = DM_HIGH,
    day[].low = DM_LOW,
    day[].vol = DM_VOL
from
    DAILY_MARKET
where
    DM_S_SYMB = symbol and
    DM_DATE >= start_day
order by
    DM_DATE asc

day_len = row_count

select
    last_price = LT_PRICE,
    last_open = LT_OPEN_PRICE,
    last_vol = LT_VOL
from
    LAST_TRADE
where
    LT_S_SYMB = symbol

// Should return max_news_len (2) rows
if (access_lob_flag)
    select first max_news_len rows
        news[].item = NI_ITEM,
        news[].dts = NI_DTS,
        news[].src = NI_SOURCE,
        news[].auth = NI_AUTHOR,
        news[].headline = "",
        news[].summary = ""
    from
        NEWS_XREF,
        NEWS_ITEM
    where
        NI_ID = NX_NI_ID and
        NX_CO_ID = co_id
else
    select first max_news_len rows
        news[].item = "",
```

Security-Detail_Frame-1 pseudo-code: Get all details about the security

```
        news[].dts      = NI_DTS,
        news[].src      = NI_SOURCE,
        news[].auth     = NI_AUTHOR,
        news[].headline = NI_HEADLINE,
        news[].summary  = NI_SUMMARY
    from
        NEWS_XREF,
        NEWS_ITEM
    where
        NI_ID = NX_NI_ID and
        NX_CO_ID = co_id

    news_len = row_count

    commit transaction
}
```

3.3.9 The Market-Feed Transaction

The Market-Feed **transaction** is invoked by the **MEE** component of the **Driver**. This **transaction** receives the latest prices for securities that come in on the market feed ticker. On average, 10 of the items on each ticker are as a result of trades submitted to the **MEE** by this brokerage house. The remaining items are generated by the **MEE** to simulate the reporting of trades from other brokerage houses. The maximum number of items in the ticker feed is 20 (max_feed_len constant in TxnHarnessStructs.h).

As the latest price of each item on the ticker is updated in the database, a check is also made to see if any pending limit orders in TRADE_REQUEST should be triggered as a result of a higher (or lower) ticker item price. If so, limit order processing is done by sending details of the trade request to the **MEE**, via the send_to_market interface. The maximum number of pending limit orders that may be triggered and sent to the **MEE** by one Market-Feed **transaction** is 40 (max_send_len constant in TxnHarnessStructs.h).

The Market-Feed **transaction** consists of a single **Frame**. The Market-Feed **transaction Frame Implementation** is allowed to process any number of ticker elements (from one to all) per database **transaction**.

3.3.9.1 Market-Feed Transaction Parameters

The inputs to the Market-Feed **Transaction** are generated by the **EGen MEE** code in MEE.cpp. The data structures used to communicate the input and output parameters must match the **EGenTxnHarness** data structures defined in TxnHarnessStructs.h.

Market-Feed Interfaces	Module/Data Structure
MEE Input generation	CMEESUTInterface::MarketFeed()
Transaction Input/Output Structure	TMarketFeedTxnInput TMarketFeedTxnOutput
Frame 1 Input/Output Structure	TMarketFeedFrame1Input TMarketFeedFrame1Output

Market-Feed Transaction Parameters:

Field	Direction	Description
price_quote[]	IN	A list of numeric prices the Market Emulator generated for each entry on the ticker list. Each security's price fluctuates between a low and high price, the fluctuation has a predefined frequency.
status_submitted	IN	The string ID value for the STATUS_TYPE Submitted status.
symbol[]	IN	A list of strings containing the Security Symbol for each security on the ticker. The security symbol string follows the definition of LT_S_SYMB in the LAST_TRADE table. The ticker was generated by the Market Emulator.
trade_qty[]	IN	A list of numbers representing the number of shares of a security that were traded for this ticker entry. The trade_qty is the same as the trade_qty requested in the Trade Request.
type_limit_buy	IN	The string ID value for the TRADE_TYPE Limit-Buy type.
type_limit_sell	IN	The string ID value for the TRADE_TYPE Limit-Sell type.
type_stop_loss	IN	The string ID value for the TRADE_TYPE Stop-Loss type.
send_len	OUT	Length of the output array. Ranges from 0 upwards. Average is about 4.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.

3.3.9.2 Market-Feed Database-Footprint

The Market-Feed **Database-Footprint** is as follows:

Market-Feed Database-Footprint		
Table Name	Column	Frame
		1
LAST_TRADE	LT_DTS	Modify
	LT_PRICE	Modify
	LT_VOL	Reference Modify
TRADE	T_ST_ID	Modify*
TRADE_HISTORY	1 Row	Add*
TRADE_REQUEST	TR_BID_PRICE	Return
	TR_QTY	Return
	TR_T_ID	Return
	TR_TT_ID	Return
	Row(s)	Remove*
Transaction Control		Start Commit (1 – max_feed_len)

3.3.9.3 Market-Feed Transaction Frame 1 of 1

For each entry in the ticker list, the **Frame** is responsible to find the symbol and to modify the row in the LAST_TRADE table for that symbol with the new price, to add the quantity traded to the daily volume, and to modify the last trade date. Next, any pending limit orders that should be triggered by this ticker price are selected, processed and sent to the **MEE** for further processing.

The database access methods used in **Frame 1** are **Modifies, Adds, References, Removes** and **Returns**.

The EGenTxnHarness controls the execution of **Frame 1** as follows:

```
{
    invoke (Market-Feed_Frame-1)
}
```

Market-Feed Frame 1 of 1 Parameters:

Field	Direction	Description
price_quote[]	IN	A list of numeric prices the Market Emulator generated for each entry on the ticker list. Each security's price fluctuates between a low and high price, the fluctuation has a predefined frequency.
status_submitted	IN	The string ID value for the STATUS_TYPE Submitted status.
symbol[]	IN	A list of strings containing the Security Symbol for each security on the ticker. The security symbol string follows the definition of LT_S_SYMB in the LAST_TRADE table. The ticker was generated by the Market Emulator.
trade_qty[]	IN	A list of numbers representing the number of shares of a security that were traded for this ticker entry. The trade_qty is the same as the trade_qty requested in the Trade Request.
type_limit_buy	IN	The string ID value for the TRADE_TYPE Limit-Buy type.
type_limit_sell	IN	The string ID value for the TRADE_TYPE Limit-Sell type.
type_stop_loss	IN	The string ID value for the TRADE_TYPE Stop-Loss type.
send_len	OUT	Length of the output arrays. Ranges from 0 upwards. Average is about 4.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.

Market-Feed_Frame-1 pseudo-code: Record the stock price and process any pending limit orders which are triggered by the ticker price.

```
{
    declare now_dts DATETIME
    declare TradeRequestBuffer[]

    get_current_dts(now_dts)
    status = "success"
    s = 0
    for (i = 1, i<=max_feed_len, i++) {
```

Market-Feed_Frame-1 pseudo-code: Record the stock price and process any pending limit orders which are triggered by the ticker price.

```
start transaction

update
    LAST_TRADE
set
    LT_PRICE = price_quote[i],
    LT_VOL = LT_VOL + trade_qty[i],
    LT_DTS = now_dts
where
    LT_S_SYMB = symbol[i]

declare request_list cursor for
select
    TR_T_ID,
    TR_BID_PRICE,
    TR_TT_ID,
    TR_QTY
from
    TRADE_REQUEST
where
    TR_S_SYMB = symbol[i] and (
        (TR_TT_ID = type_stop_loss and
         TR_BID_PRICE >= price_quote[i]) or
        (TR_TT_ID = type_limit_sell and
         TR_BID_PRICE <= price_quote[i]) or
        (TR_TT_ID = type_limit_buy and
         TR_BID_PRICE >= price_quote[i])
    )

open request_list
fetch from
    request_list
into
    trade_id,
    price_quote,
    trade_type,
    trade_qty
do until ((request_list.end_of_cursor) or (s >= max_send_len)) {
    update
        TRADE
    set
        T_DTS = now_dts,
        T_ST_ID = status_submitted
    where
        T_ID = trade_id

    delete
        TRADE_REQUEST
    where
        current of request_list
```

Market-Feed_Frame-1 pseudo-code: Record the stock price and process any pending limit orders which are triggered by the ticker price.

```
insert into
    TRADE_HISTORY
values (
    TH_T_ID = trade_id,
    TH_DTS = now_dts,
    TH_ST_ID = status_submitted
)

TradeRequestBuffer[s].symbol = symbol[i]
TradeRequestBuffer[s].trade_id = trade_id
TradeRequestBuffer[s].price_quote = price_quote
TradeRequestBuffer[s].trade_qty = trade_qty
TradeRequestBuffer[s].trade_type = trade_type
s = s + 1

fetch from
    request_list
into
    trade_id,
    price_quote,
    trade_type,
    trade_qty
} /* end of cursor fetch loop */
close request_list
commit transaction
} /* end of ticket loop */

send_len = s
// Sponsors should consider committing the database changes
// before calling the send_to_market interface.
for (i=0; i<max_send_len; i++)
{
    send_to_market(TradeRequestBuffer[i].symbol,
        TradeRequestBuffer[i].trade_id,
        TradeRequestBuffer[i].price_quote,
        TradeRequestBuffer[i].trade_qty,
        TradeRequestBuffer[i].trade_type);
}
}
```

3.3.10 The Market-Watch Transaction

The Market-Watch **transaction** is invoked by the **CE** component of the **Driver**. This **transaction** calculates the percentage change in value of the market capitalization of collection of securities at yesterday's closing price compared to the current market price for that collection of securities. The calculation is done by looking at yesterday's closing price for each security in the list. The closing price can be found in the DM_CLOSE column of the DAILY_MARKET table. For each security in the list, the closing price is multiplied by the number of outstanding shares for that security. The number of outstanding shares for a security can be found in the S_NUM_OUT column of the SECURITY table. The product is added to a running total for yesterday's closing market capitalization. The current price (found in the LT_PRICE column of the LAST_TRADE table) for each security in the list is also multiplied by the number of outstanding shares for that security. This product is added to a running sum for current market capitalization. The difference between the total market capitalization for yesterday's closing and the current total is expressed as a percentage and returned to the **EGenTxnHarness**. The **transaction** supports making this market watch calculation on a group of securities chosen based on the following list of criteria:

- **Prospective-Watch** - The collection of securities is chosen using all the securities in a customer's watch list. The customer identifier is chosen at random from the possible customer identifiers using a non-uniform by customer tier distribution.
- **Industry-Watch** - The collection of securities is chosen using all the securities in an industry. The industry name is chosen at random from the possible industry names using a uniform distribution.
- **Portfolio-Watch** - The collection of securities is chosen using all the securities that are held in a customer's account. A customer identifier is chosen at random from all possible customer identifiers using a non-uniform by customer tier distribution. The customer account identifier is chosen at random from all the possible accounts for that customer using a uniform distribution.

The Market-Watch **transaction** consists of a single **Frame**.

3.3.10.1 Market-Watch Transaction Parameters

The inputs to the Market-Watch **transaction** are generated by the **EGen CE** code in *CETxnInputGenerator.cpp*. The data structures used to communicate the input and output parameters must match the **EGenTxnHarness** data structures defined in **TxnHarnessStructs.h**.

Market-Watch Interfaces	Module/Data Structure
CE Input generation	GenerateMarketWatchInput()
Transaction Input/Output Structure	TMarketWatchTxnInput TMarketWatchTxnOutput
Frame 1 Input/Output Structure	TMarketWatchFrame1Input TMarketWatchFrame1Output

Market-Watch Transaction Parameters:

Field	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. A single customer account id, as defined by CA_ID in CUSTOMER_ACCOUNT, is chosen at random, uniformly, from the range of customer account id's for the chosen customer. This input will be used 35% of the time. The securities collection will be all the securities held this customer account. The other 65% of the time when this input is not being used its value will be 0.
cust_id	IN	A number randomly selected from the possible customer identifiers as defined by C_ID in CUSTOMER table using a non-uniform by customer tier distribution. This input will be used 60% of the time. The securities collection will be all the securities in this customer's watch list. The other 40% of the time when this input is not being used its value will be 0.
ending_co_id	IN	Company identifier of the last company in the range of 5,000 companies to be searched for companies in IN_NAME industry. The value will be starting_co_id + 4,999. This input will only be used when industry_name is used which is 5% of the time. The other 95% of the time when this input is not being used its value will be zero.
industry_name	IN	A randomly selected industry name string as defined in IN_NAME in INDUSTRY table using uniform distribution. This input will be used 5% of the time. The securities collection will be all the securities of companies in this industry. The other 95% of the time when this input is not being used its value will be an empty string.
starting_co_id	IN	A number randomly selected from the range of possible company identifiers minus 4,999. Company identifier of the first company in the range of 5,000 companies to be searched for companies in IN_NAME industry. This input will only be used when industry_name is used which is 5% of the time. The other 95% of the time when this input is not being used its value will be zero.
pct_change	OUT	Numeric value calculated during the transaction by finding the percentage change from yesterday's close of business capitalization for the collection of securities and the current capitalization for the collection of securities.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.

3.3.10.2 Market-Watch Transaction Database-Footprint

The Market-Watch **Database-Footprint** is as follows:

Market-Watch Database-Footprint		
Table	Column	Frame
		1
COMPANY	CO_ID	Reference*
	CO_IN_ID	Reference*
DAILY_MARKET	DM_CLOSE	Reference
HOLDING_SUMMARY	HS_S_SYMB	Reference*
INDUSTRY	IN_ID	Reference*
	IN_NAME	Reference*
LAST_TRADE	LT_PRICE	Reference

SECURITY	S_CO_ID	Reference*
	S_NUM_OUT	Reference
	S_SYMB	Reference*
WATCH_ITEM	WI_S_SYMB	Reference*
WATCH_LIST	WL_C_ID	Reference*
	WL_ID	Reference*
Transaction Control		Start Commit

3.3.10.3 Market-Watch Transaction Frame 1 of 1

The database access methods used in **Frame 1** are all **References**.

The EGenTxnHarness controls the execution of **Frame 1** as follows:

```
{
    invoke (Market-Watch_Frame-1)
}
```

Market-Watch Frame 1 of 1 Parameters:

Field	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. A single customer account id, as defined by CA_ID in CUSTOMER_ACCOUNT, is chosen at random, uniformly, from the range of customer account id's for the chosen customer. This input will be used 35% of the time. The securities collection will be all the securities held this customer account. The other 65% of the time when this input is not being used its value will be 0.
cust_id	IN	A number randomly selected from the possible customer identifiers as defined by C_ID in CUSTOMER table using a non-uniform by customer tier distribution. This input will be used 60% of the time. The securities collection will be all the securities in this customer's watch list. The other 40% of the time when this input is not being used its value will be 0.
ending_co_id	IN	Company identifier of the last company in the range of 5,000 companies to be searched for companies in IN_NAME industry. The value will be starting_co_id + 4,999. This input will only be used when industry_name is used which is 5% of the time. The other 95% of the time when this input is not being used its value will be zero.
industry_name	IN	A randomly selected industry name string as defined in IN_NAME in INDUSTRY table using uniform distribution. This input will be used 5% of the time. The securities collection will be all the securities of companies in this industry. The other 95% of the time when this input is not being used its value will be an empty string.

starting_co_id	IN	A number randomly selected from the range of possible company identifiers minus 4,999. Company identifier of the first company in the range of 5,000 companies to be searched for companies in IN_NAME industry. This input will only be used when industry_name is used which is 5% of the time. The other 95% of the time when this input is not being used its value will be zero.
pct_change	OUT	Numeric value calculated during the transaction by finding the percentage change from yesterday's close of business capitalization for the collection of securities and the current capitalization for the collection of securities.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.

Market-Watch_Frame-1 pseudo-code: Build list of securities and compute percentage

```

{
  start transaction
  if (cust_id != 0) then {
    declare stock_list cursor for
      select
        distinct WI_S_SYMB
      from
        WATCH_ITEM
      where
        WI_WL_ID in (select
                      WL_ID
                    from
                      WATCH_LIST
                    where
                      WL_C_ID = cust_id)
  } else if (industry_name != "") then {
    declare stock_list cursor for
      select
        S_SYMB
      from
        INDUSTRY,
        COMPANY,
        SECURITY
      where
        IN_NAME = industry_name and
        CO_IN_ID = IN_ID and
        CO_ID between (starting_co_id and ending_co_id) and
        S_CO_ID = CO_ID
  } else if (acct_id != 0) then {
    declare stock_list cursor for
      select
        HS_S_SYMB
      from
        HOLDING_SUMMARY
      where

```

Market-Watch_Frame-1 pseudo-code: Build list of securities and compute percentage

```
        HS_CA_ID = acct_id
    } else {
        status = bad_input_data
    }
    old_mkt_cap = 0.0
    new_mkt_cap = 0.0
    pct_change = 0.0
    if (status != bad_input_data) then {
        open stock_list
        do until (stock_list.end_of_cursor) {
            fetch from
                stock_list cursor
            into
                symbol

            select
                new_price = LT_PRICE
            from
                LAST_TRADE
            where
                LT_S_SYMB = symbol

            select
                s_num_out = S_NUM_OUT
            from
                SECURITY
            where
                S_SYMB = symbol

            // Only want one row, the most recent closing price for this security.
            select first 1 row
                old_price = DM_CLOSE
            from
                DAILY_MARKET
            where
                DM_S_SYMB = symbol
            order by
                DM_DATE desc

            old_mkt_cap += s_num_out * old_price
            new_mkt_cap += s_num_out * new_price
        }
        pct_change = 100 * (new_mkt_cap / old_mkt_cap - 1)
        close stock_list
        commit transaction
    } else {
        rollback transaction
    }
}
```


3.3.11 The Data-Maintenance Transaction

This **transaction** runs once per minute. It simulates periodic modifications to data tables that are mainly used for reference by the other **transactions**. The **Driver** provides as input the name of the table to be modified by the **transaction**.

Each time this **transaction** is run the **Driver** modifies the next table in the list. This means that each table in the list will only get modified once every twelve minutes. (WATCH_ITEM and WATCH_LIST are modified in the same **transaction**)

The following is the list of table names that can be passed as arguments to this **transaction**:

- ACCOUNT_PERMISSION
- ADDRESS
- COMPANY
- CUSTOMER
- CUSTOMER_TAXRATE
- DAILY_MARKET
- EXCHANGE
- FINANCIAL
- NEWS_ITEM
- SECURITY
- TAXRATE
- WATCH_ITEM (table WATCH_LIST is also modified)

The Data-Maintenance **transaction** consists of a single **Frame**.

The intent of the **transaction** is to modify data tables that would not otherwise be written to by the benchmark. The **EGenTxnHarness** will pick the next table in the list to modify, each time this **transaction** is run.

Below is a description of what kind of modification is done to each table when that table is selected:

1. ACCOUNT_PERMISSION - The **EGenTxnHarness** will pass a customer account identifier to the Data-Maintenance **transaction**. Each customer account will have at least one row in the ACCOUNT_PERMISSION table. The first ACCOUNT_PERMISSION row for the customer will be found. That row in the ACCOUNT_PERMISSION table will have an Access Control List (AP_ACL). That access control list will be updated to 1111 if it is not already 1111. If the access control list is already 1111, the access control list will be updated to 0011.
2. ADDRESS - 67% of the time **EGenTxnHarness** will pass a customer identifier to the Data-Maintenance **transaction**. The other 33% of the time **EGenTxnHarness** will pass a company identifier to the Data-Maintenance **transaction**. That customer's or company's ADDRESS will be modified. The AD_LINE2 will be set to "Apt. 10c" or to "Apt. 22" if it was already "Apt. 10c".
3. COMPANY - The **EGenTxnHarness** will pass a company identifier to the Data-Maintenance **transaction**. That company's Standard and Poor credit rating will be updated to "ABA" or to "AAA" if it was already "ABA".
4. CUSTOMER - The **EGenTxnHarness** will pass a customer identifier to the Data-Maintenance **transaction**. The ISP part of that customer's second email address

(C_EMAIL_2) will be updated to “@mindspring.com” or to “@earthlink.com” if it was already “@mindspring.com”.

5. CUSTOMER_TAXRATE – The **EGenTxnHarness** will pass a customer identifier to the Data-Maintenance **transaction**. A tax rate identified by “999” will be inserted into the CUSTOMER_TAXRATE table for that customer, or if the customer already has the “999” tax rate, the tax rate will be deleted. The “999” tax rate must exist in the TAXRATE table otherwise the **Foreign Key** constraint on the CUSTOMER_TAXRATE table will fail. The “999” tax rate will be inserted into the TAXRATE table during initial population of the database.
6. DAILY_MARKET – The **EGenTxnHarness** will pass a security symbol, a day in the month, and a random positive or negative number to the Data-Maintenance **transaction**. The DM_VOL will be updated by adding the random positive or negative number for rows in DAILY_MARKET for that security and for that day in the month, each month each year.
7. EXCHANGE – The **EGenTxnHarness** will not pass any additional information to the Data-Maintenance **transaction**. There are only four rows in the EXCHANGE table. Every row will have its EX_DESC updated. If EX_DESC does not already end with “LAST UPDATED ” and a date and time, that string will be appended to EX_DESC. Otherwise the date and time at the end of EX_DESC will be updated to the current date and time.
8. FINANCIAL – The **EGenTxnHarness** will pass a company identifier to the Data-Maintenance **transaction**. That company’s FI_QTR_START_DATES will be updated to the second of the month or to the first of the month if the dates were already the second of the month.
9. NEWS_ITEM – The **EGenTxnHarness** will pass a company identifier to the Data-Maintenance **transaction**. The NI_DTS for that company’s news items will be updated by one day.
10. SECURITY – The **EGenTxnHarness** will pass in a security symbol. That security’s S_EXCH_DATE will be incremented by one day.
11. TAXRATE – The **EGenTxnHarness** will pass in tax rate identifier to the Data-Maintenance **transaction**. That tax rate’s TX_NAME will be updated to end with the word “rate”, or the word “rate” will be removed from the end of the TX_NAME if TX_NAME already ends with the word “rate”.
12. WATCH_ITEM – The **EGenTxnHarness** will pass in a customer identifier to the Data-Maintenance **transaction**. A Watch List containing items with the security symbols “AA”, “ZAPS”, and “ZONS” will be added for the customer if they do not already have a Watch List with those items. If the customer already has a watch list with those items, the Watch List will be deleted.

3.3.11.1 Transaction Parameters

The inputs to the Data-Maintenance **transaction** are generated by the **EGenTxnHarness** in *CETxnInputGenerator.cpp*. The data structures used to communicate the input and output parameters must match the **EGenTxnHarness** data structures defined in **TxnHarnessStructs.h**.

Data-Maintenance Interfaces	Module/Data Structure
Input generation	GenerateDataMaintenanceInput()
Transaction Input/Output Structure	TDataMaintenanceTxnInput TDataMaintenanceTxnOutput

Frame 1 Input/Output Structure	TDDataMaintenanceFrame1Input TDDataMaintenanceFrame1Output
--------------------------------	---

Data-Maintenance Transaction Parameters:

Field	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. A single customer account id, as defined by CA_ID in CUSTOMER_ACCOUNT, is chosen at random, uniformly, from the range of customer account id's for the chosen customer. This input is used when table_name is "ACCOUNT_PERMISSION", otherwise it is set to 0.
c_id	IN	A number randomly selected from the possible customer identifiers as defined by C_ID in CUSTOMER table using a uniform distribution. This input is always used when table_name is "CUSTOMER", "CUSTOMER_TAXRATE" or "WATCH_ITEM". This input (instead of co_id) is used 67% of the time when table_name is "ADDRESS". Otherwise this input is set to 0.
co_id	IN	A number randomly selected from the possible company identifiers as defined by CO_ID in COMPANY table using a uniform distribution. This input is always used when table_name is "COMPANY", "FINANCIAL" or "NEWS_ITEM". This input (instead of c_id) is used 33% of the time when table_name is "ADDRESS". Otherwise this input is set 0.
day_of_month	IN	A number randomly selected from 1 to 31 with a uniform distribution. This input is only used when table_name is "DAILY_MARKET", otherwise it is set to 0. When table_name is "DAILY_MARKET" all the rows with this day of the Month in DM_DATE are modified.
symbol	IN	A string containing a Security Symbol. The security symbol string follows the definition of S_SYMB in the SECURITY table. This input is only used when table_name is "DAILY_MARKET", or "SECURITY", otherwise it is set to empty string.
table_name	IN	A string containing the name of the table to be modified. Valid values are "ACCOUNT_PERMISSION", "ADDRESS", "COMPANY", "CUSTOMER", "CUSTOMER_TAXRATE", "DAILY_MARKET", "EXCHANGE", "FINANCIAL", "NEWS_ITEM", "SECURITY", "TAXRATE", "WATCH_ITEM". This input is always used.
tx_id	IN	A string containing a tax identifier. The tax identifier string follows the definition of TX_ID in the TAXRATE table. This input is only used when table_name is "TAXRATE", otherwise it is set to empty string.
vol_incr	IN	A randomly selected positive or negative number. This number is only used when the table_name is "DAILY_MARKET", otherwise vol_incr is set to 0 and ignored. When table_name is "DAILY_MARKET" this number is added to DM_VOL.
status	OUT	Number representing the error status. Valid values are 0 for OK, non zero for an error. The sponsor can specify their own error codes.

3.3.11.2 Data-Maintenance Transaction Database-Footprint

This **transaction** includes a mix of Reference, Modify, Remove and Add operations. The **transaction** implementation would potentially require access to the following database tables and columns.

Data-Maintenance Database-Footprint		
Table Name	Column	Frame
		1
ACCOUNT_PERMISSION	AP_ACL	Reference * Modify *
	AP_CA_ID	Reference *
	Count(*)	Reference *
ADDRESS	AD_ID	Reference *
	AD_LINE2	Reference * Modify (1 row)*
COMPANY	CO_AD_ID	Reference*
	CO_ID	Reference *
	CO_SP_ID	Reference * Modify (1 row)*
CUSTOMER	C_AD_ID	Reference *
	C_EMAIL_2	Reference * Modify (1 row)*
	C_ID	Reference *
CUSTOMER_ACCOUNT	CA_ID	Reference *
CUSTOMER_TAXRATE	CX_C_ID	Reference *
	CX_TX_ID	Reference *
	Count(*)	Reference *
	1 row	Add or Remove
DAILY_MARKET	DM_DATE	Reference *
	DM_S_SYMB	Reference *
	DM_VOL	Reference * Modify *
EXCHANGE	EX_DESC	Reference * Modify *
	Count(*)	Reference *
FINANCIAL	FI_CO_ID	Reference *
	FI_QTR_START_DATE	Reference * Modify *
	Count(*)	Reference *
SECURITY	S_EXCH_DATE	Modify *
	S_SYMB	Reference *
NEWS_ITEM	NI_DTS	Modify *
	NI_ID	Reference *
NEWS_XREF	NX_CO_ID	Reference *
	NX_NI_ID	Reference *
TAXRATE	TX_ID	Reference *
	TX_NAME	Reference *

		Modify *
WATCH_ITEM	WI_S_SYMB	Reference *
	WI_WL_ID	Reference *
	Rows	Add 3 rows * OR Remove * (all rows with WI_WL_ID > max_initial_load_wl_id)
WATCH_LIST	WL_C_ID	Reference *
	WL_ID	Reference *
	1 row	Add*
	Whole row	Remove * (all rows with WL_ID > max_initial_load_wl_id)
Transaction Control		Start Commit

3.3.11.3 Data-Maintenance Transaction Frame 1 of 1

The EGenTxnHarness controls the execution of **Frame 1** as follows:

```
{
    invoke (Data-Maintenance_Frame-1)
}
```

Data-Maintenance Frame 1 of 1 Parameters:

Field	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. A single customer account id, as defined by CA_ID in CUSTOMER_ACCOUNT, is chosen at random, uniformly, from the range of customer account id's for the chosen customer. This input is used when table_name is "ACCOUNT_PERMISSION", otherwise it is set to 0.
c_id	IN	A number randomly selected from the possible customer identifiers as defined by C_ID in CUSTOMER table using a uniform distribution. This input is always used when table_name is "CUSTOMER", "CUSTOMER_TAXRATE" or "WATCH_ITEM". This input (instead of co_id) is used 67% of the time when table_name is "ADDRESS". Otherwise this input is set to 0.
co_id	IN	A number randomly selected from the possible company identifiers as defined by CO_ID in COMPANY table using a uniform distribution. This input is always used when table_name is "COMPANY", "FINANCIAL" or "NEWS_ITEM". This input (instead of c_id) is used 33% of the time when table_name is "ADDRESS". Otherwise this input is set 0.
day_of_month	IN	A number randomly selected from 1 to 31 with a uniform distribution. This input is only used when table_name is "DAILY_MARKET", otherwise it is set to 0. When table_name is "DAILY_MARKET" all the rows with this day of the Month in DM_DATE are modified.
symbol	IN	A string containing a Security Symbol. The security symbol string follows the definition of S_SYMB in the SECURITY table. This input is only used when table_name is "DAILY_MARKET", or "SECURITY", otherwise it is set to empty string.

table_name	IN	A string containing the name of the table to be modified. Valid values are "ACCOUNT_PERMISSION", "ADDRESS", "COMPANY", "CUSTOMER", "CUSTOMER_TAXRATE", "DAILY_MARKET", "EXCHANGE", "FINANCIAL", "SECURITY", "TAXRATE", "WATCH_ITEM". This input is always used.
tx_id	IN	A string containing a tax identifier. The tax identifier string follows the definition of TX_ID in the TAXRATE table. This input is only used when table_name is "TAXRATE", otherwise it is set to empty string.
vol_incr	IN	A randomly selected positive or negative number. This number is only used when the table_name is "DAILY_MARKET", otherwise vol_incr is set to 0 and ignored. When table_name is "DAILY_MARKET" this number is added to DM_VOL.
status	OUT	Number representing the error status. Valid values are 0 for OK, non zero for an error. The sponsor can specify their own error codes.

Data-Maintenance Frame 1: Update a table

```

/* Check which table is to be updated. */
if (strcmp(table_name, "ACCOUNT_PERMISSION")==0) {

    //ACCOUNT_PERMISSION
    //Update the AP_ACL to "1111" or "0011" in rows for a
    //customer account of c_id.

    acl = NULL

    select first 1 row
        acl = AP_ACL
    from
        ACCOUNT_PERMISSION
    where
        AP_CA_ID = acct_id

    if (strcmp(acl,"1111") != 0) {
        update
            ACCOUNT_PERMISSION
        set
            AP_ACL="1111"
        where
            AP_CA_ID = acct_id
    } else { /*ACL is "1111" change it to "0011" */
        update
            ACCOUNT_PERMISSION
        set
            AP_ACL = "0011"
        where
            AP_CA_ID = acct_id
    }
}

```

Data-Maintenance Frame 1: Update a table

```
} else if (strcmp(table_name,"ADDRESS")==0) {

    // ADDRESS
    // Change AD_LINE2 in the ADDRESS table for
    // the CUSTOMER with C_ID of c_id or the COMPANY with CO_ID of co_id.

    line2 = NULL
    ad_id = 0
    // Customer ID provided
    if (c_id != 0) {
        select
            line2 = AD_LINE2,
            ad_id = AD_ID
        from
            ADDRESS, CUSTOMER
        where
            AD_ID = C_AD_ID and
            C_ID = c_id
    }
    // Company ID provided
    else {
        select
            line2 = AD_LINE2,
            ad_id = AD_ID
        from
            ADDRESS, COMPANY
        where
            AD_ID = CO_AD_ID and
            CO_ID = co_id
    }
    if (strcmp(line2, "Apt. 10C") != 0) {
        update
            ADDRESS
        set
            AD_LINE2 = "Apt. 10C"
        where
            AD_ID = ad_id
    } else {
        update
            ADDRESS
        set
            AD_LINE2 = "Apt. 22"
        where
            AD_ID = ad_id
    }
} else if (strcmp(table_name,"COMPANY")==0) {
    // COMPANY
    // Update a row in the COMPANY table identified
    // by co_id, set the company's Standard and Poor
    // credit rating to "ABA" or to "AAA".
```

Data-Maintenance Frame 1: Update a table

```
sprate = NULL
select
    sprate = CO_SP_RATE
from
    COMPANY
where
    CO_ID = co_id
if (strcmp(sprate, "ABA") != 0) {
    update
        COMPANY
    set
        CO_SP_RATE = "ABA"
    where
        CO_ID = co_id
} else {
    update
        COMPANY
    set
        CO_SP_RATE = "AAA"
    where
        CO_ID = co_id
}
} else if (strcmp(table_name, "CUSTOMER") == 0) {
    // CUSTOMER
    // Update the second email address of a CUSTOMER
    // identified by c_id. Set the ISP part of the customer's
    // second email address to "@mindspring.com"
    // or "@earthlink.com".
    email2 = NULL
    len = 0
    lenMindspring = strlen("@mindspring.com")
    select
        email2 = C_EMAIL_2
    from
        CUSTOMER
    where
        C_ID = c_id
    len = strlen(email2)
    if ( ((len - lenMindspring) > 0) and
        (strcmp(substr(email2, len-lenMindspring,
            lenMindspring), "@mindspring.com") == 0) ) {
        update
            CUSTOMER
        set
            C_EMAIL_2 = substring(C_EMAIL_2, 1,
                charindex("@", C_EMAIL_2) + 'earthlink.com'
        where
            C_ID = c_id
    } else { /* set to @mindspring.com */
        update
```

Data-Maintenance Frame 1: Update a table

```
        CUSTOMER
    set
        C_EMAIL_2 = substring(C_EMAIL_2, 1,
            charindex("@",C_EMAIL_2) ) + 'mindspring.com'
    where
        C_ID = c_id
    }
} else if (strcmp(table_name, "CUSTOMER_TAXRATE") == 0) {

    // CUSTOMER_TAXRATE
    // A tax rate identified by "999" will be inserted into
    // the CUSTOMER_TAXRATE table for the CUSTOMER identified
    // by c_id.If the customer already has the "999" tax
    // rate, the tax Rate will be deleted. To preserve for
    // foreign key integrity The "999" tax rate must exist
    // in the TAXRATE table.

    rowcount = 0
    tax_rate = 0

    // Flip the special tax rate between 11% and 13%
    select
        tax_rate = TX_RATE
    from
        TAXRATE
    where
        TX_ID = "999"

    if (tax_rate = 0.11) {
        update
            TAXRATE
        set
            TX_RATE = 0.13
        where
            TX_ID = "999"
    }
    else {
        update
            TAXRATE
        set
            TX_RATE = 0.11
        where
            TX_ID = "999"
    }

    select
        rowcount = count(*)
    from
        CUSTOMER_TAXRATE
    where
```

Data-Maintenance Frame 1: Update a table

```
CX_C_ID = c_id and
CX_TX_ID = "999"

if (rowcount == 0) {
    /* No 999 tax rate for customer, */
    /* add one for them                */
    insert into
    CUSTOMER_TAXRATE (CX_TX_ID, CX_C_ID)
    values
    ("999", c_id)
} else {
    // There was a "999" tax rate for
    // this, customer delete it.
    delete from
    CUSTOMER_TAXRATE
    where
    CX_TX_ID = "999" and
    CX_C_ID = c_id
}
} else if (strcmp(table_name, "DAILY_MARKET") == 0) {
    // DAILY_MARKET
    // A security symbol, a day in the month and a
    // random positive or negative number are passed into
    // the Data-Maintenance function, when table_name
    // is DAILY_MARKET. The DM_VOL column in the DAILY_MARKET
    // table will be updated by adding the random positive or
    // negative number.
    // The rows to be updated are those for the security
    // whose symbol was passed in, and for that day in the
    // month that was passed in.
    update
    DAILY_MARKET
    set
    DM_VOL = DM_VOL + vol_incr
    where
    DM_S_SYMB = symbol
    and substring ((convert(char(8),DM_DATE,3),1,2) = day_of_month
} else if (strcmp(table_name, "EXCHANGE") == 0) {
    // EXCHANGE
    // Other than the table_name, no additional
    // parameters are used when the table_name is EXCHANGE.
    // There are only four rows in the EXCHANGE table. Every
    // row will have its EX_DESC updated. If EX_DESC does not
    // already end with "LAST UPDATED " and a date and time,
    // that string will be appended to EX_DESC. Otherwise the
    // date and time at the end of EX_DESC will be updated
    // to the current date and time.

    rowcount = 0
```

Data-Maintenance Frame 1: Update a table

```
select
    rowcount = count(*)
from
    EXCHANGE
where
    EX_DESC like "%LAST UPDATED%"

if (rowcount == 0) {
    update
        EXCHANGE
    set
        EX_DESC = EX_DESC + " LAST UPDATED " + getdatetime()
} else {
    update
        EXCHANGE
    set
        EX_DESC = substring(EX_DESC,1,
            len(EX_DESC)-len(getdatetime())) + getdatetime()
}
} else if (strcmp(table_name,"FINANCIAL") == 0) {
    // FINANCIAL
    // Update the FINANCIAL table for a company identified by
    // co_id. That company's FI_QTR_START_DATES will be
    // updated to the second of the month or to the first of
    // the month if the dates were already the second of the
    // month.

    rowcount = 0
    select
        rowcount = count(*)
    from
        FINANCIAL
    where
        FI_CO_ID = co_id and
        substring(convert(char(8),
            FI_QTR_START_DATE,2),7,2) = "01"
    if (rowcount > 0) {
        update
            FINANCIAL
        set
            FI_QTR_START_DATE = FI_QTR_START_DATE + 1 day
        where
            FI_CO_ID = co_id
    } else {
        update
            FINANCIAL
        set
            FI_QTR_START_DATE = FI_QTR_START_DATE - 1 day
        where
            FI_CO_ID = co_id
    }
}
```

Data-Maintenance Frame 1: Update a table

```
    }
} else if (strcmp(table_name, "NEWS_ITEM") == 0) {
    // NEWS_ITEM
    // Update the news items for a specified company.
    // Change the NI_DTS by 1 day.
    update
        NEWS_ITEM
    set
        NI_DTS = NI_DTS + 1day
    where
        NI_ID = (
            select
                first 1 NX_NI_ID
            from
                NEWS_XREF
            where
                NX_CO_ID = @co_id)
} else if (strcmp(table_name, "SECURITY") == 0) {
    // SECURITY
    // Update a security identified symbol, increment
    // S_EXCH_DATE by 1 day.
    update
        SECURITY
    set
        S_EXCH_DATE = S_EXCH_DATE + 1day
    where
        S_SYMB = symbol
} else if (strcmp(table_name, "TAXRATE") == 0) {
    // TAXRATE
    // Update a TAXRATE identified by tx_id. The tax rate's
    // TX_NAME Will be updated to end with the word "rate",
    // or the word "rate" will be removed from the end of the
    // TX_NAME if TX_NAME already ends with the word "rate".
    tx_name = NULL
    name_len = 0
    rate_len = 0
    pos = 0
    select
        tx_name = TX_NAME
    from
        TAXRATE
    where
        TX_ID = tx_id
    if (tx_name != NULL) {
        name_len = strlen(tx_name)
        rate_len = strlen(" rate")
        pos = name_len - rate_len
        if (pos < 0) {
            pos = 0
        }
    }
}
```

Data-Maintenance Frame 1: Update a table

```
// TX_NAME does not already end with " rate"
if (strcmp(substring(tx_name,pos,rate_len)," rate")!=0) {
    update
        TAXRATE
    set
        TX_NAME = TX_NAME + " rate"
    where
        TX_NAME not like "% rate" and
        TX_ID = tx_id
} else {
// row already has a TX_NAME that ends " rate"
update
    TAXRATE
set
    TX_NAME = substring(TX_NAME,1,
        len(TX_NAME)-len(" rate"))
where
    TX_ID = tx_id
}
}
} else if (strcmp(table_name,"WATCH_ITEM") == 0) {
// WATCH_ITEM
// A WATCH_LIST containing the WATCH_ITEMS with security
// symbols "AA", "ZAPS" and "ZONS" will be added for the
// customer identified by c_id, if the customer does not
// already have a watch list with those items. If the
// customer already has a watch list with those items,
// the watch list will be deleted.
wl_id = 0
wi_symbol = NULL
last_wl_id = 0
// If the CUSTOMER identified by c_id has a watch
// list with "AA", "ZAPS", "ZONS", it would have the
// highest WL_ID of that customer's watch lists.
select
    wl_id = max(WL_ID)
from
    WATCH_LIST
where
    WL_C_ID = c_id
if (wl_id != NULL) {
// See if the watch list has items other than
// "AA", "ZAPS", "ZONS" in it
select
    wi_symbol = max(WI_S_SYMB)
from
    WATCH_ITEM
where
    WI_WL_ID = wl_id and
    WI_S_SYMB not in ("AA","ZAPS","ZONS")
}
```

Data-Maintenance Frame 1: Update a table

```
if (wi_symbol != NULL) {
    // Customer does not have "AA", "ZAPS", "ZONS"
    // watch list. Find the last watch list
    // identifier used.
    select
        last_wl_id = max(WL_ID)
    from
        WATCH_LIST
    insert into
        WATCH_LIST (WL_ID, WL_C_ID)
    values
        (last_wl_id + 1, c_id)
    insert into
        WATCH_ITEM (WI_WL_ID, WI_S_SYMB)
    values
        (last_wl_id + 1, "AA")
    insert into
        WATCH_ITEM (WI_WL_ID, WI_S_SYMB)
    values
        (last_wl_id + 1, "ZAPS")
    insert into
        WATCH_ITEM (WI_WL_ID, WI_S_SYMB)
    values
        (last_wl_id + 1, "ZONS")
} else {
    // Customer already has the "AA", "ZAPS", "ZONS"
    // watch list, so delete it, and delete all the
    // other customers "AA" "ZAPS", "ZONS" watch lists,
    // so that WL_ID never gets too big.
    select
        wl_id = max(WL_ID)
    from
        WATCH_LIST
    where
        WL_ID in (
            select
                distinct(WI_WL_ID)
            from
                WATCH_ITEM
            where
                WI_S_SYMB != "AA" and
                WI_S_SYMB != "ZAPS" and
                WI_S_SYMB != "ZONS"
            group by
                WI_WL_ID
        )
    delete from
        WATCH_ITEM
    where
        WI_WL_ID > wl_id
}
```

Data-Maintenance Frame 1: Update a table

```
        delete from
            WATCH_LIST
        where
            WL_ID > wl_id
    }
} else {
    // Customer has no watch lists, so add the
    // "AA", "ZAPS", "ZONS" watch list
    select
        last_wl_id = max(WL_ID)
    from
        WATCH_LIST
    insert into
        WATCH_LIST (WL_ID, WL_C_ID)
    values
        (last_wl_id + 1, c_id)
    insert into
        WATCH_ITEM (WI_WL_ID, WI_S_SYMB)
    values
        (last_wl_id + 1, "AA")
    insert into
        WATCH_ITEM (WI_WL_ID, WI_S_SYMB)
    values
        (last_wl_id + 1, "ZAPS")
    insert into
        WATCH_ITEM (WI_WL_ID, WI_S_SYMB)
    values
        (last_wl_id + 1, "ZONS")
    }
}
commit transaction
}
```

3.3.12 The Trade-Cleanup Transaction

The Trade-Cleanup **transaction** is used to cancel any pending or submitted trades from the database. The **Sponsor** may use **EGenTxnHarness** to call Trade-Cleanup or may invoke the **transaction** by other means.

Trade-Cleanup is used to bring the database to a known state before the start of a **Test Run**.

The Trade-Cleanup **transaction** consists of a single **Frame**. The Trade-Cleanup **transaction** may be implemented using more than one database **transaction**.

3.3.12.1 Trade-Cleanup Transaction Parameters

The inputs to the Trade-Cleanup **transaction** are supplied by the **sponsor**. The data structures used to communicate the input and output parameters must match the **EGenTxnHarness** data structures defined in *TxnHarnessStructs.h*.

Trade-Cleanup Interfaces	Module/Data Structure
Transaction Input/Output Structure	TTradesCleanupTxnInput TTradesCleanupTxnOutput
Frame 1 Input/Output Structure	TTradesCleanupFrame1Input TTradesCleanupFrame1Output

Trade-Cleanup Transaction Parameters:

Field	Direction	Description
st_canceled_id	IN	Identifier for the “Canceled” trade order status – passed in for ease of benchmarking.
st_pending_id	IN	Identifier for the “Pending” trade order status – passed in for ease of benchmarking.
st_submitted_id	IN	Identifier for the “Submitted” trade order status – passed in for ease of benchmarking.
trade_id	IN	The trade identifier to be used as the start for handling outstanding submitted and/or pending limit trades.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.

3.3.12.2 Trade-Cleanup Transaction Database-Footprint

The Trade-Cleanup **Database-Footprint** is as follows:

Trade-Cleanup Database-Footprint		
Table	Column	Frame
		1
TRADE	T_DTS	Modify
	T_ID	Reference
	T_ST_ID	Modify
TRADE_HISTORY	Row(s)	Add
TRADE_REQUEST	Row(s)	Remove
	TR_T_ID	Reference
Transaction Control		Start Commit

3.3.12.3 Trade-Cleanup Transaction Frame 1 of 1

The database access methods used in **Frame 1** are a mixture of **References**, **Modifies**, **Removes** and **Adds**.

If **EGenTxnHarness** is used to invoke the **Frame**, it controls the execution of **Frame 1** as follows:


```

{
    invoke (Trade-Cleanup_Frame-1)
}

```

Trade-Cleanup Frame 1 of 1 Parameters:

Field	Direction	Description
st_canceled_id	IN	Identifier for the “Canceled” trade order status – passed in for ease of benchmarking.
st_pending_id	IN	Identifier for the “Pending” trade order status – passed in for ease of benchmarking.
st_submitted_id	IN	Identifier for the “Submitted” trade order status – passed in for ease of benchmarking.
trade_id	IN	The trade identifier to be used as the start for handling outstanding submitted and/or pending limit trades.
status	OUT	Code indicating the execution status for this transaction. Valid values are 0 for success or a non-zero value to report an error.

Trade-Cleanup_Frame-1 pseudo-code: cancel pending and submitted trades

```

{
    start transaction

    Declare t_id          TRADE_T
    Declare tr_t_id       TRADE_T
    Declare now_dts       DATETIME

    /* Find pending trades from TRADE_REQUEST */

    declare pending_list for
    select
        TR_T_ID
    from
        TRADE_REQUEST
    order by
        TR_T_ID

    open pending_list

    /* Insert a submitted followed by canceled record into TRADE_HISTORY, mark the trade
    canceled and delete the pending trade */

    do until (end_of_pending_list) {
        fetch from
            pending_list
        into
            tr_t_id

        get_current_dts ( now_dts )
    }
}

```

Trade-Cleanup_Frame-1 pseudo-code: cancel pending and submitted trades

```
insert into
    TRADE_HISTORY (
        TH_T_ID, TH_DTS, TH_ST_ID
    )
values (
    tr_t_id,          // TH_T_ID
    now_dts,          // TH_DTS
    st_submitted_id   // TH_ST_ID
)

update
    TRADE
set
    T_ST_ID = st_canceled_id,
    T_DTS = now_dts
where
    T_ID = tr_t_id

insert into
    TRADE_HISTORY (
        TH_T_ID, TH_DTS, TH_ST_ID
    )
values (
    tr_t_id,          // TH_T_ID
    now_dts,          // TH_DTS
    st_canceled_id   // TH_ST_ID
)

} //end of pending_list

/* Remove all pending trades */
delete
from
    TRADE_REQUEST

/* Find submitted trades, change the status to canceled and insert a canceled record
into TRADE_HISTORY*/
declare submit_list for
select
    T_ID
from
    TRADE
where
    T_ID >= trade_id and
    T_ST_ID = st_submitted_id

open submit_list

do until (end_of_submit_list) {
```

Trade-Cleanup_Frame-1 pseudo-code: cancel pending and submitted trades

```
fetch from
    submit_list
into
    t_id

get_current_dts ( now_dts )

/* Mark the trade as canceled, and record the time */
update
    TRADE
set
    T_ST_ID = st_canceled_id
    T_DTS = now_dts
where
    T_ID = t_id

insert into
    TRADE_HISTORY (
        TH_T_ID, TH_DTS, TH_ST_ID
    )
values (
    t_id,           // TH_T_ID
    now_dts,       // TH_DTS
    st_canceled_id // TH_ST_ID
)

} //end of submit_list

commit transaction

}
```

CLAUSE 4 -- DESCRIPTION OF SUT, DRIVER, AND NETWORK

4.1 Overview

TPC-E is a distillation of an abstraction of a “real-world” OLTP environment. In order to understand what TPC-E tests and, as a consequence, what TPC-E does not test, it is necessary to understand the base “real-world” environment (Clause 0.1.1 Description of Real-World OLTP Environment), the abstraction of that base environment (Clause 0.1.2 Functional Component Abstraction of the Real-World OLTP Environment) and the distillation of that abstraction (Clause 0.1.3 Functional Components of the Test Configuration).

4.1.1 Description of the Real-World OLTP Environment

- 4.1.1.1 The figure below shows the “real-world” environment upon which TPC-E is based. Users connect to the brokerage house over a network using a myriad of possible interface devices (e.g. PCs or handheld units). The brokerage house is also able to connect via a network to external businesses (e.g. the stock market exchanges).

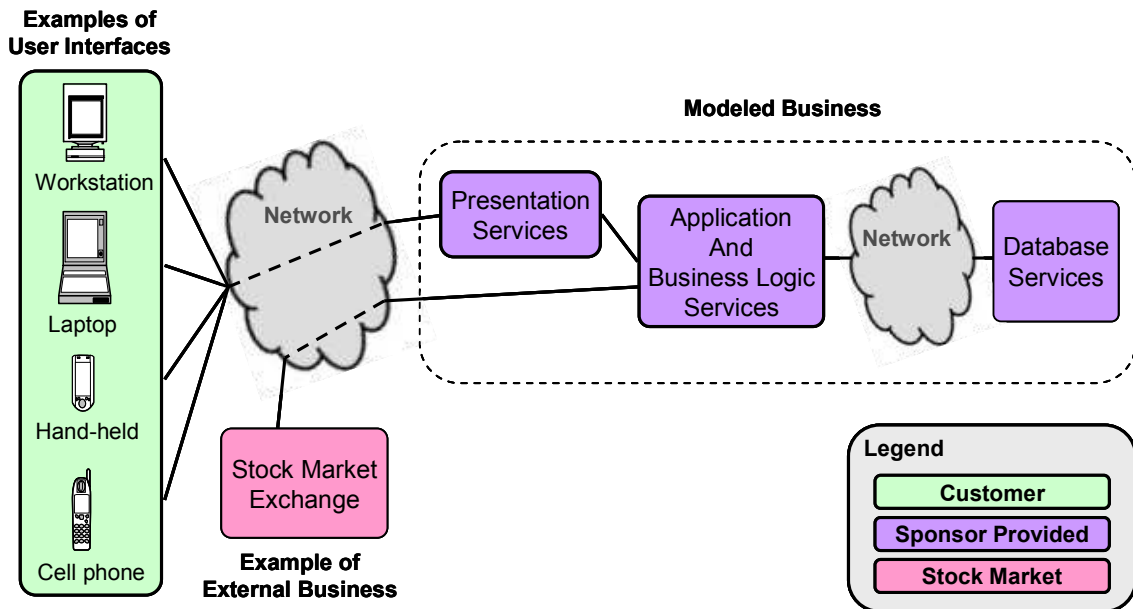


Figure 4.a - Diagram of the Real-World OLTP Environment

4.1.2 Functional Component Abstraction of the Real-World OLTP Environment

4.1.2.1 From the diagram of the real-world OLTP environment, the following diagram of the key functional components can be abstracted.

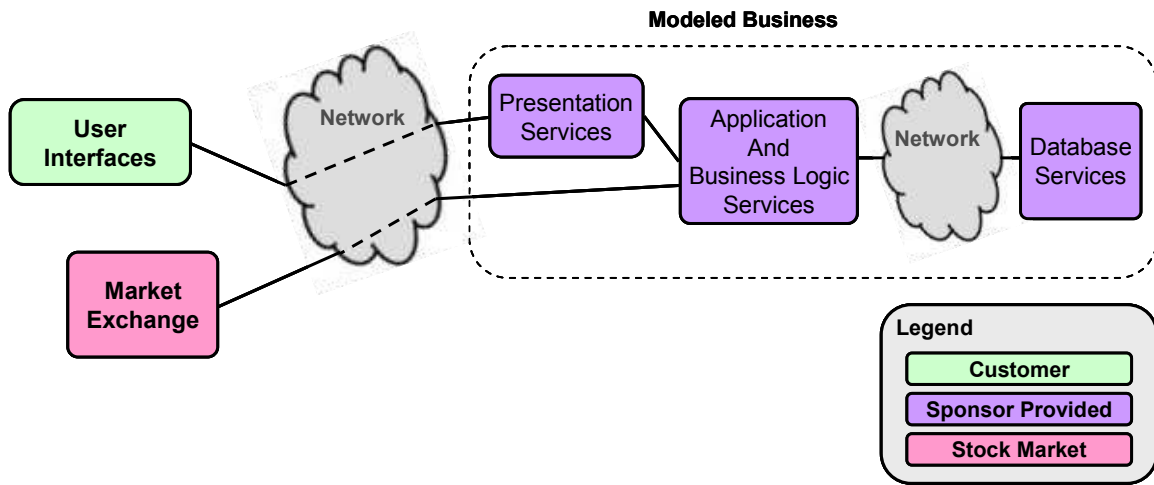


Figure 4.a - Abstraction of the Functional Components in an OLTP Environment

- 4.1.2.2 A user makes use of some device to connect, via the network, to the business's presentation services. As is typical in a Customer-to-Business environment, the presentation layer provides a way for the user to navigate the available services, select the desired operation, enter data and read results. A practical example of this would be a customer using a home PC to connect to a web site to conduct business.
- 4.1.2.3 The brokerage house would likewise connect via a network to an external business, such as the market exchange. As is typical of a Business-to-Business environment, presentation services are not needed. Rather, data can be exchanged directly without the need for a human-readable format.
- 4.1.2.4 Regardless of how the data arrives at the brokerage house, it ultimately will pass through transaction management functions where connection multiplexing/de-multiplexing occurs; routing may also occur here as well as other possible functions. The transaction management layer ensures the data will be delivered to the right business logic code that can perform the requested task.
- 4.1.2.5 A critical step in the business logic occurs when the data is handed off to some function or method implementation for database processing. This method implementation will include **Database Interface** code for packaging up the appropriate data and sending it to the database application logic (e.g. stored SQL procedure) running in the context of the **DBMS**. The database application logic will then use **DBMS** services to perform the necessary tasks, and the results will ultimately be returned "up-stream" as appropriate.

4.1.3 Distillation of Functional Components into the TPC-E Environment

- 4.1.3.1 By design, TPC-E is database-centric. Therefore, even though Presentation Services are an important part of a complete Customer-to-Business solution, they have been distilled out of the TPC-E workload. As a practical matter, Presentation Services often scale out such that a **test sponsor** will configure (replicate) enough servers to run the Presentation Services so they are not a limiting factor for the benchmark. So, to focus on what is being evaluated and to facilitate ease of benchmarking, Presentation Services are not a functional component in the test configuration.
- 4.1.3.2 In the context of the diagram of the functional components of the target system model, the role of the Customer is that of a decision maker and data provider (i.e., deciding what transaction to do and supplying the necessary inputs for that transaction). However, the absence of Presentation Services in TPC-E leads to some simplifications in the test configuration emulation of the User. The decision making and data input generation characteristics of the User are still essential, but characteristics of the User like typing rates and think times are not necessary.
- 4.1.3.3 The role of the User Interface Device (UID) is to accept inputs from the User and send those inputs to the Presentation Services, and accept outputs from the Presentation Services and display those outputs to the User. However, TPC-E does not define or required display layouts (since there are no Presentation Services). Consequently there is no requirement to transmit transaction input and output data in a display format. For example, there is no need to send and receive fully formed HTML pages via HTTP; transaction inputs and outputs may be communicated in a binary format (i.e. by sending C++ data structures over a socket).
- 4.1.3.4 Based on these items and the diagram of the functional components of the target system model, a diagram for the functional components of the test configuration can be derived. Note that the implementation of these functional components implies a combination of hardware and software.

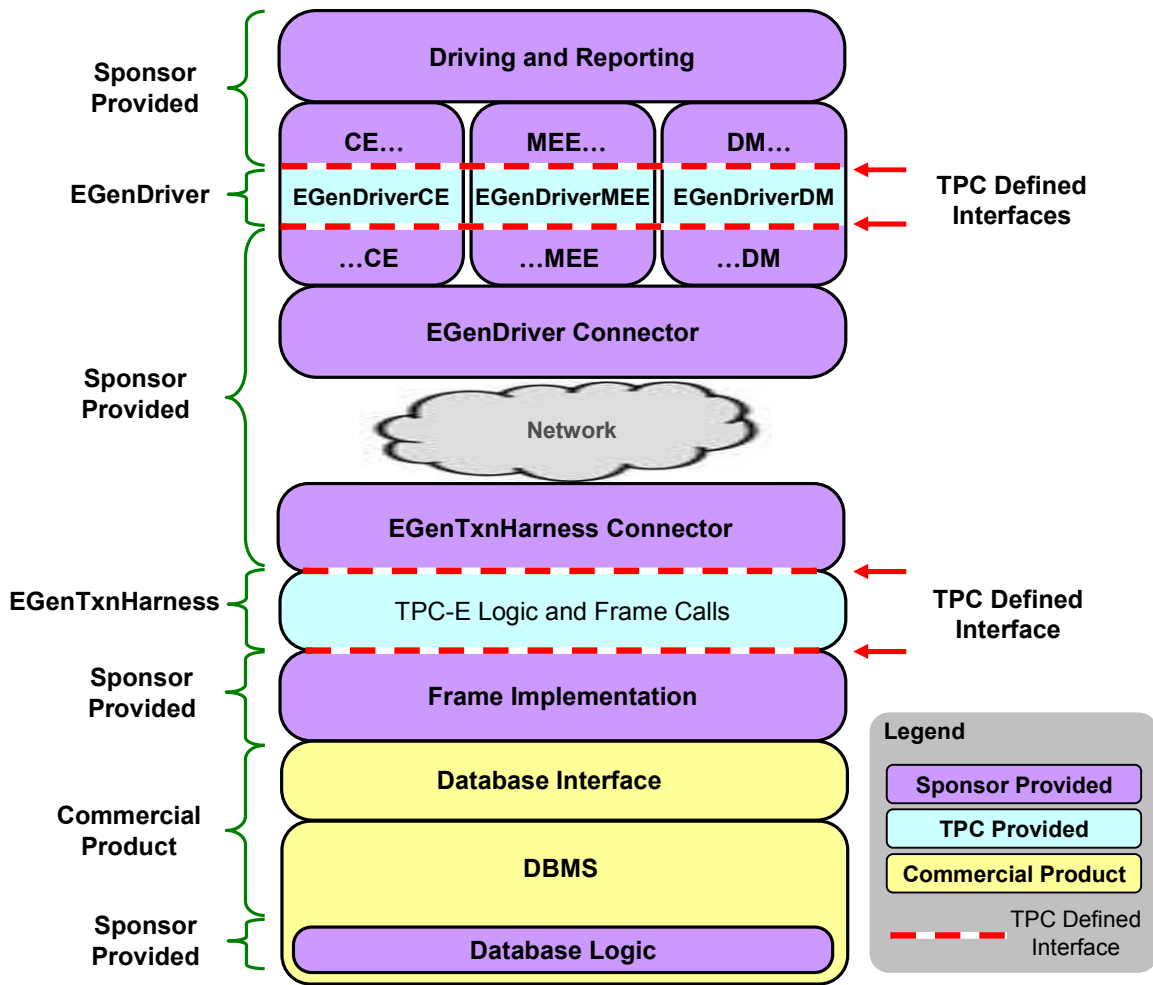


Figure 4.a - Functional Components of the Test Configuration

- **Driving & Reporting** – **Sponsor** provided functionality to set up, administer and execute a test, collect runtime data and generate summary reports. The **sponsor** written code must invoke **EGenDriver** through a TPC defined interface.
- **CE** – **Sponsor** provided functionality to set up, administer and execute the **Customer-Emulator**. The **sponsor** written code must invoke **EGenDriverCE**.
- **MEE** – **Sponsor** provided functionality to set up, administer and execute the **Market-Exchange-Emulator**. The **sponsor** written code must invoke **EGenDriverMEE**.
- **DM** – **Sponsor** provided functionality to set up, administer and execute the Data-Maintenance transaction once a minute. The **sponsor** may also provide functionality to call the Trade-Cleanup **transaction** once prior to the start of the run (see description of **EGeenDriverDM** below). The **sponsor** written code must invoke **EGenDriverDM**.

- **EDriver** – TPC provided C++ source code that implements essential runtime functionality. The use of **EDriver** is mandatory. The following are parts of **EDriver**.
 - **EDriverCE – Customer-Emulator** that provides the required **transaction mix** and user input data generation
 - **EDriverMEE – Market-Exchange-Emulator** that provides the stock market functionality and data generation
 - **EDriverDM – Data-maintenance functionalities** that generates data for and invokes the Data-Maintenance **transaction**. Also, supplies an interface that can be used by the **sponsor** to invoke the Trade-Cleanup **transaction**.
- **EDriver Connector – Sponsor** provided functionality that complies with a TPC defined interface. The **EDriver** Connector is invoked from inside **EDriver** through the interface. The **EDriver** Connector is responsible for sending the **EDriver** generated data to, and receiving the corresponding resultant data back from, the **ETxnHarness** Connector via the **Network**. An example of the hardware and software needed to implement the Connector is:
 - **Sponsor** written code
 - An **Operating System** that provides a socket API and the underlying functionality
 - The hardware system the **Operating System** runs on and the network interface card necessary to connect to the **Network** (the network cable coming out of the NIC to connect it to the **Network** would not be considered part of the Connector but rather part of the **Network**).
- **Network – Sponsor** provided functionality that must support communication through an industry standard communications protocol using a physical means. One outstanding feature of the Connector – Network – Connector communication is that it follows the relevant standards and must imply more than just an application package. It must be possible to have concurrent use of the means by other applications. Physical transport of the data is required and the underlying means of this transport must be capable of operating over arbitrary globally geographic distances. TPC/IP over a local area network is an example of an acceptable Network implementation.
- **ETxnHarness Connector – Sponsor** provided functionality responsible for receiving the data sent from, and sending the appropriate resultant data back to, the **EDriver** Connector via the **Network**. The **ETxnHarness** Connector provides the data to, and accepts the resultant data from, **ETxnHarness** by invoking a TPC defined interface. The **EDriver** Connector example implementation above applies here as well.
- **ETxnHarness** – TPC provided C++ source code that implements essential runtime functionality. **ETxnHarness** invokes the **sponsor's** implementations of the **transaction frames**, providing the necessary inputs and accepting the necessary outputs through a TPC defined interface. The use of **ETxnHarness** is mandatory.
- **Frame Implementation – Sponsor** provided functionality that accepts inputs from, and provides outputs to, **ETxnHarness** through a TPC defined interface. The **Frame Implementation** and all down-stream functional components are responsible for providing the appropriate functionality outlined in the **transaction** profiles (Clause 3.3).

- **Database Interface** – Commercially available product used by the **Frame Implementation** to communicate with the **Database Server**. It is possible that the **Database Interface** may communicate with the **Database Server** over a **Network**, but this is not a requirement.
- **Database Server** – Commercially available product(s). **Sponsor** provided logic may run in the context of the **Database Server** (e.g. a stored SQL procedure). An example of a **Database Server** is:
 - commercially available **DBMS** running on a
 - commercially available **Operating System** running on a
 - commercially available hardware system utilizing
 - commercially available storage
- **Database Logic** – **Sponsor** written **Frame implementation** logic (e.g. stored SQL procedure)

Comment: **EGenDriver Connector** and **EGenTxnHarness Connector** implementations are allowed to perform modifications to the format of the data provided to them if and only if: such modifications are done to support differing characteristics of the underlying transport mechanisms. For example, transporting the data from a big-endian machine to a little-endian machine or from an ASCII environment to an EBCDIC environment will require changes in the data format.

4.2 Driver & System Under Test (SUT) Definitions

The diagram of the functional components of the Test System can be leveraged to provide pictorial definitions of the **Driver**, **SUT**, **Tier A** and **Tier B**.

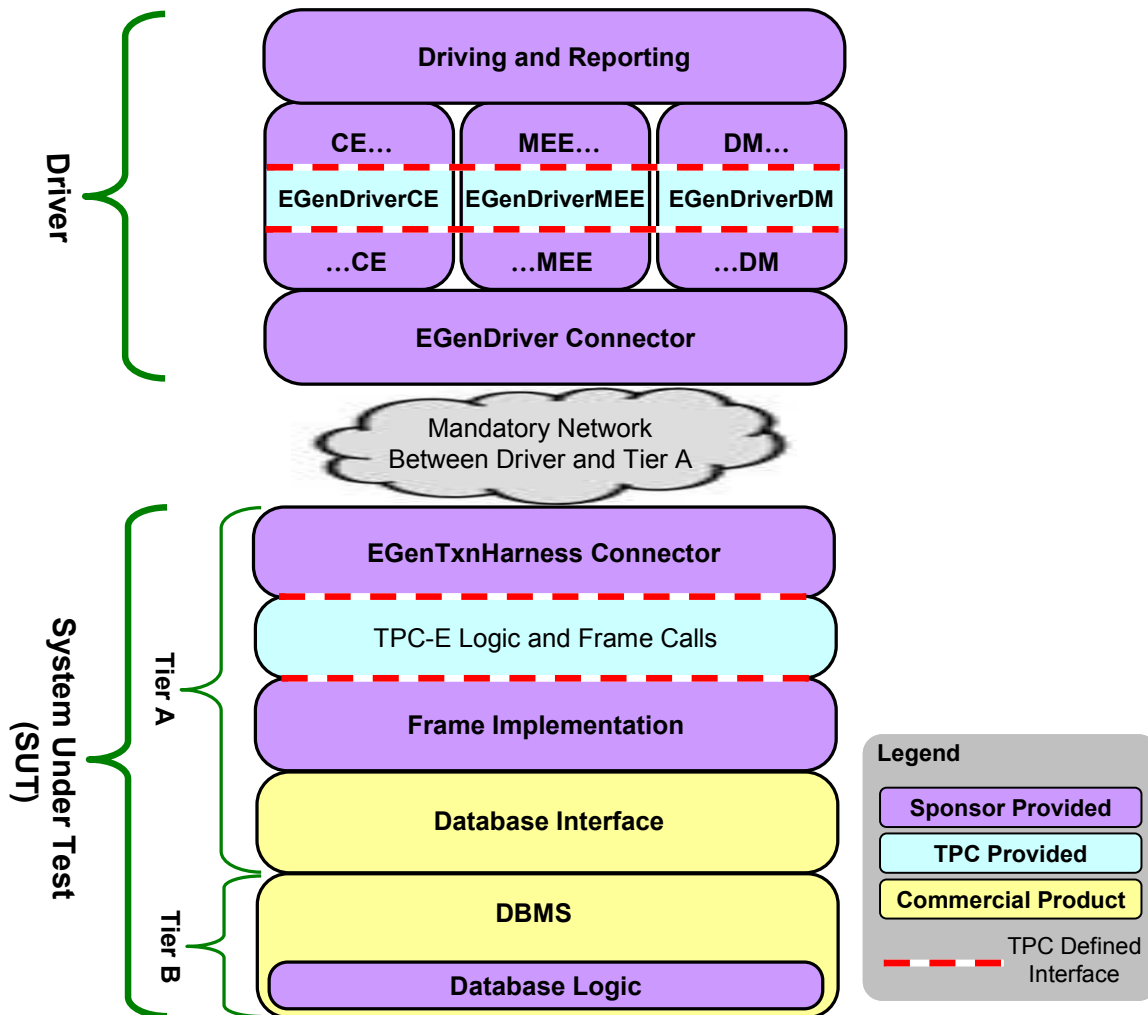


Figure 4.a - Defined Components of the Test Configuration

- **The Driver** – is defined to be all hardware and software needed to implement the Driving & Reporting, **EGenDriver** and up-stream Connector functional components.
- **Tier A** – is defined to be all hardware and software needed to implement the down-stream Connector, **EGenTxnHarness**, **Frame Implementation** and **Database Interface** functional components.
- **Tier B** – is defined to be all hardware and software needed to implement the **Database Server** functional component. This includes data storage media sufficient to satisfy the initial database population requirements of clause 2.6.1 and the **Business Day** growth requirements of clause 6.5.6.3 and clause 6.5.6.4.
- **System Under Test (SUT)** – is defined to be the sum of **Tier A** and **Tier B**.

Comment 1: The use of a **Network** (as defined in Clause 4.1.3) between the **Driver** and **Tier A** is mandatory.

Comment 2: It is possible for the **Driver**, **Tier A** and **Tier B** to share implementation resources. For example, the software portion of the **Driver** implementation and the software portion of **Tier A** may both run on the same underlying hardware.

4.3 Example Test Configuration Implementations

4.3.1 The following figure shows the physical components that could be assembled to implement a hypothetical test configuration.

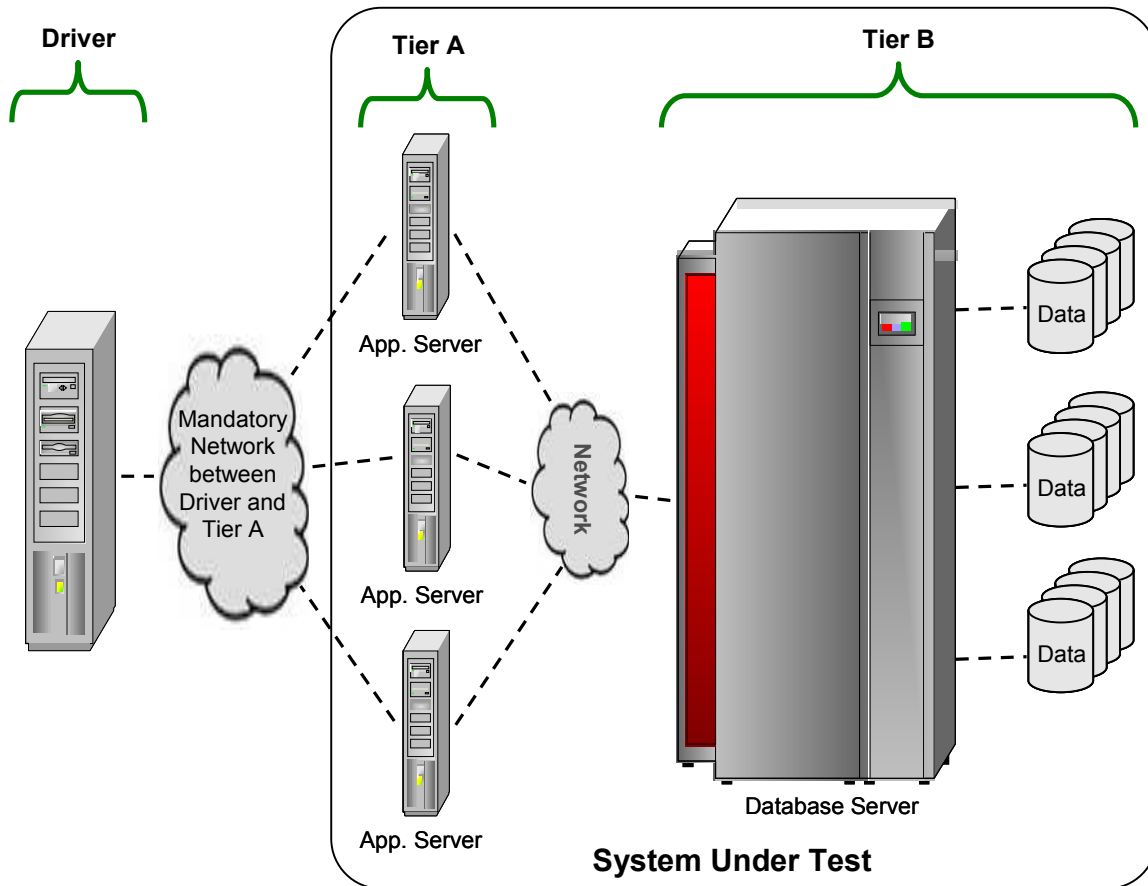


Figure 4.a - Sample Component of Physical Test Configuration

4.3.2 The next few figures show some valid variations on the above test configuration and some of the valid ways for the Driver, Tier A, and Tier B to share common resources.

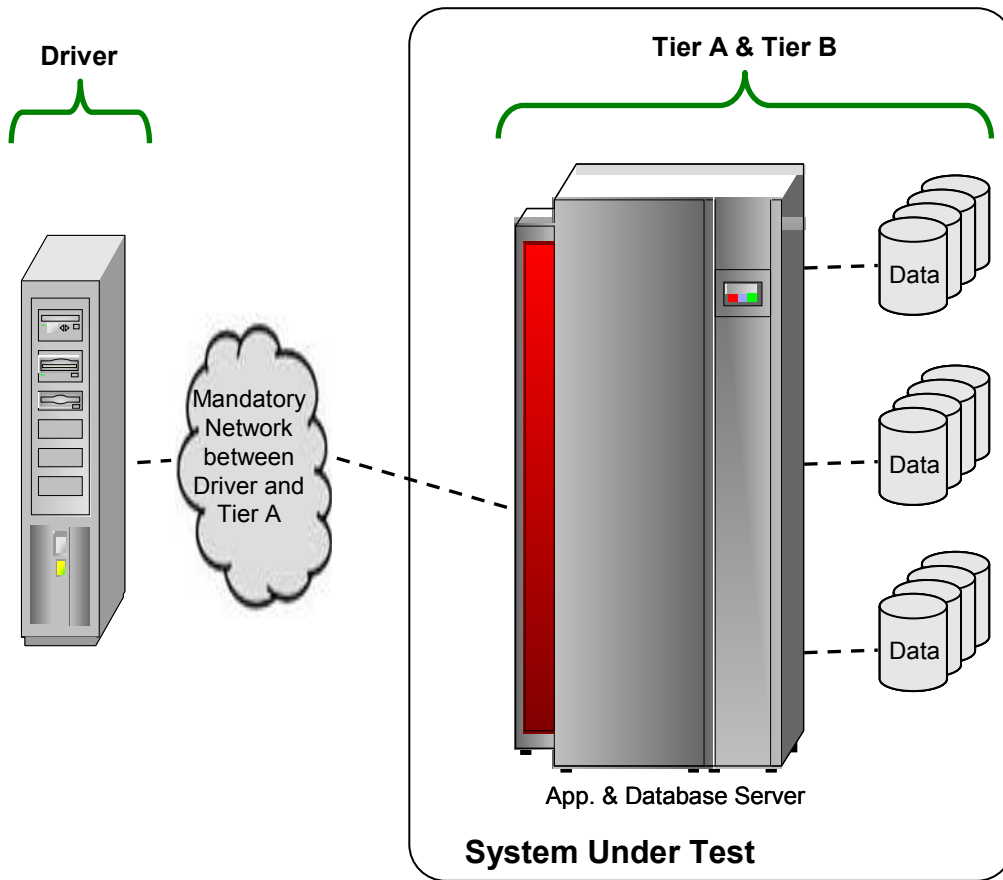


Figure 4.a - Separate Driver with combined Tier A and Tier B

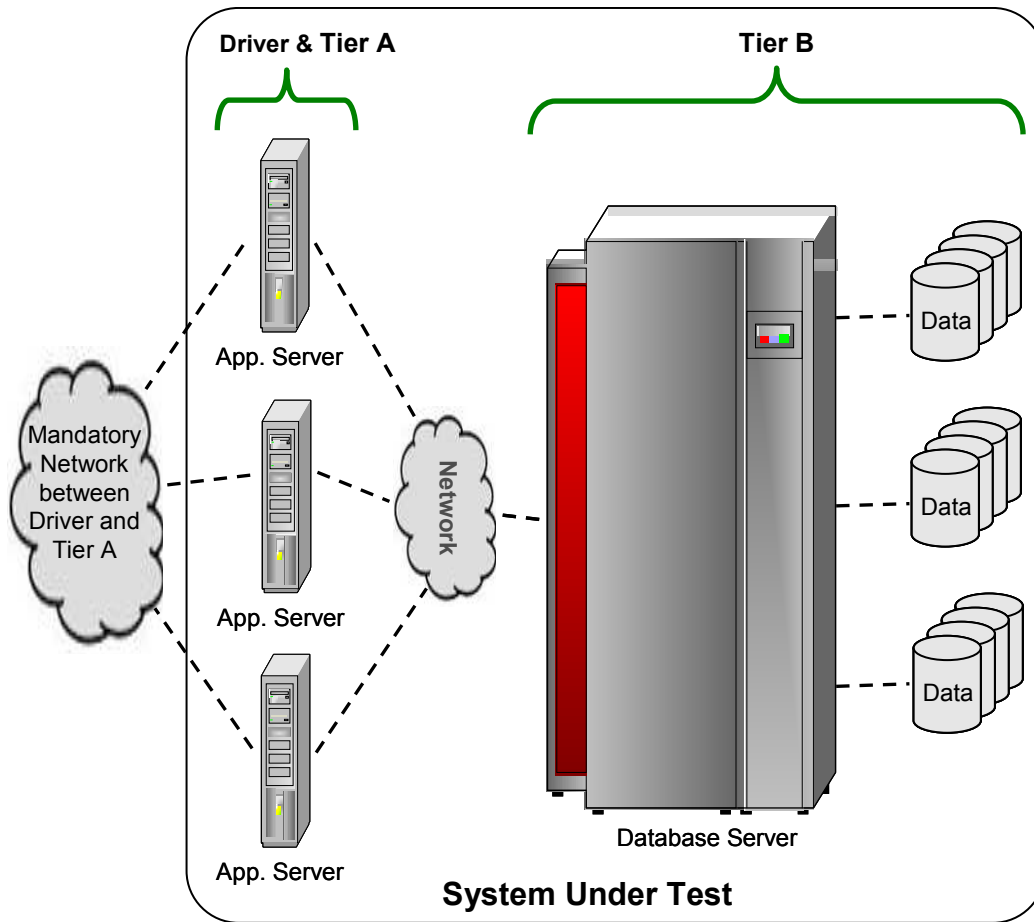


Figure 4.b - Driver and Tier A combined, separate Tier B

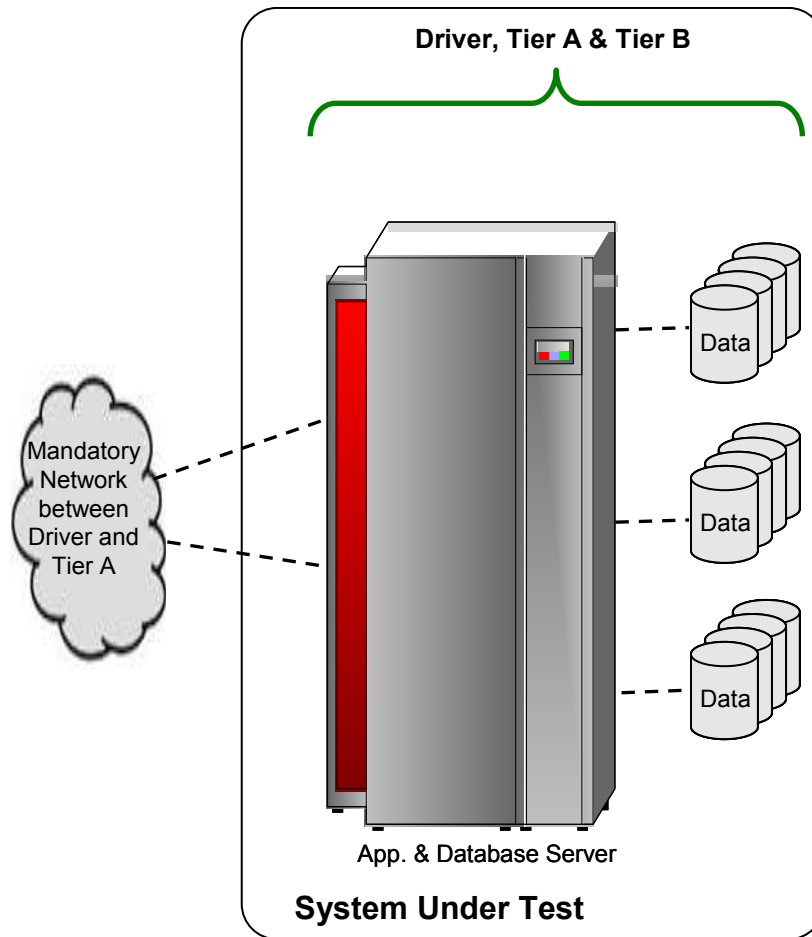


Figure 4.c - Combined Driver, Tier A and Tier B

4.4 Further Requirements for SUT and Driver Implementations

4.4.1 Restrictions on the Driver

- 4.4.1.1 Copies of any part of the tested database or file system or its data structures, indices, etc. are not allowed to be present on the **Driver** during the **Test Run**. The exception is **EGenInputFiles**. **EGenInputFiles** are allowed to be present on the **Driver** during the **Test Run**, even though they contain part of the tested database. **EGenInputFiles** are required by **EGenDriver** to generate **transaction** input parameters.
- 4.4.1.2 Partitioning at the **Driver** level by customer identifier (C_ID) is allowed. If C_ID partitioning is used, the restrictions described in Clause 6.3.2) must be followed.
- 4.4.1.3 The **no-peeking-in-the-packet** rule: Data predicated routing (based on the content of the packet) in the Connector is not allowed. The **sponsor** written logic between the **Driver** and the **EGenTxnHarness** (the Connector code) is not allowed to be predicated on the content of the **transaction** related data being transferred.
- 4.4.1.4 Routing is not allowed in the **Frame Implementation**. This is prohibited through the data access transparency rules in Clause 2.5.

4.4.2 Disclosure of Network Configuration

The **test sponsor** shall describe completely the **Network** configurations of both the tested services and the proposed real (target) services which are being represented.

4.4.3 SUT Implementation Limits on Operator Intervention

Systems must be able to run for at least a **Business Day** without operator intervention.

CLAUSE 5 -- EGEN

5.1 Overview

EGen is a TPC provided software package designed to facilitate the implementation of TPC-E. **EGen** provides:

- consistent data generation independent of the underlying environment
- **transaction** and **frame** flow control management
- project build and makefile templates

This clause covers the constraints and regulations governing the use of **EGen**. For detailed information on **EGen**, what features and functionality it provides and how a **Test Sponsor** is to use those features and functionality see Appendix A.

5.2 EGen Terms

- 5.2.1 **EGen** is a TPC provided software environment that must be used in a **Test Sponsor's** implementation of the TPC-E benchmark. The software environment is logically divided into three packages: **EGenProjectFiles**, **EGenInputFiles**, and **EGenSourceFiles**. The software packages provide functionality: to load the database using **EGenLoader**, and generate transactional data using **EGenDriver**.
- 5.2.2 **EGenProjectFiles** is a set of TPC provided files used to facilitate building the **EGen** packages in a **Test Sponsor's** environments.
- 5.2.3 **EGenInputFiles** is a set of TPC provided text files containing rows of tab-separated data, which are used by various **EGen** packages as “raw” material for data generation.
- 5.2.4 **EGenSourceFiles** is the collection of TPC provided C++ source and header files.
- 5.2.5 **EGenLoader** is a binary executable, generated by using the methods described in **EGenProjectFiles** with source code from **EGenSourceFiles**, including any extensions by a **Test Sponsor** (see Clause 5.7.3). When executed, **EGenLoader** uses **EGenInputFiles** to produce a set of data that represents the initial state of the TPC-E database.
- 5.2.6 **EGenDriver** comprises the following parts:
- **EGenDriverCE** provides the core functionality necessary to implement a **Customer-Emulator**.
 - **EGenDriverMEE** provides the core functionality necessary to implement a **Market-Exchange-Emulator**.
 - **EGenDriverDM** provides the core functionality necessary to implement the **Data-Maintenance Generator**.

EGenDriver provides core transactional functionality (e.g. **transaction mix** and input generation) necessary to implement a **Driver**.

- 5.2.7 **EGenTxnHarness** defines a set of interfaces that are used to control the execution of, and communication of inputs and outputs, of **transactions** and **frames**.
- 5.2.8 **EGenTester** is a binary executable, generated by using methods described in **EGenProjectFiles** with source code from **EGenSourceFiles**. When executed, **EGenTester** uses **Sponsor** provided input to validate that the **Sponsor's Measurement Interval** had compliant Trade-Results per **Load Unit**.
- 5.2.9 **EGenLogger** logs the initial configuration and any re-configuration of **EGenDriver** and **EGenLoader**, and compares current configuration with the TPC-E prescribed defaults.

5.3 Compliant EGen Versions

- 5.3.1 By definition, the following version(s) of **EGen** are compliant with this version of the TPC-E specification.

- **EGen v3.14**

The **EGen** version can be determined by calling the `GetEGenVersion` function provided in `EGen/src/EGenVersion.cpp` file.

- 5.3.2 **EGen** is intended to produce correct data. However, it is the **test sponsor's** responsibility to ensure that the random distribution of all data values, inputs and **transaction mix** frequencies produced by **EGen** is compliant with all constraints documented in the specification (e.g. transaction mix, execution rules, population constraints, etc.).

- 5.3.3 Any existing errors in a compliant version of **EGen**, as provided by the TPC, are deemed to be in compliance with the specification. Therefore, any such errors may not serve as the basis for a compliance challenge.

- 5.3.4 **EGen** is written in ISO C/C++ based on the following standards:

- ISO/IEC 9899:1999 Programming Language C
- ISO/IEC 14882:2003 Programming Language C++

Failure of a C/C++ compiler to properly compile **EGen** because of the compiler's non-conformance with the above standards does not constitute a bug or error in **EGen**.

5.3.5 Addressing Errors in EGen

If a **test sponsor** must correct an error in **EGen** in order to publish a result, the following steps must be performed:

1. The error must be reported to the TPC, following the method described in clause 5.3.6, no later than the time when the result is submitted.
2. The error and the modification used to correct the error must be **reported** in the **FDR**, as described in clause 9.2.4.2.
3. The modification used to correct the error must be reviewed by a **TPC-Certified Auditor**.

Furthermore, the modification and any consequences of the modification may be used as the basis for a non-compliance challenge.

5.3.6 Process for Reporting Issues with EGen

EGen has been tested on a variety of platforms. None-the-less, it is impossible to guarantee that **EGen** is functionally correct in all aspects or will run correctly on all platforms.

5.3.6.1 Portability Issues

If a **sponsor** believes there is a portability issue with **EGen**, the **sponsor** must:

- Document the exact nature of the portability issue.
- Document the exact nature of the proposed fix.
- Contact the TPC Administrator with the above specified documentation (hard or soft copy is acceptable) and clearly state that this is an EGen portability issue. The **sponsor** must provide return contact information (e.g. Name, Address, Phone number, Email).

The TPC will provide an initial response to the **sponsor** within 7 days of receiving notification of the portability issue. This does not guarantee resolution of the issue within 7 days.

If the TPC approves the request, the **sponsor** will be contacted with detailed instructions on how to proceed. Possible methods of resolution include:

- The TPC releasing an updated specification and **EGen** update
- The TPC issuing a formal waiver documenting the allowed changes to **EGen**. In the event a waiver is issued and used by the **sponsor**, certain documentation policies apply (see Clause 9.2.4.2).

If the TPC does not approve the request, the TPC will provide an explanation to the **sponsor** of why the request was not approved. The TPC **may** also provide an alternative solution that would be deemed acceptable by the TPC.

5.3.6.2 Other Issues

For any other issues with **EGen**, the **sponsor** must:

1. Document the exact nature of the issue.
2. Document the exact nature of the proposed fix.
3. Contact the TPC Administrator with the above specified documentation (hard or soft copy is acceptable) and clearly state that this is an EGen issue not related to portability. The **sponsor** must provide return contact information (e.g. Name, Address, Phone number, Email).

5.3.7 Submitting EGen Enhancement Suggestions

As a result of using **EGen**, **test sponsors** may have suggestions for enhancements. To submit a suggestion the **sponsor** must:

1. Document the exact nature of the proposed enhancement
2. Document any proposed implementation for the enhancement
3. Contact the TPC Administrator with the above specified documentation (hard or soft copy is acceptable) and clearly state that this is an **EGen** enhancement suggestion. The **sponsor** must provide return contact information (e.g. Name, Address, Phone number, Email).

The TPC does not guarantee acceptance of any submitted suggestion. However, all constructive suggestions will be reviewed by the TPC, and a response will be provided to the **test sponsor**.

5.4 EGenProjectFiles

The **EGenProjectFiles** provided by the TPC are meant to be used as a template for **Test Sponsors** to develop their **EGen** environments. Use of **EGenProjectFiles** is optional.

5.5 EGenInputFiles

Modification of **EGenInputFiles** provided by the TPC is not permitted.

5.6 EGenSourceFiles

Modification of **EGenSourceFiles** provided by the TPC is not allowed, except as permitted by clause 5.3.

5.7 EGenLoader

5.7.1 The data for a compliant TPC-E database must be generated by **EGenLoader**. The version of **EGenLoader** used must be compliant with the version of the specification the result is being published under, as listed in clause 5.3.

5.7.2 It is presumed that **EGenLoader** produces the correct number of rows for each table. However due to the random nature of the data generated by **EGenLoader**, the data may not be compliant with Clause 2 of this specification. In that event the test database is considered invalid.

5.7.3 If the **test sponsor** extends the loading interface of **EGenLoader** (as described in Appendix A.7.6), all extension code must be reviewed by a **TPC-Certified Auditor**. The use of and audit of extension code must be **reported** in the **Report**. The extension code must be **reported** in the **Supporting Files**.

Comment: The intent of this clause is to ensure that all data generated by **EGenLoader** is not modified, other than to support formatting issues of database data types and sorting of the data, while still allowing **sponsors** the ability to customize **EGenLoader** to specify how the data gets loaded into the database.

5.8 EGenDriver

- 5.8.1 All **EGenLogger** output must be **reported** in the **Supporting Files**. If any **EGenLogger** output contains “NO”, indicating the correct default values were not used, the benchmark result is not compliant.
- 5.8.2 **Sponsors** must use a constructor for each object class (CCE, CMEE, or CDM) that does not have RNGSEED parameter(s).
- 5.8.3 **Sponsors** must ensure that the values provided for the UniqueID parameters to the constructors for each object group (CCE, CMEE or CDM) are unique within each object group.

5.8.4 EGenDriverCE

- 5.8.4.1 A compliant **CE** implementation must use **EGenDriverCE**.
- 5.8.4.2 See Clause 6.3.2 for rules governing the configuration of **EGenDriverCE** when partitioning by C_ID.

5.8.5 EGenDriverMEE

- 5.8.5.1 A compliant **MEE** implementation must use **EGenDriverMEE**.

5.8.6 EGenDriverDM

- 5.8.6.1 A compliant Data-Maintenance **transaction** generator must use **EGenDriverDM**.
- 5.8.6.2 One, and only one, instance of the Data-Maintenance **transaction** generator is required and allowed during runtime.

5.9 EGenTxnHarness

- 5.9.1 A compliant TPC-E implementation must use **EGenTxnHarness**.

5.10 EGenTester

- 5.10.1 A compliant TPC-E implementation must use **EGenTester** to verify a **Measurement Interval** has compliant Trade-Results per **Load Unit**.

CLAUSE 6 -- EXECUTION RULES & METRICS

6.1 Introduction

This clause defines the execution rules and the methods for calculating the benchmark metric.

6.1.1 Definition of Terms

6.1.1.1 The term **Reported** refers to an item that is part of the **FDR** (see Clause 9 -- for detailed requirements).

6.1.1.2 The term **Valid Transaction** refers to any **Transaction** for which input data has been sent in full by the **Driver**, whose processing has been successfully completed on the **SUT** and whose correct output data has been received in full by the **Driver**.

Comment: Any **Transaction** that requires a rollback that runs successfully and produces the correct output is considered a **valid Transaction**.

6.2 Transaction Mix

The TPC-E workload is made up of a set of **transactions** acting against a database following a required **transaction mix**.

Over the **Reported Measurement Interval**, the **Driver** must maintain the mix of **transactions** specified in Clause 6.2.2.1. As a natural result of using the random number selection process, the **Driver** is likely to deviate from the specified **transaction mix** percentages. The maximum deviation allowed by the benchmark is defined in Clause 6.2.2.3.

For the purpose of computing the mix, only **valid transactions** may be counted. **Transaction** errors are not allowed during the **Measurement Interval**.

Comment: **Transaction** errors due to a **transaction** never being able to complete are not allowed.

6.2.1 Mix Control

6.2.1.1 During the **Measurement Interval**, the **Driver** cycles through a process of selecting the next **Transaction**, generating the corresponding input data, requesting that **Transaction** by sending the input data to the **SUT**, waiting for the **Transaction** to execute, receiving the output data from the **SUT**, and measuring its **Response Time**.

6.2.1.2 The **Driver** maintains the required mix of **transactions** by using a 64-bit random number generator. The **Driver** first generates a new random number and then selects a next **Transaction** from the random number according to mix frequencies. Then the **Driver** generates the required inputs for that **Transaction** and submits it to the **SUT** for execution. Once the **Transaction** has completed, the **Driver** optionally waits a **Pacing Delay** amount of time and then loops back to picking the next **Transaction**.

6.2.1.3 The following diagram illustrates the selection and the processing of the **transactions** by the **Driver**.

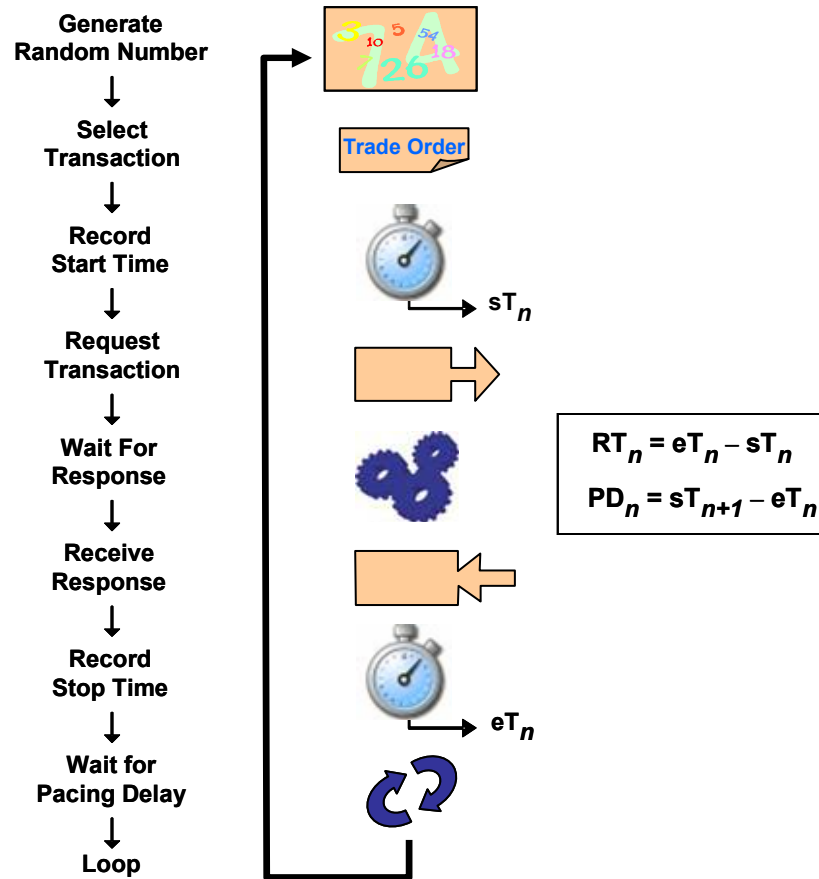


Figure 6.a - Transactions Selection and Processing

6.2.2 Mix Requirements

6.2.2.1 The following table defines the required percentages (P) of mix for each **Transaction**. The valid range of measured mix percentages for each specified Mix % P is $P \pm (0.005 \times P)$. For example, for Trade-Result this would be $10 \pm (0.005 \times 10)$ or a range of 9.95 to 10.05 for a valid measured percentage.

Transaction	Nominal Mix	Comment
Trade-Order	10.1%	~1% of Trade Orders rollback (see Clause 6.3.1.2, rollback is 1 out of each 101 Trade Orders.). 99% of 10.1% is the 10% for Trade Result.
Trade-Result	10%	There is one Trade-Result per Trade-Order completed by the MEE, but ~1% of Trade-Order Transactions rollback at time of initial processing.
Trade-Lookup	8%	
Trade-Update	2%	
Trade-Status	19%	
Broker-Volume	4.9%	
Customer-Position	13%	
Security-Detail	14%	

Market-Feed	1%	Each Market-Feed contains entries for 10 completed Trade-Results.
Market-Watch	18%	
Total	100%	

Comment: The Trade-Result and Market-Feed **transactions** are not generated by the **CE** but arise from asynchronous actions in the **MEE**. The number of completed Trade-Results is one per non-aborted Trade-Order, but pending limit orders are delayed until their trigger price is reached, so mix percentages may vary over short periods of time. The number of Market-Feed **transactions** is determined by the **MEE** to be one for every ten Trade-Results that complete. Again, the mix frequencies will vary over short periods of time, but will closely follow 1/10th of the mix frequency of Trade-Result.

- 6.2.2.2 Computing the mix frequencies actually obtained during the **Measurement Interval** must be done with at least four decimal digits and must be rounded to the nearest three decimal digits when **reported**. For example, 7.2344 must be **reported** as 7.234 and 7.2345 must be **reported** as 7.235
- 6.2.2.3 The computed mix frequency actually obtained for each **transaction** during the **Measurement Interval** must be within the valid range provided in Clause 6.2.2.1 for that **transaction** in order for the **Measurement Interval** to be valid.
- 6.2.2.4 The Data-Maintenance **transaction** is not part of the **Transaction Mix**. Exactly one Data-Maintenance **transaction** must be invoked every sixty seconds. Each **transaction** must successfully complete in 55 seconds or less in order to be considered valid.
- 6.2.2.5 The special Trade-Cleanup **transaction** is not part of the **Transaction Mix**. There are no **Response Time** criteria for the Trade-Cleanup **transaction**, except that the **transaction** must finish before any other type of **transaction** can be executed.

6.3 Input Parameters

The individual **transaction** types have specific input requirements. These **transaction** inputs are generated by the **EGenDriver** code.

6.3.1 Inputs to EGenDriver Code

- 6.3.1.1 The **transaction** inputs are generated by the **EGenDriverCE**, **EGenDriverMEE** and **EGenDriverDM** classes. Each **CE**, **MEE** and **DM** instance must be instantiated using consistent values for some global inputs, and must use the same values used by all **EGenLoader** instances during the initial data generation. Specifically:
 - Each **CE**, **MEE** and **DM** instance must be instantiated in accordance with the specific requirements outlined in Clause 5.8.
 - The contents of **EGenInputFiles** used by all **EGenLoader** instances (when building the database) and by all **CE**, **MEE** and **DM** instances (when running against the database) must be the **EGenInputFiles** for the version of TPC-E that is used in the benchmark publication.

6.3.1.2 Some **transactions** have several input variants for individual **transactions** or mix ratios for individual **frames** within a **transaction**. The required settings for these variations are defined in DriverParamSettings.h. The following list defines these variants and the frequency of their selection by the **EGenDriverCE** code. While random variability is allowed, the **CE Driver** cannot be artificially weighted to generate input data different from the requirements described here. To be valid, the frequency of input data generated during a reported **Measurement Interval** must not exceed the following variability:

Input Parameter	Value	Required Pct	Acceptable Range
Customer-Position			
by_tax_id	True	50%	48% to 52%
get_history	True	50%	48% to 52%
Market-Watch			
Securities chosen by	Watch list	60%	57% to 63%
	Account ID	35%	33% to 37%
	Industry	5%	4.5% to 5.5%
Security-Detail			
access_lob	True	1%	0.9% to 1.1%
Trade-Lookup			
frame_to_execute	1	30%	28.5% to 31.5%
	2	30%	28.5% to 31.5%
	3	30%	28% to 31.5%
	4	10%	9.5% to 10.5%
Trade-Order			
Transactions requested by a third party		10%	9.5% to 10.5%
Security chosen by company name and issue		40%	38% to 42%
type_is_margin	True	8%	7.5% to 8.5%
roll_it_back	True	~1%	0.94% to 1.04% (*)
is_lifo	True	35%	33% to 37%
trade_qty	100	25%	24% to 26%
	200	25%	24% to 26%
	400	25%	24% to 26%
	800	25%	24% to 26%
trade_type	TMB	30%	29.7% to 30.3%
	TMS	30%	29.7% to 30.3%
	TLB	20%	19.8% to 20.2%
	TLS	10%	9.9% to 10.1%
	TSL	10%	9.9% to 10.1%
Trade-Update			
frame_to_execute	1	33%	31% to 35%
	2	33%	31% to 35%

	3	34%	32% to 36%
--	---	-----	------------

(*) **Comment:** The ratio of aborted trades to completed trades is 1/100 or 1%, so the ratio of aborted trades to all trades is 1/101 or only ~1%. The actual expected percentage is closer to 0.99%, which is why the range of acceptable values is 0.94% to 1.04% (not 0.95% to 1.05%), since this range is centered around the expected 0.99% value.

6.3.1.3 The following are **EGen** assigned constants that are used in some of the **transaction** parameters. The constants must not be changed.

Description	Constant	Value	EGen Filename
Broker-Volume			
Minimum number of input broker names	min_broker_list_len	20	TxnHarnessStructs.h
Maximum number of input broker names	max_broker_list_len	40	TxnHarnessStructs.h
Customer-Position			
Maximum customer accounts per customer	max_acct_len	10	TxnHarnessStructs.h
Maximum number of TRADE_HISTORY rows to return	max_hist_len	30	TxnHarnessStructs.h
Market-Feed			
Maximum number of items on the ticker	max_feed_len	20	TxnHarnessStructs.h
Maximum number of triggered limit orders	max_send_len	40	TxnHarnessStructs.h
Security-Detail			
Minimum number of DAILY_MARKET rows to return	iSecurityDetailMinRows	5	MiscConsts.h
Maximum number of DAILY_MARKET rows to return	iSecurityDetailMaxRows	20	MiscConsts.h
Maximum number of DAILY_MARKET rows to return	max_day_len	20	TxnHarnessStructs.h
Maximum number of FINANCIAL rows to return	max_fin_len	20	TxnHarnessStructs.h
Maximum number of NEWS_ITEM rows to return	max_news_len	2	TxnHarnessStructs.h
Maximum number of COMPANY_COMPETITOR rows to return	max_comp_len	3	TxnHarnessStructs.h
Trade-Status			
Maximum number of trade status rows to return	max_trade_status_len	50	TxnHarnessStructs.h

6.3.2 EGenDriverCE partitioning

6.3.2.1 More than one instance of the **CE** may be executing simultaneously. In this case, each instance must keep the proper mix of **CE** transactions. The **CE** instances may be partitioned by **C_ID** (customer identifier). If **C_ID** partitioning is used:

- The **C_ID** sub-range for a given **EGenDriverCE** instance must be a contiguous set of **C_ID**s.
- The minimum **C_ID** of the sub-range must be the starting **C_ID** for a **load unit**.

- The minimum size of the sub-range of C_IDs is 5,000.
- The size of the sub-range of C_IDs must be an integral number of the **load unit** size.
- The size of the sub-range of C_IDs does not have to be the same for each **CE** instance.
- Each **Load Unit** must do approximately the same number of Trade-Results during the **Measurement Interval**. A **Load Unit's** trade rate must be fairly constant during **Steady State**. These requirements must be demonstrated by providing the results of **EGenTester** in the **Supporting Files** (see Clause 6.6.3).

6.3.2.2 In addition, when C_ID partitioning is used, the **EGenDriverCE** code will ensure that:

- C_ID values are chosen from the entire **Configured Customer** range 50% of the time.
- C_ID values are chosen from the provided partitioned customer sub-range 50% of the time.

Example: Assume a database with 60,000 **configured customers**, four (4) **CE** instances. The first two instances will use 20,000 customers and the remaining two instances will use 10,000 customers.

- Instance 1 would be configured to use iMyStartingCustomerId of 1, iMyCustomerCount of 20,000, and iPartitionPercent of 50%.
- Instance 2 would be configured to use iMyStartingCustomerId of 20,001, iMyCustomerCount of 20,000, and iPartitionPercent of 50%.
- Instance 3 would be configured to use iMyStartingCustomerId of 40,001, iMyCustomerCount of 10,000, and iPartitionPercent of 50%.
- Instance 4 would be configured to use iMyStartingCustomerId of 50,001, iMyCustomerCount of 10,000, and iPartitionPercent of 50%.

6.4 Response Time and Pacing Delays

6.4.1 Response Time

6.4.1.1 The **Response Time** (RT) is defined by:

$$RT_n = eT_n - sT_n$$

where:

sT_n and eT_n are measured at the **Driver**;

sT_n = time measured before the first byte of input data of the **transaction** is sent by the **Driver** to the **SUT**; and

eT_n = time measured after the last byte of output data from the **transaction** is received by the **Driver** from the **SUT**.

Comment 1: The resolution of the time stamps used for measuring **Response Time** must be at least 0.01 seconds.

Comment 2: For the purpose of calculating the **Response Time**, only **valid transactions** may be included.

6.4.1.2 During the **Measurement Interval**, at least 90% of each **transaction** type must have a **Response Time** less than or equal to the constraint specified in the table below.

Transaction	90% Response Time Constraint
-------------	------------------------------

Trade-Order	2 sec.
Trade-Result	2 sec.
Trade-Lookup	3 sec.
Trade-Update	3 sec.
Market-Feed	2 sec.
Trade-Status	1 sec.
Customer-Position	3 sec.
Security-Detail	3 sec.
Market-Watch	3 sec.
Broker-Volume	3 sec.

6.4.1.3 The following diagram illustrates where **Response Time**'s are measured for each type of Transaction. Time stamps are taken on the driver.

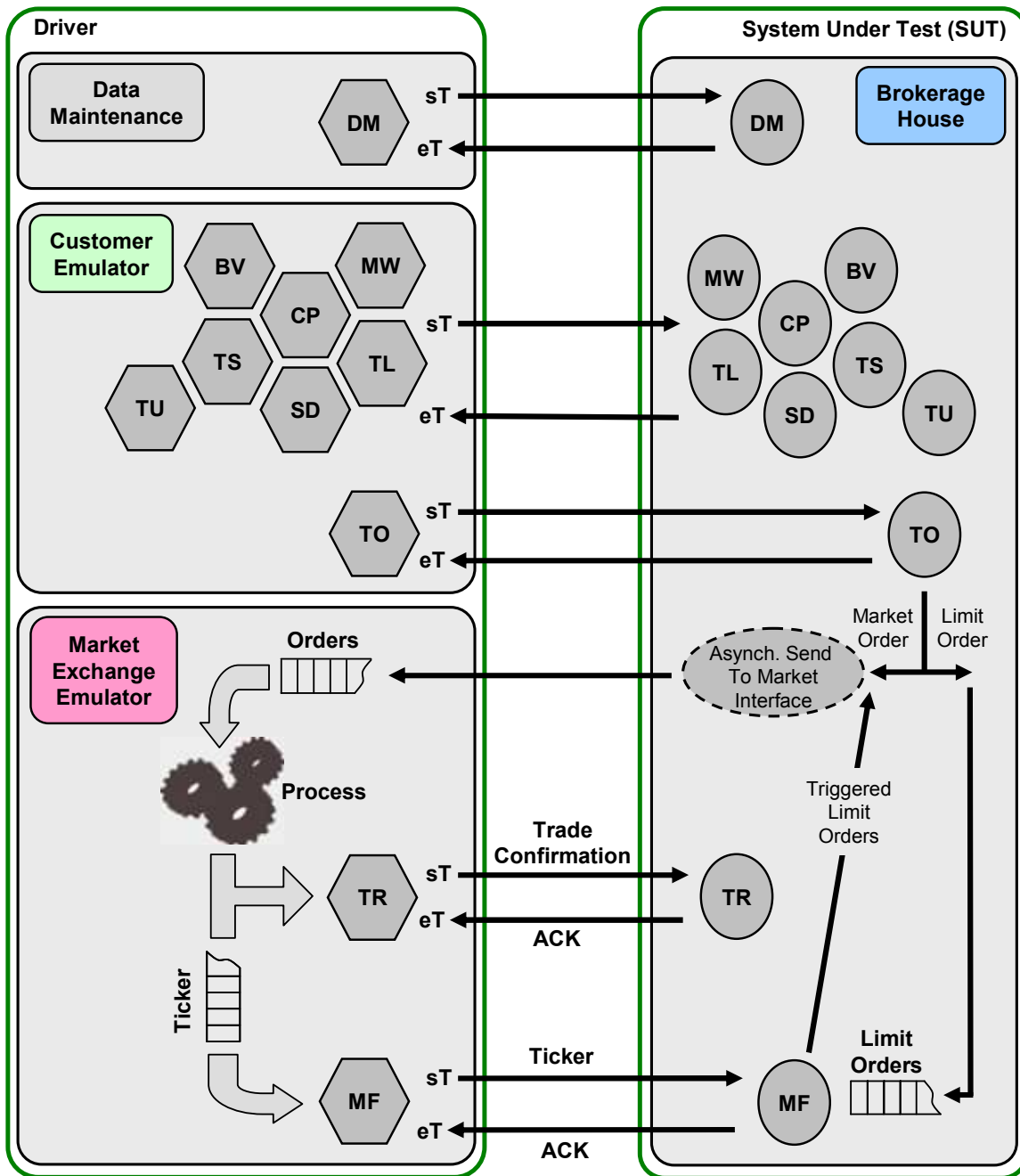


Figure 6.a - Measuring Response Time

- 6.4.1.4 Over the **Measurement Interval**, the average **Response Time** for each type of **transaction** must not be longer than the 90th percentile **Response Time** for that **transaction**.
- 6.4.1.5 The Data-Maintenance **transaction** does not have average and 90th percentile **Response Time** requirements. Instead, each Data-Maintenance **transaction** must successfully complete in 55 seconds or less.

- 6.4.1.6 There are no **Response Time** criteria for the Trade-Cleanup **transaction**. It must complete successfully before a Test Run can start and thus before any other type of **transaction** can be executed.

6.4.2 Pacing Delay

The **Pacing Delay** is defined by:

$$PD_n = sT_{n+1} - eT_n$$

where:

sT_{n+1} and eT_n are measured at the **CE Driver**;

eT_n = time measured after the last byte of output data from the current **Transaction** is received by the **CE Driver** from the **SUT**; and

sT_{n+1} = time measured before the first byte of input data of the next **transaction** is sent by the **CE Driver** to the **SUT**.

- 6.4.2.1 The **Pacing Delay** can be adjusted by the **test sponsor** to control the pacing of **transaction** requests. The **Pacing Delay** must be set to a value that produces a compliant measured tpsE in relation to the configured database scaling (see Clause 6.5.8). The **test sponsor** can set the **Pacing Delay** to zero. The value for **Pacing Delay** is predetermined prior to a **Steady State**. The value remains constant during the **Steady State**.
- 6.4.2.2 It is expected that the **CE Driver** does some amount of processing during the **Pacing Delay** (e.g., generating the next request, logging statistics, etc.) As a result, the amount of time that the **CE Driver** waits after completing all processing and before sending the first byte of the next request to the **SUT** can be reduced to the balance of the **sponsor** defined **Pacing Delay**.

6.5 Test Run

6.5.1 Definition of Terms

- 6.5.1.1 The term **Test Run** refers to the entire period of time during which **Drivers** submit and the **SUT** completes **transactions** other than Trade-Cleanup. A **Test Run** is subdivided into the three consecutive and non-overlapping time periods of **Ramp-up**, **Steady State** and **Ramp-down**.
- 6.5.1.2 The term **Ramp-up** refers to the period of time from the start of the **Test Run** to the start of **Steady State**.
- 6.5.1.3 The term **Steady State** refers to the period of time from the end of the **Ramp-up** to the start of the **Ramp-down**.
- 6.5.1.4 The term **Ramp-down** refers to the period of time from the end of **Steady State** to the end of the **Test Run**.
- 6.5.1.5 The term **Measurement Interval** refers to the period of time during **Steady State** chosen by the **Test Sponsor** to compute the **Reported Throughput Rating**.
- 6.5.1.6 The term **Business Day** refers to a period of eight hours of transaction processing activity.

- 6.5.1.7 The performance of the **SUT** is defined to be **sustainable** if the performance over a given period of time (computed as the average throughput over that time) shows no significant variations.

6.5.2 Database Content

- 6.5.2.1 Prior to a **Test Run**, the database must satisfy Clause 2.4 and all tables must satisfy the minimum cardinality defined in Clause 2.6.1.
- 6.5.2.2 At the start of a **Test Run** the database must not contain any pending or submitted trades. This must be accomplished either by using a database in its initially populated state or by executing the Trade-Cleanup **transaction** prior to the start of the **Test Run**.

6.5.3 Sustainable Performance

- 6.5.3.1 During **Steady State** the throughput of the **SUT** must be **sustainable** for the remainder of a **Business Day** started at the beginning of the **Steady State**.

Some aspects of the benchmark implementation can result in rather insignificant but frequent variations in throughput when computed over somewhat shorter periods of time. To meet the **sustainable** throughput requirement, the cumulative effect of these variations over one **Business Day** must not exceed 2% of the **Reported Throughput**.

Comment 1: This requirement is met when the throughput computed over any period of one hour, sliding over the **Steady State** by increments of ten minutes, varies from the **Reported Throughput** by no more than 2%.

Some aspects of the benchmark implementation can result in rather significant but sporadic variations in throughput when computed over some much shorter periods of time. To meet the **sustainable** throughput requirement, the cumulative effect of these variations over one **Business Day** must not exceed 20% of the **Reported Throughput**.

Comment 2: This requirement is met when the throughput level computed over any period of ten minutes, sliding over the **Steady State** by increments one minute, varies from the **Reported Throughput** by no more than 20%.

- 6.5.3.2 Any resources or components required by the **SUT** to meet the **sustainable** performance requirements must be configured at all time during the **Test Run**.

Comment 1: An example of a non-compliant configuration would be one where the database log file is assigned to a heterogeneous device starting with a high performance drive and overflowing on a slower drive, achieving better performance during the first few hours of **Steady State** than during the remainder of the **Business Day**.

Comment 2: An example of a compliant implementation would be one where the database log file is assigned to a homogeneous device large enough to hold the log over a complete checkpoint cycle and configured to be reused over each subsequent checkpoint cycles, achieving a **sustainable** throughput during **Steady State** and for the remainder of the **Business Day**.

6.5.4 Steady State

6.5.4.1 All work or events that must be performed at regular intervals by the **SUT** during **Steady State** must occur in full at least once between the start of **Steady State** and the start of the **Measurement Interval**.

6.5.4.2 The duration of **Steady State** is set by the **Sponsor** and must be sufficient to:

- Include a compliant **Measurement Interval**,
- Provide sufficient evidence, at the discretion of the **auditor**, that the **sustainable** performance requirement is met,
- Meet all other requirements of the **Test Run**.

6.5.5 Measurement Interval

6.5.5.1 The **Measurement Interval** must be a minimum of two hours and must occur in its entirety during **Steady State**.

6.5.5.2 During the **Measurement Interval**, the database contents (excluding the transaction log) stored on durable media cannot be more than 15 minutes older than any committed state of the database.

6.5.5.3 The **Pacing Delay** may be adjusted by the **sponsor** during **Ramp-up time**. The **Pacing Delay** must be set and fixed at least 15 minutes before the start of the **Measurement Interval**. The **Pacing Delay** must not be changed during the remainder of **Steady State**.

Comment: Changing **Pacing Delay** can have a future effect for up to 15 minutes, due to pending limit orders that are guaranteed to be satisfied within 15 minutes (see clause 1.3.3.4).

6.5.6 Database Growth

6.5.6.1 The resources or components configured on the **SUT** to support executing the **transaction mix** at the **Reported Throughput** during the period of required **sustainable** performance must allow for the resulting increase in the size of the **DBMS** data files (referred to as **Data-Growth**) and the **DBMS** log files (referred to as **Log-Growth**).

6.5.6.2 The total storage space in the **DBMS** data files can be decomposed into the following:

- **Free-Space**, which includes any space allocated to the test database and available for future use. It includes all database storage space not already used to store a database entity (e.g., a row, an index, metadata) or not already used as formatting overhead by the **DBMS**.
- **Growing-Space**, which includes any space used to store existing rows from the **Growing Tables** (i.e., the **CASH_TRANSACTION**, **HOLDING**, **HOLDING_HISTORY**, **SETTLEMENT**, **TRADE**, and **TRADE_HISTORY** tables) and their associated indices and storage overhead. It includes all database storage space that is added to the test database as a result of inserting a new row in the **Growing Tables**, such as row data, index data and other overheads such as index overhead, page overhead, block overhead, and table overhead.
- **Fixed-Space**, which includes any other space used to store static information and indices. It includes all database storage space allocated to the test database which does not qualify as either **Free-Space** or **Growing-Space**.

6.5.6.3 The **Data-Growth** must be computed based on the **Test Run** as follows:

- The measured **Growing-Space** before the **Test Run** is recorded.
- The **Test Run** is executed in full.
- The measured **Growing-Space** after the **Test Run** is recorded.
- The *Data-Space-per-Trade-Result* is computed as the total increase in **Growing-Space** over the **Test Run** divided by the total number of Trade-Result **transactions** completed during the **Test Run**.
- The **Data-Growth** is computed by multiplying the *Data-Space-per-Trade-Result* by the **Reported Throughput** and by the duration of the required **sustainable** performance:
$$\text{Data-Growth} = \text{Data-Space-per-Trade-Result} * \text{tpsE} * \text{Business Day duration in seconds}$$

6.5.6.4 The **Log-Growth** must be computed based on the **Test Run** as follows:

- The space used in the database log file before the **Test Run** is recorded.
- The **Test Run** is executed in full.
- The space used in the database log file after the **Test Run** is recorded.
- The total increase in the used for database log space is divided by the number of Trade-Result **transactions** completed during the **Test Run**, giving the *Log-Space-per-Trade-Result*.
- The *Log-Space-per-Trade-Result* is multiplied by the **Reported Throughput** and by the duration of the required **sustainable** performance to compute the **Log-Growth** as follows:
$$\text{Log-Growth} = \text{Log-Space-per-Trade-Result} * \text{tpsE} * \text{Business Day duration in seconds}$$

6.5.7 Performance & Database Size

- 6.5.7.1 To keep throughput proportional to database size, the **Measured Throughput** must be within a certain range of performance based on the database size.
- 6.5.7.2 The **Nominal Throughput** of the TPC-E benchmark is defined to be 2.00 **Transactions-Per-Second-E** (tpsE) for every 1000 customer rows in the **Configured Customers**.
- 6.5.7.3 Another way of expressing the **Nominal Throughput** is by using a **Scale Factor**. The **Scale Factor (SF)** is the number of customer rows (500) per single **Transaction-Per-Second-E** (tpsE).
- 6.5.7.4 The **Measured Throughput** is computed as the total number of **Valid Trade-Result transactions** within the **Measurement Interval** divided by the duration of the **Measurement Interval** in seconds.
- 6.5.7.5 The number of **load units** configured must be equal to the number of **load units** actually accessed during the **Test Run**.

6.5.8 Throughput Rating

- 6.5.8.1 The performance metric **reported** by TPC-E is the **Throughput Rating**. The name of the metric used for the **Throughput Rating** of the **SUT** is **tpsE**. The value of this metric is based on the **Measured Throughput** and is bound by the **Nominal Throughput** limits of the **SUT** as described in Clause 6.5.7.2.
- 6.5.8.2 If the **Measured Throughput** is between 80% and 100% of the **Nominal Throughput**, then the **Throughput Rating** is set to the **Measured Throughput**. Otherwise, if **Measured Throughput** exceeds the **Nominal Throughput**, but not by more than 2%, the measurement may be used, but the **Throughput Rating** must be set to the **Nominal Throughput**. As a result, the **Measured Throughput** can be as much as 2% greater than the **Throughput Rating**. If the **Measured Throughput** is not within these bounds, then the measurement is invalid and may not be **reported**.

Comment 1: For example, for a database size of 5000 customers, the nominal performance is 10.00 tpsE. A measurement run with throughput between 10.00 tpsE and 10.20 tpsE would be **reported** as 10.00 tpsE. A measurement run with throughput between 8.00 tpsE and 10.00 tpsE would be **reported** as that. A measurement run with throughput lower than 8.00 tpsE, or higher than 10.20 tpsE, is invalid for the database size and must not be **reported**.

Comment 2: To increase the level of throughput that can be **reported**, the number of **Configured Customers** in the database must be increased. To decrease the level of throughput that can be **reported**, the number of **Configured Customers** in the database must be decreased. Either of these two actions requires building a new database.

- 6.5.8.3 The **Reported Throughput Rating** must be measured, rather than interpolated or extrapolated, and rounded down to the nearest two decimal places. For example, suppose 105.748 tpsE is measured during a **Measurement Interval** for which all 90% **Response Time** constraints are met and 117.572 tpsE is measured during a **Measurement Interval** for which some 90% **Response Time** constraints are exceeded. Then the **Reported Throughput** is 105.74 tpsE rather than 105.75 or some interpolated value between 105.748 and 117.572.

6.6 Required Reporting

6.6.1 Test Run Graph

A graph of the **Measured Throughput** versus elapsed wall clock time measured in minutes must be **reported** for the entire **Test Run**. The x-axis represents the elapsed time from the **Test Run** start. The y-axis represents the throughput in tpsE. A plot interval size of 1 minute must be used. The **Ramp-up**, the **Measurement Interval** and **Steady State** must be identified on the graph. The **Test Run Graph** must be **reported** in the **Report**.

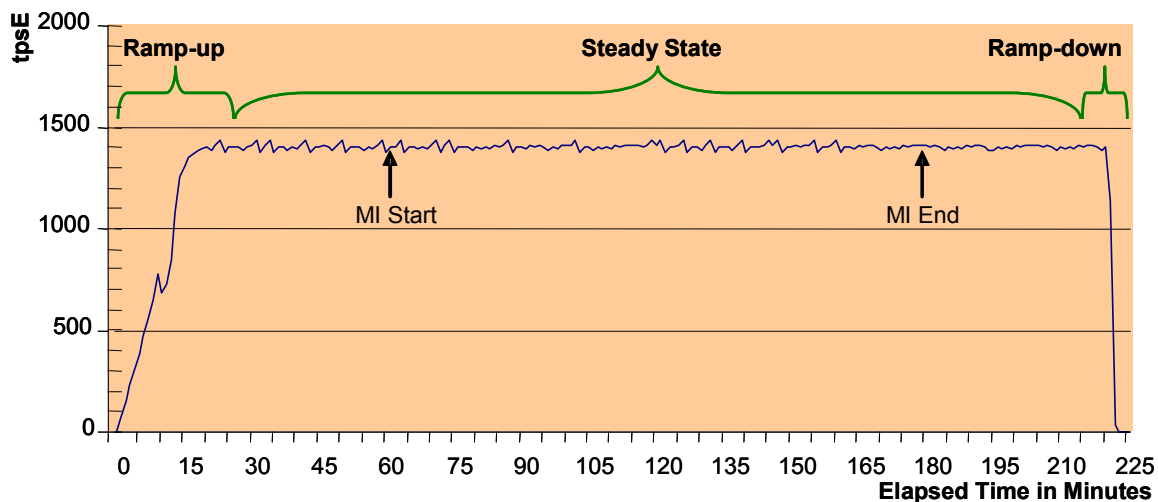


Figure 6.a - Example of the Measured Throughput versus Elapsed Time Graph

6.6.2 Primary Metrics

- 6.6.2.1 To be compliant with the TPC-E standard and the TPC's Fair Use Policies and Guidelines, all public references to TPC-E results for a configuration must include the following components which will be known as the Primary Metrics.
- The TPC-E **Throughput Rating** as expressed in tpsE. This is known as the **Performance Metric**.
 - The TPC-E total 3-year pricing divided by the **Throughput Rating** is price/tpsE. This is also known as the **Price/Performance metric** (See Clause 8 --).
 - The date when all products necessary to achieve the stated performance will be available (stated as a single date on the **Executive Summary Statement**). This is known as the **Availability Date** (See Clause 9.1.2.2).

6.6.3 EGenTester results

6.6.3.1 Each **Load Unit** must do approximately the same number of Trade-Results during the **Measurement Interval**. A **Load Unit's** trade rate must be fairly constant during **Steady State**. These requirements must be demonstrated by providing the results of **EGenTester** in the **Supporting Files**. **EGenTester** is TPC provided code. A **Sponsor** must generate the **EGenTester** binary executable for their environment.

6.6.3.2 When the **Sponsor** runs **EGenTester** they must provide the following inputs:-

- number of **Configured Customers**
- file path to a file containing the Trade-Results per **Load Unit** per minute for each of the **Load Units** used in the measurement and for every minute of the **Measurement Interval**. The file must be a comma separated file with **Load Unit** number followed by minute number of the **Measurement Interval**, followed by the number of Trade-Results for that **Load Unit** during that minute.
- file path to an output file that will be generated

6.6.3.3 The **Sponsor** may run the test several times in parallel if they provide different input and output files for each test instance. All the instances must use the same number of **Configured Customers**. That number must be the number of **Configured Customers** used for the benchmark result. If multiple files are used, the **Load Units** must not be duplicated over the files. The sum of the number of **Load Units** in all the files must equal the number of configured **Load Units** that were used in the benchmark result.

6.6.3.4 **EGenTester** does the following:-

1. Reads the number of **Configured Customers**.
2. Uses **EGen** code to simulate 10,000 runs on a TPC-E database for that number of **Configured Customers**. The program calculates the average number and standard deviation of Trade-Results per **Load Unit** during the **Measurement Interval**. The program calculates the average number and standard deviation of Trade-Results per **Load Unit** during one minute.
3. Reads the input file.
4. For each **Load Unit** and every minute in the input file the program checks that the Trade-Results for that **Load Unit** for each minute are not outside the standard deviation per minute that the program calculated during the simulation.
5. For each **Load Unit** in the input file the program checks that Trade-Results for that **Load Unit** for the whole **Measurement Interval** are not outside the standard deviation for the **Measurement Interval** that it calculated during the simulation.

6.6.3.5 The output from **EGenTester** is written to the output file specified in the input parameters. The output file is a comma separated file.

6.6.3.6 The first record in the output file is:-

- the number of **Configured Customers**
- followed by the average Trade-Results per **Load Unit** during the **Measurement Interval** as calculated by the test program's simulations,

- followed by the standard deviation of Trade-Results per **Load Unit** during the **Measurement Interval** as calculated by the test program's simulations,
- followed by the average Trade-Results per **Load Unit** during a minute as calculated by the test program's simulations,
- followed by the standard deviation of Trade-Results per **Load Unit** during a minute as calculated by the test program's simulations

6.6.3.7 The other records in the output file are:-

- the **Load Unit** number followed by **Load Unit** status.

Valid **Load Unit** status values are "OK", "BAD MEASUREMENT INTERVAL", and "BAD RATE". If any of the records have a **Load Unit** status other than "OK" the benchmark result is not compliant.

CLAUSE 7 -- TRANSACTION AND SYSTEM PROPERTIES (ACID)

7.1 ACID Properties

7.1.1 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.

7.1.2 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 of meeting the ACID requirements. However, for fairness of reporting, only the tests specified here are required and must appear in the **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.

7.1.3 All mechanisms needed to insure full ACID properties must be enabled during the **Test Run**. This applies both to attributes of the database (including tables and auxiliary structures) and to attributes of the **database session(s)** used to execute the ACID and **Test Run**. The attributes of the **session** executing the ACID tests must be the same as those used in the **Test Run**. The term "attributes" includes all database properties and characteristics that can be externally defined. For example, configuration and initialization files, environmental settings, SQL commands and stored procedures, loadable modules and plug-ins. For example, if the **SUT** relies on **Undo/Redo Logs**, then logging must be enabled for all **transactions**, including those that do not include rollback in the **transaction** profile. When this benchmark is implemented on a distributed system, tests must be performed to verify that **transactions** that are processed on two or more nodes; satisfy the ACID properties.

7.1.4 Although the ACID tests do not exercise all **transaction** types of this workload, the ACID properties must be satisfied for all **transactions**.

7.1.5 **Test sponsors** reporting TPC results may perform ACID tests on any one system for which results have been submitted, provided that they use the same software executables (e.g. **Operating System**, database 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 Clause 7.5.3 must be run on all the systems that are measured. All **FDRs** must identify the systems that were used to verify ACID requirements and full details of the ACID tests conducted and results obtained.

7.2 Atomicity Requirements

7.2.1 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.

7.2.2 Atomicity Tests

Perform a market Trade-Order **transaction** with the `roll_it_back` flag set to false. Verify that the appropriate rows have been inserted in the `TRADE` and `TRADE_HISTORY` tables.

Perform a market Trade-Order **transaction** with the `roll_it_back` flag set to true. Verify that no rows have been added to the `TRADE`, and `TRADE_HISTORY` table.

7.3 Consistency Requirements

7.3.1 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.

7.3.1.1 A TPC-E database when first populated by **EGenLoader** must meet these consistency conditions.

7.3.1.2 If data is replicated, as permitted under Clause 2.3.4, each copy must meet the consistency conditions defined above.

7.3.2 Consistency Conditions

Three consistency conditions are defined in the following clauses. Explicit demonstration that the conditions are satisfied is required for all three conditions.

7.3.2.1 Consistency condition 1

Entries in the `BROKER` and `TRADE` tables must satisfy the relationship:

```
B_NUM_TRADES = count(*)
```

For each broker defined by:

(`B_ID` = `CA_B_ID`) and (`CA_ID` = `T_CA_ID`) and (`T_ST_ID` = "CMPT").

7.3.2.2 Consistency condition 2

Entries in the `BROKER` and `TRADE` tables must satisfy the relationship:

```
B_COMM_TOTAL = sum(T_COMM)
```

For each broker defined by:

(`B_ID` = `CA_B_ID`) and (`CA_ID` = `T_CA_ID`) and (`T_ST_ID` = "CMPT").

7.3.2.3 Consistency condition 3

Entries in the `HOLDING_SUMMARY` and `HOLDING` tables must satisfy the relationship:

```
HS_QTY = sum(H_QTY)
```

For each holding summary defined by:

(`HS_CA_ID` = `H_CA_ID`) and (`HS_S_SYMB` = `H_S_SYMB`).

7.3.3 Consistency Tests

The three consistency conditions must be tested after initial database population and after any **Business Recovery** tests.

7.4 Isolation Requirements

7.4.1 Isolation Property Definition

7.4.1.1 Given a **transaction** T1 and a concurrently executing **transaction** T2, the following phenomena (P0 to P3) are defined as they occur in T1.

- **P0 (“Dirty Write”) - Transaction** T2 modifies (or inserts) data element R. Then, before T2 performs a COMMIT, **transaction** T1 starts and is able to modify (or delete) data element R and is subsequently able to perform a COMMIT.

Comment: T2 may execute additional database operations based on the state it left data element R in, potentially compromising the consistency of the data.

- **P1 (“Dirty Read”) - Transaction** T2 modifies (or inserts) data element R. Then, before T2 performs a COMMIT, transaction T1 starts, reads data element R and is able to obtain the state of the data element as changed by T2. Subsequently, T2 is able to perform a ROLLBACK.

Comment: T1 may execute additional database operations based on a state of data element R that has been rolled back and is considered to have never existed, potentially compromising the consistency of the data.

- **P2 (“Non-repeatable Read”) - Transaction** T1 reads data element R. Then, before T1 performs a COMMIT, **transaction** T2 starts, modifies (or deletes) data element R and performs a COMMIT. Subsequently, T1 repeats the read of data element R and is able to obtain the state of the data element as changed by T2.

Comment: Prior to discovering the modified (or deleted) state of data element R, T1 may have executed additional database operations based on a state of data element R that is considered to be no longer correct, potentially compromising the consistency of the data.

- **P3 (“Phantom Read”) - Transaction** T1 reads a set of data elements that satisfy some <search condition>. Then, before T1 performs a COMMIT, **transaction** T2 starts and inserts (or deletes) one or more data elements that satisfy the <search condition> used by T1. Subsequently, T1 repeats the initial read with the same <search condition> and is able to obtain a different set of data elements than the initial set.

Comment: Prior to discovering the larger (or smaller), set of data elements, T1 may have executed additional database operations based on a set of data elements that is considered to no longer match the <search condition>, potentially compromising the consistency of the data.

7.4.1.2 The isolation property of a **transaction** is the level to which it is isolated from the actions of other concurrently executing **transactions**. The table below, arranged from least (L0) to most (L3) restrictive, defines four isolation levels based on which phenomena must not occur.

		Phenomena			
		P0	P1	P2	P3
Isolation Level	L0	Must not occur	Is possible	Is possible	Is possible
	L1	Must not occur	Must not occur	Is possible	Is possible
	L2	Must not occur	Must not occur	Must not occur	Is possible
	L3	Must not occur	Must not occur	Must not occur	Must not occur

- 7.4.1.3 During the **Test Run**, each TPC-E **transaction** must provide a level of isolation from **Arbitrary Transactions** that is at least as restrictive as the level defined in the table below:

TPC-E Transaction	Isolation Level
Trade-Result	L3
Trade-Order Market-Feed Trade-Update	L2
Customer-Position Broker-Volume Trade-Status Market-Watch Trade-Lookup Security-Detail Data-Maintenance	L1

- 7.4.1.4 During the **Test Run** the **SUT** must allow concurrent execution of **Arbitrary Transactions**.
- 7.4.1.5 During the **Test Run**, the data read by each TPC-E **transaction** must be no older than the most recently committed data at the time the **transaction** started.
- 7.4.1.6 Systems that implement **transaction** isolation using a locking and/or versioning scheme must demonstrate compliance with the isolation requirements by executing the tests described in Clause 7.4.2.
- 7.4.1.7 Systems that implement **transaction** isolation using techniques others than a locking and/or versioning scheme may require different techniques to demonstrate compliance with the isolation requirements. It is the responsibility of the **test sponsor**, in collaboration with the **Auditor**, to define those techniques, to implement them, to execute them as a demonstration of compliance with the isolation requirements and to provide sufficient details in the **FDR** to support the assertion that the isolation requirements were met.

7.4.2 Isolation Tests

The following isolation tests are designed to verify that the configuration and implementation of the **System Under Test** provides the **transactions** with the required isolation levels defined in Clause 7.4.1.3.

7.4.2.1 P3 Test in Read-Write

This test demonstrates that a read-write Trade-Result **transaction** is protected against the Phantom phenomenon P3 when executing concurrently with another read-write Trade-Result **transaction**. For the purpose of this test the Trade-Result **transaction** must be instrumented to record HS_QTY after returning from **frame 1** and to be able to pause before starting the execution of **frame 2**.

Using four **sessions**, S1 to S4, the following steps are executed in order:

1. From S1, select a CA_ID. Using an ad hoc read-only transaction, find a S_SYMB that does not have a corresponding row in the HOLDING_SUMMARY table for the selected CA_ID and perform a commit.
2. From S1, request and successfully complete a Trade-Order for the CA_ID and S_SYMB selected in step 1. Record the T_ID assigned to this new trade.

3. From S2, request and successfully complete another Trade-Order for the CA_ID and S_SYMB used in step 2. Record the T_ID assigned to this new trade.
4. From S3, request a Trade-Result for the T_ID from step 2. Pause between frame 1 and frame 2. Record HS_QTY and verify that it is set to zero.
5. From S4, request a Trade-Result for the T_ID from step 3. Complete the execution of frame 1. Record HS_QTY and verify that it is set to zero. Then, proceed with the execution of the remaining frames. The remainder of the transaction may complete successfully, may fail or may be temporarily blocked before completion.
6. From S3, repeat the execution of frame 1 and pause again between frame 1 and frame 2. Record HS_QTY and verify that it is still set to zero.
7. From S3, resume execution by invoking frame 2. Verify the successful completion of the remaining frames.

Comment: This isolation test creates one or more new holdings. Subsequently executing the P2 Test in Read-Write (see Clause 7.4.2.2) for the same selected CA_ID can result in closing the positions created during the execution of this test.

7.4.2.2 P2 Test in Read-Write

This test demonstrates that a read-write Trade-Result **transaction** is protected against the Non-repeatable Read phenomenon P2 when executing concurrently with another read-write Trade-Result **transaction**. For the purpose of this test the Trade-Result **transaction** must be instrumented to record HS_QTY after returning from **frame** 1 and to be able to pause before starting the execution of **frame** 2.

Using four **sessions**, S1 to S4, the following steps are executed in order:

1. From S1, select a CA_ID. Using an ad hoc read-only transaction, find a S_SYMB that has a corresponding row in the HOLDING_SUMMARY table for the selected CA_ID, record the SH_QTY for that holding and perform a commit.
2. From S1, request and successfully complete a Trade-Order for the CA_ID and S_SYMB selected in step 1. Record the T_ID assigned to this new trade.
3. From S2, request and successfully complete another Trade-Order for the CA_ID and S_SYMB used in step 2. Record the T_ID assigned to this new trade.
4. From S3, request a Trade-Result for the T_ID from step 2 and pause between frame 1 and frame 2. Record HS_QTY and verify that it is equal to HS_QTY from step 1.
5. From S4, request a Trade-Result for the T_ID from step 3. Complete the execution of frame 1. Record HS_QTY and verify that it is equal to HS_QTY from step 1. Then, proceed with the execution of the remaining frames. The remainder of the transaction may complete successfully, may fail or may be temporarily blocked from fully executing.
6. From S3, repeat the execution of frame 1 and pause again before invoking frame 2. Record HS_QTY and verify that it is still equal to HS_QTY from step 1.
7. From S3, resume execution by invoking frame 2. Verify the successful completion of the remaining frames.

7.4.2.3 P1 Test in Read-Write

This test demonstrates that a read-write Trade-Result **transaction** is protected against the dirty-read phenomenon P1 when executing concurrently with another read-write Trade-Result **transaction**. For the purpose of this test the Trade-Result **transaction** must be instrumented to record `se_amount` after returning from **frame** 5 and to be able to pause in **frame** 6 just prior to committing.

Using three **sessions**, S1 to S3, the following steps are executed in order:

1. From S1, request a Customer-Position for a selected `CUST_ID`, complete the transaction and record the set of resulting `CA_ID` and `CA_BAL`.
2. From S2, request a Trade-Result for a `TRADE_ID` originating from a `CA_ID` selected from the set recorded in step 1, for a given `T_S_SYMB` and with a `TRADE_IS_CASH` set to "True". After executing frame 5, record `se_amount`, then invoke frame 6 and pause before committing.
3. From S3, request a Trade-Result for the same `CA_ID` but a different `T_S_SYMB` than those used in step 2, and with a `TRADE_IS_CASH` set to "True". The transaction may fail or may be blocked from fully executing. If it reaches the end of frame 5, record `se_amount`, then invoke frame 6. If it reaches the end of frame 6, pause before committing.
4. From S2, proceed with committing and successfully completing the transaction. Record the resulting `CA_BAL`.
5. From S3, depending on how the transaction behaved at the end of step 3:
If it reached the pause in **frame** 6, allow it to proceed and verify that it committed and completed successfully.
If it was blocked before the end of **frame** 5, verify that it was released, completed **frame** 5, recorded `se_amount`, executed **frame** 6, committed and completed successfully.
If it failed and was forced to rollback, repeat the Trade-Result request with the same input parameters. Verify that it executes in full, records `se_amount` at the end of **frame** 5, commits at the end of **frame** 6 and completes successfully.
6. From S3, record the resulting `CA_BAL` and verify that it is equal to `CA_BAL` from step 1 minus the sum of the `se_amount` for the two trades.

7.4.2.4 P1 Test in Read-Only

This test demonstrates that the read-only Customer-Position **transaction** is protected against the dirty-read phenomenon P1 when executing concurrently with the read-write Trade-Result **transaction**. For the purpose of this test the Trade-Result **transaction** must be instrumented to be able to pause in **frame** 6 just prior to committing.

Using four **sessions**, S1 to S4, the following steps are executed in order:

1. From S1, request a Customer-Position for a selected `CUST_ID`, complete the transaction and record the set of resulting `CA_ID` and `CA_BAL`.
2. From S2, request a Trade-Result for a `TRADE_ID` where the associated `T_CA_ID` matches one of the `CA_ID` recorded in the step 1 and `TRADE_IS_CASH` is "True"; then pause in frame 6 before committing.
3. From S3, request a Customer-Position for the `CUST_ID` selected in step 1. The transaction may fail or may be blocked from fully executing.

4. From S2, proceed with committing and successfully completing the transaction. Record the resulting CA_BAL.
5. From S3, depending on how the transaction behaved at the end of step 3:
 - a. If it completed, record the set of resulting CA_ID and CA_BAL and verify that the CA_BAL for the CA_ID used in step 2 is unchanged from step 1.
 - b. If it was blocked, verify that it has now completed, record the set of resulting CA_ID and CA_BAL and verify that the CA_BAL for the CA_ID used in step 2 matches the CA_BAL from step 4.
 - c. If it failed, proceed to the next step.
6. From S4, request a Customer-Position for the CUST_ID selected in step 1, complete the transaction, record the set of resulting CA_ID and CA_BAL and verify that the CA_BAL for the CA_ID used in step 2 has changed from steps 1 and reflects the amount of the trade completed in step 4.

7.5 Durability and Data Accessibility Requirements

The intent of this clause is to verify the **SUT**'s ability to provide access to all committed **transactions** and maintain consistency of the database despite certain types of failure. Some types of failure are **non-catastrophic** and data access is maintained throughout the failure and recovery process. Other types of failures can be **catastrophic** in nature, and while data access is temporarily lost, the **SUT** is able to preserve the state of the database and restore access to the data.

7.5.1 Definition of Terms

- **Catastrophic:** a type of single failure where processing is interrupted without any foreknowledge given to the **SUT**. Subsequent to this interruption, all contexts for all active applications are lost and all main memory in the **SUT** is cleared.
- **Durability:** the ability to preserve the effects of committed transactions and ensure database consistency after recovery from any one of the failures listed in Clauses 7.5.2 and 7.5.3.
- **Accessibility:** the ability to retrieve the contents of database tables by the database after a failure in Clause 7.5.2.

Comment: No system provides complete **durability** (i.e., durability under all possible types of failures). The specific set of single failures addressed in Clauses 7.5.2 and 7.5.3 is deemed sufficiently significant to justify demonstration of durability across such failures.

- **Business Recovery** is the process of recovering from a **catastrophic** system failure and reaching a point where the business meets certain operational criteria.

Comment 1: For the purpose of the TPC-E benchmark, the start of **Business Recovery** is the time at which database files are first accessed by a process that has knowledge of the contents of the files and has the intent to recover the database or issue **transactions** against the database.

Comment 2: Access to files by **operating system** processes that check for integrity of file systems or volumes to repair damaged data structures does not constitute the start of **Business Recovery**.

Comment 3: The end of **Business Recovery** is deemed to be once the **SUT** has operated at or above 95% of **reported throughput** for 20 minutes.

- **Business Recovery Time** is the elapsed period of time between start of **Business Recovery** and end of **Business Recovery**.
- A **durable medium** is a data storage medium that is either:
 - An inherently non-volatile medium (e.g., magnetic disk, magnetic tape, optical disk, etc.) or
 - 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.

Comment 1: A configured and priced Uninterruptible Power Supply (UPS) is not considered external power.

Comment 2: 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 requirements 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).

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

- A **Measured throughput period** is a period of time in which the following criteria are satisfied:
 - be performed with a fully scaled database and **driver** load
 - be in **Steady State**
 - satisfy **Response Time** constraints listed in Clause 6.4.1.2
 - satisfy the **Transaction Mix** requirements listed in Clause 6.2.2
 - be at or above 95% of the **Reported Throughput** with no errors
 - match all **driver** and **SUT** configuration settings used during the **Reported Throughput Measurement Interval**.

7.5.2 Data Accessibility

The **SUT** must maintain database access to data on durable media during and after the permanent irrecoverable failure of any single durable medium containing database tables, recovery log data, or database metadata. The **test sponsor** must also restore the durable medium environment to its pre-failure condition, while maintaining database access to the data on durable media.

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 an **Undo/Redo Log** (after image), 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.

7.5.2.1 Redundancy Levels

The redundancy levels refer to the level of guarantee for data access given a single failure among the data storage components. The **test sponsor** must report the Redundancy Level and describe the test(s) used to demonstrate compliance in the **Report**.

- **Redundancy Level One:** Guarantees access to the data on durable media when a single durable media failure occurs.
- **Redundancy Level Two:** Includes **Redundancy Level One** and guarantees access to the data on durable media when a single failure in the processor/cache/controller of the durable media enclosure occurs.

Comment: The intent is to encourage redundancy in the disk storage enclosure. This can be demonstrated by removing a controller card in the storage enclosure.

- **Redundancy Level Three:** Includes **Redundancy Level Two** and guarantees access to the data on durable media when a single failure of a storage controller/interface card in the central processing complex occurs.

Comment: The intent is to encourage multi-pathing at the **Operating System** or storage system level. This can be demonstrated by removing all cables from a controller/interface card on the system bus.

7.5.2.2 Durability test procedure for data accessibility

1. Determine the current number of completed trades in the database by running:
2. *select count(*) as count1 from SETTLEMENT*
3. Start submitting **transactions** and ramp up to a **Measured throughput period** (as defined in Clause 7.5.1) of at least 5 minutes.
Comment: After entering the **Measured throughput period**, no **driver** or SUT configuration changes are permitted until the conclusion of step 6.
4. Induce the failure described for the redundancy level being demonstrated.
5. Begin the necessary recovery process.
6. Continue running the **driver** for 20 minutes.
7. Terminate the run gracefully from the **driver**.
8. Retrieve the new number of completed trades in the database by running:
select count() as count2 from SETTLEMENT*
9. Compare the number of executed Trade-Result **transactions** on the **driver** to (count2 - count1). Verify that (count2 - count1) is equal to the number of successful Trade-Result **transaction** records in the **driver** log file.
10. Allow recovery process to complete as needed.
11. Verify consistency conditions as specified in Clause 7.3.1.1

7.5.2.3 Reported Metrics for data accessibility

A description of the data **accessibility** tests run and the redundancy level they were demonstrating must be **reported** in the **FDR**.

7.5.2.4 Data Accessibility Graph

A graph of the measured throughput versus elapsed time must be **reported** in the **Report** for the run portions of the data **accessibility** tests, prepared in accordance with the following conventions:

- The x-axis represents the elapsed time for the runs described in Clause 7.5.2.2, steps 2 through 6
- The y-axis represents the throughput in tpsE
- A plot interval size of 1 minute must be used

Comment: The intent is to show how throughput is affected during recovery.

7.5.3 Business Recovery

Some failures can be **catastrophic** in nature, and data access is temporarily lost. The **SUT** is able to preserve the state of the database and restore access to the data.

7.5.4 List of Single Failures

- Instantaneous interruption (system crash/system hang) in processing that requires system reboot to recover.

Comment 1: This implies abnormal system shutdown that 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 (see Clause 8.3.1.3). 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.

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

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

- Power Failure

Comment 1: 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.

Comment 2: 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.

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.

- 7.5.4.1 The above 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-E.

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 all three points of failure described above).

Comment 3: Roll-forward recovery from an archive database copy (e.g., a copy taken prior to the run) using **Undo/Redo Log** data is not acceptable as the recovery mechanism in the case of failures listed in Clause 7.5.4. Note that "checkpoints", "control points", "consistency points", etc. of the database taken during a run are not considered to be archives.

7.5.5 Durability test procedure when Business Recovery is needed

1. Determine the current number of completed trades in the database by running:
2. *select count(*) as count1 from SETTLEMENT.*
3. Start submitting **transactions** and ramp up to a **Measured throughput period** (as defined in Clause 7.5.1) of at least 20 minutes.
4. Induce one of the failures selected from the list in clause 7.5.4.
5. Stop the **driver**.
6. If necessary, restart the **SUT** (may necessitate a full reboot).
7. Note the time when **Business Recovery** starts, either automatically or manually by an operator.
8. Once the **SUT** will accept **transactions**, start submitting **transactions** and ramp up to a **Measured throughput period** (as defined in Clause 7.5.1) of at least 20 minutes.

Comment: The Trade-Cleanup **transaction** must not be executed during the **Business Recovery** test following a **catastrophic** failure.
9. Note this time as the end of **Business Recovery**.
10. Terminate the **driver** gracefully.
11. Verify that no errors were reported by the **driver** during steps 8 through 10. The intent is to ensure that an end-user would not see any adverse effects (aside from availability of the

application and potentially reduced performance) due to the **SUT** failure and subsequent **Business Recovery**.

12. Retrieve the new number of completed trades in the database by running:

13. *select count(*) as count2 from SETTLEMENT*

14. Compare the number of executed Trade-Result **transactions** on the **driver** to (count2 - count1). Verify that (count2 - count1) is greater or equal to the aggregate number of successful Trade-Result **transaction** records in the **driver** log file for the runs performed in step 2 and step 8. If there is an inequality, the SETTLEMENT table must contain additional records and the difference must be less than or equal to the number of **CE** instances.

Comment: This difference must be due only to **transactions** which were committed on the **System Under Test**, but for which the output data was not returned to the **driver** before the failure.

15. Verify consistency conditions as specified in Clause 7.3.1.1.

7.5.6 Required Reporting for Business Recovery tests

7.5.6.1 Reported Metrics

The **Business Recovery Time** (the elapsed time between steps 6 through step 8 of Clause 7.5.5) must be **reported** on the **Executive Summary Statement** and in the **Report**. If the failures described in Clause 7.5.4 were not combined into one **durability** test (usually powering off the **Database Server** during the run), then the **Business Recovery Time** for the failure described for instantaneous interruption is the **Business Recovery Time** that must be **reported** in the **Executive Summary Statement**. All the **Business Recovery Times** for each test requiring **Business Recovery** must be **reported** in the **Report**.

7.5.6.2 Business Recovery Time Graph

A graph of the measured throughput versus elapsed time must be **reported** in the **Report** for the run portions of the **Business Recovery** tests, prepared in accordance with the following conventions:

- The x-axis represents the maximum of the elapsed times for the two runs described in Clause 7.5.5 steps 2 and 7
- The y-axis represents the throughput in tpsE
- A plot interval size of 1 minute must be used
- The y-axis data for both runs is to be overlaid on a single graph, with the end times of each run clearly marked
- For graphing purposes, time 0 is defined as follows:
 - For the run outlined in 7.5.5 step 2, time 0 is defined as the point in time where the first **transaction** is issued to the database
 - For the run outlined in 7.5.5 step 7, time 0 is defined as the point in time where **Business Recovery** begins
- For graphing purposes, the end of the run is defined as follows:
 - For the run outlined in 7.5.5 step 2, the end of the run is the time at which the failure is induced (see 7.5.5 step 3)
 - For the run outlined in 7.5.5 step 7, the end of the run is the time at which the **Business Recovery** has ended successfully (see 7.5.5 step 8)

CLAUSE 8 -- PRICING

Rules for pricing the **Priced Configuration** and associated software and maintenance are included in the current revision of version 1 of the TPC Pricing Specification, located at www.tpc.org.

The following requirements are intended to supplement the TPC Pricing Specification:

8.1 **Priced Configuration**

The system to be priced is the aggregation of the **SUT**, **Network** and any additional component that would be required to achieve the **reported** performance level. Calculation of the priced system consists of:

- Price of the **SUT** as tested and defined in Clause 4 -- .
- If the **SUT** does not already include sufficient **Free-Space** to satisfy the **60-Day-Space** requirement any additional storage and associated infrastructure needed to satisfy the **60-Day-Space** requirement must be included.
- Price of additional products that are required for the operation, administration or maintenance of the priced system.
- Price of additional products required for application development.

Comment: Any component, for example a Network Interface Card (NIC), must be included in the price of the **SUT** if it draws resources for its own operation from the **SUT**. This includes, but is not limited to, power and cooling resources. In addition, if the component performs any of the function defined in the TPC-E specification it must be priced regardless of where it draws its resources.

8.2 **On-line Storage Requirement**

8.2.1 **Continuous Operation Requirement**

Within the priced system, there must be sufficient on-line storage to support

- The initial database population (see Clause 2.6) and all indices present during the **Test Run**.
- Any expanding system files and the durable database population resulting from executing the TPC-E **transaction mix** for a **Business Day** at the **reported** tpsE. This means that the test database must have an additional **Business Day's** worth of data, index and log disks online. The methods to calculate the **Data-Growth** and the **Log-Growth** are described in Clauses 6.5.6.3 and 6.5.6.4. Storage is considered on-line, if any record can be accessed randomly and updated within 1 second. On-line storage may include magnetic disks, optical disks, or any combination of these, provided that the above mentioned access criteria is met.

Comment 1: The intent of this clause is to consider as on-line any storage device capable of providing an access time to data, for random read or update, of one second or less, even if this access time requires the creation of a logical access path not present in the tested database. For example, a disk based sequential file might require the creation of an index to satisfy the access time requirement.

Comment 2: The requirement to support a **Business Day** of recovery log data can be met with storage on any durable media if all data required for recovery from failures listed in Clause 7.5 are on-line.

8.2.2 60-Day Data Space

Storage must be priced for sufficient space to store and maintain the data and indices generated during a period of 60 **Business Days** at the **Reported Throughput Rating** called the **60-day period**. This storage must be configurable but does not need to be on-line during the **Measurement Interval**.

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

$$\text{60-Day-Space} = \text{Initial Database Size} + (60 * \text{Data-Growth})$$

The calculation of **Data-Growth** is described in clause 6.5.6.1.

Comment: If the measured configuration is configured with more storage than the **60-Day-Space** requires, all of the configured storage must be priced. The amount of storage priced cannot be less than what was configured during the measurement. But the amount of storage priced may be greater than what was configured during the measurement, either because not all storage required for the **60-day period** was on-line or because not all storage included in the priced system is needed to satisfy all of the space requirements.

8.2.3 Archive Operation Requirement

TPC-E has no requirements for pricing additional archive requirements.

8.2.4 Back-up Storage Requirements

TPC-E has no requirements for on-line back-up data in the **Priced Configuration**.

8.3 TPC-E Specific Pricing Requirements

8.3.1 Additional Operational Components

- 8.3.1.1 Additional products that might be included on a customer installed configuration, such as operator consoles and magnetic tape drives, are also to be included in the priced system if explicitly required for the operation, administration, or maintenance, of the priced system.
- 8.3.1.2 Copies of the software, on appropriate media, and a software load device, if required for initial load or maintenance updates, must be included.
- 8.3.1.3 The price of an Uninterruptible Power Supply, specifically contributing to a durability solution, must be included (see Clause 7.5.4).
- 8.3.1.4 The price of all components, including cables, used to interconnect components of the **SUT** must be included.

8.3.2 Additional Software

- 8.3.2.1 The price must include the software licenses necessary to create, compile, link, and execute this benchmark application as well as all run-time licenses required to execute on host system(s), client system(s) and connected workstation(s) if used.
- 8.3.2.2 In the event the application program is developed on a system other than the **SUT**, the price of that system and any compilers and other software used must also be included as part of the priced system.

8.4 Component Substitution

- 8.4.1 Hardware and software substitution(s) of the server or back-end system or the host system, **Operating System** or **Database Management System** of the **Tier B** is not allowed under any circumstances.
- 8.4.2 Hardware and software used in the configuration of **Tier A** may be substituted.
- 8.4.3 Hardware and software of secondary components in the **SUT** such as disks, disk enclosures, network interface cards, routers, bridges, repeaters and similar items may be substituted.

Comment: The component substitution will be open to challenge for a period of 60 days.

8.5 Required Reporting

- 8.5.1 Two metrics will be **reported** with regard to pricing. The first is the total 3-year pricing as described in the current revision of version 1 of the TPC Pricing specification. The second is the total 3-year pricing divided by the **Reported Throughput** (tpsE), as defined in Clause 6.5.8.
- 8.5.2 The 3-year pricing metric must be fully **reported** in the basic monetary unit of the local currency unit rounded up and the **price/performance metric** must be **reported** to a minimum precision of three significant digits rounded up. Neither metric may be interpolated or extrapolated. For example, if the total price is \$ 5,734,417.89 USD and the **Reported Throughput** is 105 tpsE, then the 3-year pricing is \$ 5,734,418 USD and the price/performance is \$ 54,700 USD per tpsE (5,734,418/105).

CLAUSE 9 -- FULL DISCLOSURE REPORT

9.1 Full Disclosure Report Requirements

A **Full Disclosure Report (FDR)** is required. This section specifies the requirements for the FDR.

The **FDR** is a zip file of a directory structure containing the following:

- A **Report** in Adobe Acrobat PDF format,
 - An **Executive Summary Statement** in Adobe Acrobat PDF format, and
 - The **Supporting Files** consisting of various source files, scripts, and listing files.
- Requirements for the **FDR** file directory structure are described below.

Comment: The purpose of the **FDR** is to document how a benchmark result was implemented and executed in sufficient detail so that the result can be reproduced given the appropriate hardware and software products.

9.1.1 General Items

The order and titles of sections in the **Report** must correspond with the order and titles of sections from the TPC-E Standard Specification (i.e., this document). The intent is to make it as easy as possible for readers to compare and contrast material in different **Reports**.

The directory structure of the **FDR** has three folders:

- *Report* - contains the **Report**,
- *ExecutiveSummaryStatement* - contains the **Executive Summary Statement**
- *SupportingFiles* - contains the **Supporting Files**.

The directory structure under *SupportingFiles* must follow the clause numbering in Clause 9. If there is more than one instance of one type of file, subfolders may be used for each instance. For example if multiple **Tier A** machines were used in the benchmark, there may be a folder for each **Tier A** machine.

9.1.2 Executive Summary Statement

9.1.2.1 The TPC **Executive Summary Statement** must be included near the beginning of the **Report**. A diagram describing the components of the priced configuration that are required to achieve the performance result must be included in the **Executive Summary Statement**. An example of the **Executive Summary Statement** is presented in Appendix B. The latest version of the required format is available from the TPC Administrator.

9.1.2.2 The first page of the Executive Summary Statement must include the following:

- **Sponsor's** name
- Measured server's name
- TPC-E Specification version number under which the benchmark is published
- TPC-Pricing Specification version number under which the benchmark is published
- Report date and/or Revision Date
- **Reported Throughput**
- **Price/Performance metric**

- **Availability Date**
- Total System Cost
- **Database server's operating system** name and version
- Database Manager name and version
- Number of **Database Server** Processors/Cores/Threads that were enabled for the benchmark
- Memory in GB configured on the **Database Server**
- **Priced configuration**
- Initial used database size in GB
- Redundancy Level and Redundancy Level implementation details
- Priced spindle count for the database

9.1.2.3 The Price Spreadsheet must be included in the **Executive Summary Statement**.

- 9.1.2.4 The numerical quantities listed below must be included in the **Executive Summary Statement**:
- **Measurement Interval** in hh:mm:ss (hours, minutes, seconds),
 - **Ramp-up** time in hh:mm:ss,
 - **Business Recovery Time** in hh:mm:ss (see Clause 7.5.6),
 - number of **transactions** (all types except Trade-Cleanup) completed within the **Measurement Interval**, (give the total, and the number per **transaction** type)
 - **Reported Throughput** in tpsE,
 - Ninetieth percentile, minimum, maximum and average response times must be **reported** for all **transaction** types except Trade-Cleanup.
 - Percentage of **transaction mix** for each **transaction** type.

Comment: Appendix B contains an example of such a summary. The intent is for data to be conveniently and easily accessible in a familiar arrangement and style. It is not required to precisely mimic the layout shown in Appendix B.

9.1.3 Report

- 9.1.3.1 Diagrams of both measured and **Priced Configurations** must be **reported** in the **Report**, accompanied by a description of the differences. This includes, but is not limited to:
- Number and type of processors, number of cores and number of threads.
 - Size of allocated memory, and any specific mapping/partitioning of memory unique to the test.
 - Number and type of disk units (and controllers, if applicable).
 - Number of channels or bus connections to disk units, including their protocol type.
 - Number of LAN (e.g. Ethernet) connections, including routers, workstations, etc., that were physically used in the test or incorporated into the pricing structure.
 - Type and the run-time execution location of software components (e.g. **DBMS**, client, processes, transaction monitors, software drivers, etc.).

Comment: Detailed diagrams for system configurations and architectures can widely vary, and it is impossible to provide exact guidelines suitable for all implementations. The intent here is to describe the system components and connections in sufficient detail to allow independent reconstruction of the measurement environment.

9.1.3.2 The following sample diagram illustrates a server benchmark (measured) configuration using a 32-processor server. The server uses 3 SCSI Controllers each attached to four 72GB 15Krpm drives. Gigabit Ethernet is used to link the **driver** machine to the middle-tier machines, and the middle-tier machines to the server. Note that this diagram does not depict or imply any optimal configuration for the TPC-E benchmark measurement.

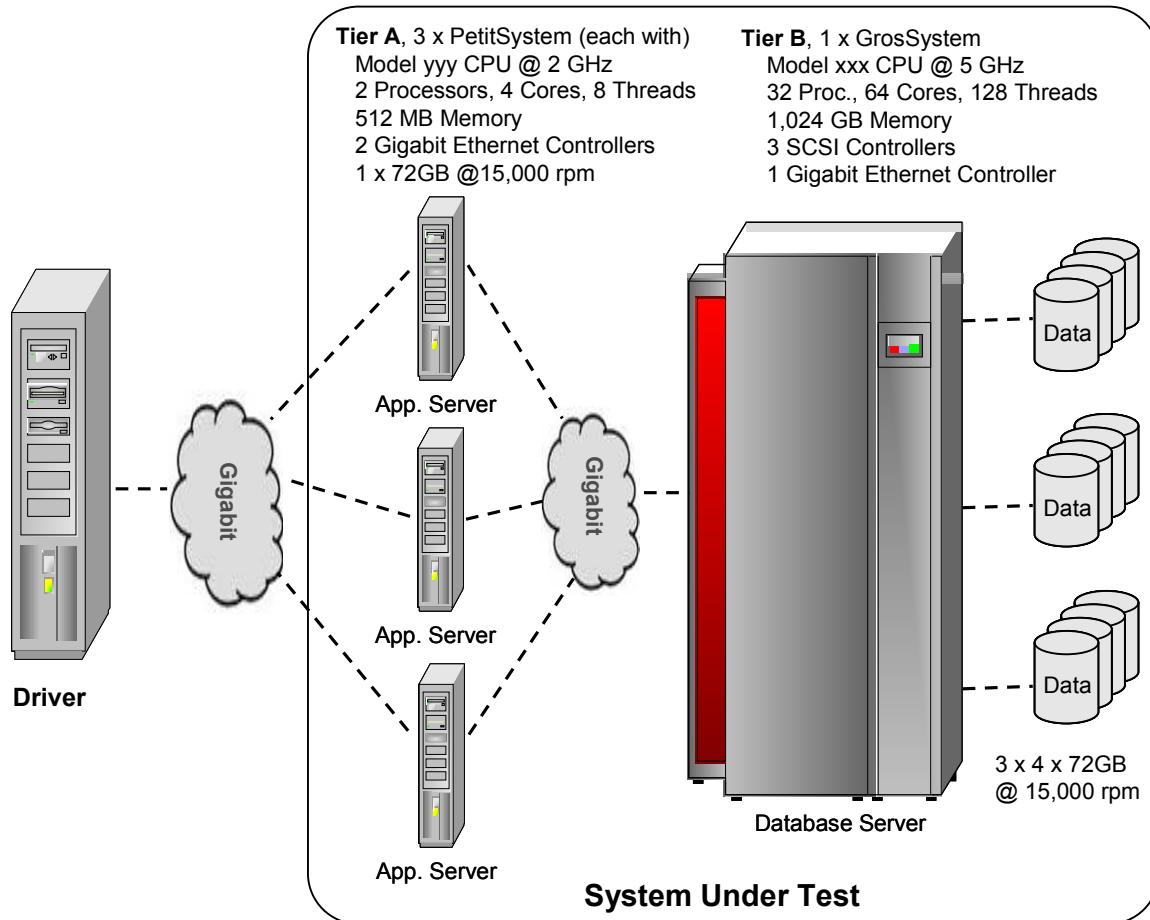


Figure 9.a - Example of Measured Benchmark Configuration

9.1.4 Supporting Files

9.1.4.1 The **Frame Implementation** (as described in Clause 4.2) of each **transaction** must be **reported** in the **Supporting Files**. This includes, but is not limited to, the code implementing the twelve **transactions** (see Clause 3.3) of this benchmark.

9.1.4.2 All information required by this clause must be **reported** in the **Supporting Files**.

A statement identifying the benchmark **sponsor(s)** and other participating companies must be **reported** in the **Report**.

Settings must be **reported** for all **Tunable Parameters** and options which have been changed from the defaults in actual products, including but not limited to:

- Database tuning options.
- Recovery/commit options.
- Consistency/locking options.

- **Operating System** and application configuration parameters.
- Compilation and linkage options and run-time optimizations used to create/install applications, OS, and/or databases.
- Parameters, switches or flags that can be changed to modify the behavior of the product.

All information required by this clause must be **reported** in the **Supporting Files**.

Comment 1: This requirement can be satisfied by providing a full list of all parameters and options.

Comment 2: The intent of the above clause is that anyone attempting to recreate the benchmark environment has sufficient information to compile, link, optimize, and execute all software used to produce the submitted benchmark result.

9.2 Additional Disclosure Requirements

9.2.1 Clause 2 Database Design, Scaling & Population Related Items

9.2.1.1 Scripts must be provided for all table definition statements and all other statements used to set-up the database. All scripts must be human readable and machine executable (i.e., able to be performed by the appropriate program without modification).

Comment: All information required by this clause must be **reported** in the **Supporting Files**.

9.2.1.2 The physical organization of tables and indices, within the database, must be **reported** in the **Report**.

Comment: The concept of physical organization includes, but is not limited to: record clustering (i.e., rows from different logical tables are co-located on the same physical data page), index clustering (i.e., rows and leaf nodes of an index to these rows are co-located on the same physical data page), and partial fill-factors (i.e., physical data pages are left partially empty even though additional rows are available to fill them).

9.2.1.3 While there are a few restrictions placed upon horizontal or vertical partitioning of tables and rows in the TPC-E benchmark (see Clause 2.5), any such partitioning must be **reported** in the **Report**. Using the CUSTOMER table as an example, such partitioning could be denoted as:

C_part_1	C_ID
	C_TAX_ID
	C_ST_ID
	C_L_NAME
	C_F_NAME
	C_M_NAME
	C_GNDR
	C_TIER
	C_DOB
	C_AD_ID
----- vertical partition -----	
C_part_2	C_CTRY_1
	C_AREA_1
	C_LOCAL_1

C_EXT_1
 C_CTRY_2
 C_AREA_2
 C_LOCAL_2
 C_EXT_2
 C_CTRY_3
 C_AREA_3
 C_LOCAL_3
 C_EXT_3
 C_EMAIL_1
 C_EMAIL_2

Once the partitioned database elements have been so identified, they can be referred to by, for example, their **T_part_N** notation when describing the physical allocation of database files (see Clause 9.2.1.7), where T indicates the table name and N indicates the partition segment number.

- 9.2.1.4 Replication of tables, if used, must be **reported** in the **Report** (see Clause 2.3.4).
- 9.2.1.5 Additional and/or duplicated attributes in any table must be **reported** in the **Report** along with a statement on the impact on performance (see Clause 2.3.5).
- 9.2.1.6 The cardinality (e.g. the number of rows) of each table, as it existed after database load (see Clause 2.6), must be **reported** in the **Report**.
- 9.2.1.7 The distribution of tables, partitions and logs across all media must be explicitly depicted for tested and priced systems.

Comment: The intent is to provide sufficient detail to allow independent reconstruction of the test database.

Disk #	Controller #	Slot #	Drives Enclosure model RAID level	Partition/file system	Size	Use
1	1	3	28 X 36.4GB EEENNN Enclosure RAID 10	E: (RAW) F: (NTFS)	200.00GB 10.00GB	DB Log MDF File
2	2	4	14 X 36.4GB EEENNN Enclosure RAID 10	C:\mp\dimension (RAW) C:\mp\market (RAW)	0.10GB 50.50GB	Dimension Market
3	2	4	14 X 74.8GB EEENNN Enclosure RAID 10	C:\mp\customer (RAW) G: (NTFS)	70.00GB 10.00GB	Customer Backup 1
4	3	5	28 X 74.8GB EEENNN Enclosure RAID 10	C:\mp\broker1 (RAW) H: (NTFS)	44.25GB 10.00GB	Broker Backup 2
5	4	1	28 X 74.8GB EEENNN Enclosure RAID 10	C:\mp\broker2 (RAW) I: (NTFS)	44.25GB 10.00GB	Broker Backup 3

9.2.1.8 A statement must be provided in the **Report** that describes:

- The **Database Interface** (e.g., embedded, call level) and access language (e.g., SQL, COBOL read/write) used to implement the TPC-E **transactions**. If more than one interface / access language is used to implement TPC-E, each interface / access language must be described and a list of which interface / access language is used with which **transaction** type must be **reported**.
- The data model implemented by the **DBMS** (e.g., relational, network, hierarchical).
- The mapping of database partitions / replications must be **reported** in the **Report** (i.e., as a textual description) in addition to the inclusion of database scripts in the **Supporting Files**.

Comment: The intent is to provide sufficient detail about partitioning and replication to allow independent reconstruction of the test bed.

9.2.1.9 The methodology used to load the database must be **reported** in the **Report**. Additionally, all non-commercially available code/scripts/etc., other than **EGen**, used to load the data into the database must be **reported** in the **Supporting Files**.

9.2.2 Clause 3 Transaction Related Items

9.2.2.1 A statement that all required TPC-provided **EGen** code was used in the benchmark must be **reported** in the **Report**. The version of **EGen** used in the benchmark must be **reported** in the **Report**.

9.2.2.2 All vendor-supplied **Frame Implementation** source code must be **reported** in the **Supporting Files**.

9.2.2.3 A statement that vendor-supplied code is functionally equivalent to pseudo-code in the specification (see Clause 3.1.2) must be provided in the **Report**.

9.2.2.4 A statement that the database footprint requirements (as described in Clause 3.1.2) were met must be provided in the **Report**.

9.2.3 Clause 4 SUT, Driver, and Network Related Items

9.2.3.1 A single benchmark Test Run maybe used to publish multiple results with minor differences in the **SUT** provided the following conditions are met:

- Each **SUT** must have the same hardware and software architecture and configuration. The only exceptions allowed are for elements not involved in the processing logic of the **SUT** (e.g., number of peripheral slots, power supply, cabinetry, fans, etc.)
- Each **SUT** must support the **Priced Configuration**.

Comment: The intent of this clause is to allow publication of benchmarks for systems with different packaging and model numbers that are considered to be identical using the same benchmark run. For example, a rack mountable system and a freestanding system with identical electronics can use the same **Test Run** for publication, with, appropriate changes in pricing.

9.2.3.2 The number of **EGenDriverMEE** and **EGenDriverCE** instances used in the benchmark must be disclosed in the **Report**. The **EGenLogger** output for each CCE object, CMEE object and CDM object must be **reported** in the **Supporting Files**.

9.2.4 Clause 5 EGen Related Items

9.2.4.1 The version of **EGen** used in the benchmark must be **reported** in the **Report**.

9.2.4.2 If the **test sponsor** modified **EGen**, those changes must be **reported** in the **Report** and in the **Supporting Files**. All formal waivers from the TPC documenting the allowed changes to **EGen** must also be **reported** in the **Report**. If any of the changes to **EGen** do not have a formal waiver that must also be **reported** in the **Report**.

9.2.4.3 If the **test sponsor** extended **EGenLoader** (as described in Appendix A.7), the use of the extended **EGenLoader** and the audit of the extension code by an **Auditor** must be **reported** in the **Report**. The extension code must be **reported** in the **Supporting Files**.

9.2.4.4 The **Driver** configuration information must be **reported**. This includes reporting the **EGenDriverCE**, **EGenDriverMEE** and **EGenDriverDM** configuration in the **Supporting Files**.

9.2.4.5 The **EGenLoader** parameters used must be **reported** in the **Supporting Files**.

9.2.5 Clause 6 Performance Metrics and Response Time Related Items

9.2.5.1 Measured tpsE must be **reported** in the **Executive Summary Statement**.

9.2.5.2 Ninetieth percentile, minimum, maximum and average response times must be **reported** for all **transaction** types in the **Executive Summary Statement**.

9.2.5.3 The **Pacing Delay** (see Clause 6.4.2) must be **reported** in the **Executive Summary Statement**.

9.2.5.4 A graph of throughput versus elapsed time must be **reported** in the **Report** for the Trade-Result **transaction** (see Clause 6.6.1).

9.2.5.5 The output from **EGenTester** must be **reported** in the **Supporting Files** (see Clause 6.6.3).

9.2.5.6 The method used to determine that the **SUT** had reached a **Steady State** prior to commencing the **Measurement Interval** must be described in the **Report**.

9.2.5.7 A description of how the work normally performed during a sustained test (for example checkpointing, writing **Undo/Redo Log** records, etc.), actually occurred during the **Measurement Interval** must be **reported** in the **Report**.

9.2.5.8 A statement of the duration of the **Measurement Interval** for the **reported Measured Throughput** must be included the **Executive Summary Statement**.

9.2.5.9 The percentage of the total mix for each **transaction** type must be **reported** in the **Executive Summary Statement**.

9.2.5.10 The recorded averages over the **Measurement Interval** for the following parameters must be **reported** in the **Report**:

- The percentage of Trade-Order **transactions** rolled back as a result of the `roll_it_back` parameter;
- The percentage of Trade-Order **transactions** for market-buy, market-sell, limit-buy, limit-sell and stop-loss;
- The percentage of Trade-Order **transactions** by security name;
- The percentage of Trade-Order **transactions** by account owner;
- The percentage of Trade-Order **transactions** by the margin variant;
- The percentage of Trade-Order **transactions** with `is_lifo` set to TRUE ;
- The percentage of Trade-Order **transactions** by company name and issue;
- The percentage of Security-Detail **transactions** with `access_lob_flag` set to TRUE;
- The number of rows requested in `max_rows_to_return` for the Security-Detail **transactions**;
- The percentage of Market-Watch **transactions** by watch list (via customer identifier), by company industry and by customer account;
- The percentage of Customer-Position **transactions** by tax ID;
- The percentage of Customer-Position **transactions** with `get_history` flag set to TRUE;
- The percentage of Trade-Lookup **transactions** executing **frame 1**, **frame 2**, **frame 3** or **frame 4**;
- The percentage of Trade-Update **transactions** executing **frame 1**, **frame 2**, or **frame 3**; and
- The number of brokers passed into the Broker-Volume **transactions**.

9.2.6 Clause 7 Transaction and System Properties Related Items

- 9.2.6.1 The results of the ACID tests must be **reported** in the **Report** along with a description of how the ACID requirements were met, and how the ACID tests were run. The scripts and the output of the ACID tests must be **reported** in the **Supporting Files**.
- 9.2.6.2 The **test sponsor** must **report** in the **Report** the Redundancy Level (see Clause 7.5.2.1) and describe the test(s) used to demonstrate compliance.
- 9.2.6.3 A Data **Accessibility** Graph for each run demonstrating a Redundancy Level must be **reported** in the **Report** (see Clause 7.5.2.4).
- 9.2.6.4 The **test sponsor** must describe in the **Report** the test(s) used to demonstrate **Business Recovery**.
- 9.2.6.5 The **Business Recovery Time** (the elapsed time between steps 6 through step 8 of Clause 7.5.5) must be **reported** on the **Executive Summary Statement** and in the **Report**. If the failures described in Clause 7.5.4 were not combined into one **durability** test (usually powering off the **Database Server** during the run), then the **Business Recovery Time** for the failure described for instantaneous interruption is the **Business Recovery Time** that must be **reported** in the **Executive Summary Statement**. All the **Business Recovery Times** for each test requiring **Business Recovery** must be **reported** in the **Report**.
- 9.2.6.6 The **Business Recovery Time** Graph (see Clause 7.5.6.2) must be **reported** in the **Report** for all **Business Recovery** tests.

9.2.7 Clause 8 Pricing Related Items

- 9.2.7.1 Rules for reporting pricing information are included in the current revision of version 1 of the TPC Pricing Specification, located at www.tpc.org.
- 9.2.7.2 Details of the **60-Day-Space** computations (see Clause 8.2.2) along with proof that the database is configured to sustain a **Business Day** of growth (see Clause 6.5.6.1) must be **reported** in the **Report**. A spreadsheet detailing the **60-Day-Space** calculations must be reported in the **Supporting Files**.
- 9.2.7.3 Price Spreadsheet Categories:
The major categories for division of the price spreadsheet are:
- Server Hardware
 - Server Storage
 - Server Software
 - Client Hardware
 - Client Software
 - Infrastructure (networking, UPS, consoles, other components that do not fit into the above categories)

CLAUSE 10 -- INDEPENDENT AUDIT

10.1 General Rules

10.1.1 Audit Requirements

- 10.1.1.1 Prior to its publication, a TPC-E **result** must be reviewed by a **TPC-Certified**, independent **Auditor**.

Comment 1: The term **TPC-Certified** is used to indicate that the TPC has reviewed the qualification of the **Auditor** and has certified his/her ability to verify that benchmark **results** are in compliance with this specification. (Additional details regarding the **Auditor** certification process and the audit process can be found in the TPC Policy document.)

Comment 2: The **Auditor** must be independent from the **sponsor** in that the outcome of the benchmark carries no financial benefit to the **Auditor**, other than fees earned as a compensation for performing the audit. More specifically:

- The **Auditor** is not allowed to have supplied any performance consulting for the benchmark under audit.
- The **Auditor** is not allowed to be financially related to the **sponsor** or to any one of the suppliers of a measured/priced component (e.g., the **Auditor** cannot be an employee of an entity owned wholly or in part by the **sponsor** or by the supplier of a benchmarked component, and the **Auditor** cannot own a significant share of stocks from the **sponsor** or from the supplier of any benchmarked component, etc.)

- 10.1.1.2 A generic audit checklist is provided as part of this specification. The **Auditor** may choose to provide the **sponsor** with additional details on the TPC-E audit process.

- 10.1.1.3 The **Auditor's** opinion regarding the compliance of a **result** must be consigned in an **attestation letter** delivered directly to the **sponsor**. To document that a result has been audited, the **attestation letter** must be included in the **Report** and made readily available to the public. Upon request, and after approval from the **sponsor**, a detailed audit report may be produced by the **Auditor**.

- 10.1.1.4 The scope of the audit is limited to the functions defined in this specification. The ability to perform arbitrary functions against the **SUT** (e.g., executing **transactions** unrelated to those defined in Clause 3.3, generating input data unrelated to those produced by the **CE** and the **MEE**, creating data structures unrelated to those necessary to implement Clause 2, etc.) is outside of the scope of the audit.

- 10.1.1.5 A **sponsor** can demonstrate compliance of a new **result** produced without running any performance test by referring to the **attestation letter** of another **result**, if the following conditions are all met:
- The referenced **result** has already been published by the same or by another **sponsor**.
 - The **sponsor** of the already published **result** gives written approval for its use as referenced by the **sponsor** of the new **result**.
 - The **Auditor** verifies that there are no significant functional differences between the priced components used for both **results** (i.e., differences are limited to labeling, packaging and pricing.)

- The **Auditor** reviews the **FDR** of the new **result** for compliance. The **Auditor** delivers a new **attestation letter** to be included in the **Report** of the new **result**.

Comment 1: Although it should be apparent to a careful reader that the **FDR** for the two **results** are based on the same set of performance tests, the **FDR** for the new **result** is not required to explicitly state that it is based on the performance tests of another published **result**.

Comment 2: When more than one **result** is published based on the same set of performance tests, only one of the **results** from this group can occupy a numbered slot in each of the benchmark result “Top Ten” lists published by the TPC. The **sponsors** of this group of **results** must all agree on which **result** from the group will occupy the single slot. In case of disagreement among the **sponsors**, the decision will be made by the **sponsor** of the earliest publication from the group.

10.2 Audit Check List

10.2.1 Auditing the Database

- 10.2.1.1 Verifying that the implementation of the tested database meets the benchmark requirements may require the **Auditor** to review the source code used to create and populate the database and any associated scripts or programs.
- 10.2.1.2 Verify that all **Primary Keys**, all **Foreign Keys**, and all check constraints specified are maintained by the database (see Clause 2.2.3).
- 10.2.1.3 Verify that the 9 tables in the Customer set have all of the specified attributes (see Clause 2.2.4).
- 10.2.1.4 Verify that the 9 tables in the Broker set have all of the specified attributes (see Clause 2.2.5).
- 10.2.1.5 Verify that the 11 tables in the Market set have all of the specified attributes (see Clause 2.2.6).
- 10.2.1.6 Verify that the 4 tables in the Dimension set have all of the specified attributes (see Clause 2.2.7).
- 10.2.1.7 Verify that the data types used to implement the attributes of the tables meet the definitions from Clause 2.2.1.
- 10.2.1.8 Verify that table partitioning, if used, meets the requirements from Clauses 2.3.3.
- 10.2.1.9 Verify that all copies of any replicated table, if used, meets all requirements for atomicity, consistency, and isolation (see Clause 2.3.4).
- 10.2.1.10 Verify that any attributes added and/or duplicated from one table to another, if any, does not also result in a performance improvement (see Clause 2.3.5).
- 10.2.1.11 Verify that all table attributes are discrete (see Clause 2.3.8).
- 10.2.1.12 Verify that **Primary Keys** are not a direct representation of the physical disk addresses of the row (see Clause 2.3.8).
- 10.2.1.13 Verify that each non-**Growing Table** can grow by a number of rows equal to at least 5% of the table cardinality (see Clause 2.3.9).
- 10.2.1.14 Verify that LOB attributes are implemented with the required properties (see Clause 2.3.12).
- 10.2.1.15 Verify that the implementation of the database satisfies the integrity rules (see Clause 2.4).
- 10.2.1.16 Verify that the implementation of the database satisfies the data access transparency requirements (see Clause 2.5). In particular:
 - Verify that the database population is generated using a compliant version of **EGen**, and that no unapproved modifications have been made to the **EGen** code or **EGen** flat files (see Clause 5.3.6).
- 10.2.1.17 Verify that the initial database meets the requirements (see Clause 2.6.1). In particular:
 - Verify that the database is populated with an integral number of **load units** (see Clause 2.6.1.2).

- Verify that the initial database population consists of a number of business days equal to **ITD** (see Clause 2.6.1.6).
- Verify that the cardinality of the tables in the initially populated database meets the requirements (see Clause 2.6.1).

10.2.2 Auditing the Transactions

- 10.2.2.1 Verifying that the implementation of the **transactions** meets the benchmark requirements may require the **Auditor** to review the source code for these transactions and for any associated scripts or programs.
- 10.2.2.2 Verify that all **Frames** are implemented without circumventing any specified database references to static or infrequently changing data elements (see Clause 3.2.1.1).
- 10.2.2.3 Verify that **Frames** do not exchange data outside of the specified input and output parameters used to communicate with the **EGenTxnHarness** (see Clause 3.2.1.2).
- 10.2.2.4 Verify that the specified access methods are used to implement the database interactions defined for each **Frame** (see Clauses 3.2.1.3 to 3.2.1.5).
- 10.2.2.5 Verify that the method calls used to invoke the execution of the **Frames** check for null columns being returned and set the output parameters appropriately.
- 10.2.2.6 Verify that the implementation of each type of **transaction** is compliant with the requirements of the benchmark. More specifically:
- Verify that the Trade-Order **transaction** is compliant with the requirements defined in Clause 3.3.1.
 - Verify that the Trade-Result **transaction** is compliant with the requirements defined in Clause 3.3.2.
 - Verify that the Trade-Lookup **transaction** is compliant with the requirements defined in Clause 3.3.3.
 - Verify that the Trade-Update **transaction** is compliant with the requirements defined in Clause 3.3.4.
 - Verify that the Trade-Status **transaction** is compliant with the requirements defined in Clause 3.3.5.
 - Verify that the Customer-Position **transaction** is compliant with the requirements defined in Clause 3.3.6.
 - Verify that the Broker-Volume **transaction** is compliant with the requirements defined in Clause 3.3.7.
 - Verify that the Security-Detail **transaction** is compliant with the requirements defined in Clause 3.3.8.
 - Verify that the Market-Feed **transaction** is compliant with the requirements defined in Clause 3.3.9.
 - Verify that the Market-Watch **transaction** is compliant with the requirements defined in Clause 3.3.10.
 - Verify that the Data-Maintenance **transaction** is compliant with the requirements defined in Clause 3.3.11.

-
- Verify that the Trade-Cleanup **transaction** is compliant with the requirements defined in Clause 3.3.12.

10.2.3 Auditing the SUT, Driver and Networks

- 10.2.3.1 Verifying that the implementation of the test environment meets the benchmark requirements may require the **Auditor** to review the source code implementing the various components involved and any associated scripts or programs.
- 10.2.3.2 Verify that the format of the data provided to the **EGenDriver Connector** and the **EGenTxnHarness Connector** is not modified, except as permitted (see Clause 4.1.3).
- 10.2.3.3 Verify the presence and use of a **Network** to communicate between the **Driver** and **Tier A** (see Clause 4.2).
- 10.2.3.4 Verify that no data from the tested database is present on the **Driver**, except as permitted (see Clause 4.4.1).
- 10.2.3.5 Verify that **Drivers** using customer partitioning comply with the associated restrictions (see Clause 4.4.1).
- 10.2.3.6 Verify that the “no-peeking-in-the-packet” rule is followed (see Clause 4.4.1).
- 10.2.3.7 Verify that no routing is done in the **Frame Implementation** (see Clause 4.4.1).
- 10.2.3.8 Verify that the restrictions on operator interventions are met (see Clause 4.4.3).

10.2.4 Auditing EGen

- 10.2.4.1 Verify that the version of **EGen** used is compliant with the version of the TPC-E specification used for publication (see Clause 5.3).
- 10.2.4.2 If the **test sponsor** modified **EGen** in response to a formal waiver issued by the TPC, verify that the changes fall under the scope of the waiver (see Clause 5.3.6).
- 10.2.4.3 If the **test sponsor** modified **EGen** outside of an existing TPC waiver, review the changes to verify that it was done for the exclusive purpose of correcting a newly discovered error in **EGen** (see Clause 5.3.5).
- 10.2.4.4 Verify that any **EGen** changes made by the **sponsor** are **reported** in detail in the **FDR** (see Clause 9.2.4.2).
- 10.2.4.5 Verify that the **EGenInputFiles** used have not been modified and that copies of the same files are used in all **EGenLoader** and **Driver** instances.
- 10.2.4.6 Verify that the **EGenSourceFiles** used have not been modified.
- 10.2.4.7 Verify that the version of **EGenLoader** used is compliant with the current version of the TPC-E specification (see Clause 5.7.1).
- 10.2.4.8 Verify that modifications or extensions made by the **sponsor** to **EGenLoader** do not compromise the values for the data generated by **EGenLoader** (see Clause 5.7.3).

- 10.2.4.9 Verify that modifications or extensions made by the **sponsor** to **EGenLoader** are documented in sufficient detail in the **Report** and that the code for the modification or extension is **reported** in the **Supporting Files** (see Clause 9.2.4.3).
- 10.2.4.10 Verify that none of the **EGenLogger** output contains “NO” where the use of default values is checked.
- 10.2.4.11 Verify that the **CE** is implemented using the **EGenDriverCE** (see Clause 5.8.4).
- 10.2.4.12 Verify that the **MEE** is implemented using **EGenDriverMEE** (see Clause 5.8.5).
- 10.2.4.13 Verify that the Data-Maintenance **transaction** is implemented using **EGenDriverDM** (see Clause 5.8.6).
- 10.2.4.14 Verify that one, and only one, Data-Maintenance **transaction** generator is used during runtime (see Clause 5.8.6.2).
- 10.2.4.15 Verify that the implementation uses **EGenTxnHarness** (see clause 5.9.1).

10.2.5 Auditing the Execution Rules and Metrics

- 10.2.5.1 Verify that the **reported transaction mix** over the **Measurement Interval** only counts **valid transactions** (see Clause 6.2).
- 10.2.5.2 Verify that the specified mix of **transactions** over the **Measurement Interval** meets the requirements (see Clause 6.2.2.1).
- 10.2.5.3 Verify that the **reported transaction mix** over the **Measurement Interval** meets all the required percentages (see Clause 6.2.2.2) and is computed and **reported** with the required precision and rounding.
- 10.2.5.4 Verify that the **reported transaction mix** over the **Measurement Interval** excludes the Data-Maintenance **transactions** (see Clause 6.2.2).
- 10.2.5.5 Verify that during the **Measurement Interval** the Data-Maintenance **transaction** is invoked every 60 seconds and completes within no more than 55 seconds (see Clause 6.2.2.4).
- 10.2.5.6 Verify that the Trade-Cleanup **transaction** was executed prior to the start of the **Test Run** or that the database was in its initially populated state (e.g., verify that the final TRADE count minus the number of Trade-Orders completed by the **Driver** during the **Test Run** is equal to the initial TRADE count). Verify that no executions of the Trade-Cleanup **transaction** occur during the **Test Run** (see Clause 6.2.2.5).
- 10.2.5.7 Verify that, for specific global inputs, each instance of the **CE** or the **MEE** is using the same values as those used by the **EGenLoader** instances during the initial database population (see Clause 6.3.1.1). This applies to the following global inputs:
 - The contents of each flat_in file.
 - The value for Scale **Factor (SF)**.
 - The number of **Initial Trade Days**.
 - The number of **Configured Customers**.

- Verify that, for specific global inputs, each instance of the **CE** or the **MEE** is using a value within the range that was used by the **EGenLoader** instances during the initial database population (see Clauses 5.8 and 6.3.1.1).

- 10.2.5.8 Verify that the **CE Driver** generates input data with a random variability that stays within the specified ranges (see Clause 6.3.1.2).
- 10.2.5.9 If the **CE** instances are partitioned, verify that they meet the requirements (see Clause 6.3.2).
- 10.2.5.10 Verify that the **transaction response times** meet the requirements (see Clause 6.4.1.2).
- 10.2.5.11 Verify that the **pacing delay** meets the requirements (see Clause 6.4.2).
- 10.2.5.12 Verify that the structure and timings of the **Test Run** meet the requirements (see Clause 6.5).
- 10.2.5.13 Verify that the **Steady State** is achieved and maintained during the **Measurement Interval** (see Clause 6.5.3).
- 10.2.5.14 Verify that all events performed at regular intervals during **Steady State** are present before and during the **Measurement Interval** as required (see Clause 6.5.5.2).
- 10.2.5.15 Verify that the **Pacing Delay** is no longer adjusted during **Steady State** (see Clause 6.5.5.3).
- 10.2.5.16 Verify that the **Data-Growth** is computed as specified and that sufficient space to accommodate it is available on-line (see Clause 6.5.6).
- 10.2.5.17 Verify the **Measured Throughput** is between 80% and 102% of the **Nominal Throughput** (see Clause 6.5.8.2).
- 10.2.5.18 Verify that the **Throughput Rating** is not greater than the **Nominal Throughput** (see Clause 6.5.8).
- 10.2.5.19 Verify that the **Reported Test Run Graph** meets the requirements (see Clause 6.6.1).
- 10.2.5.20 Verify that all primary metrics are **reported** in the **Executive Summary Statement** (see Clause 6.6.2).
- 10.2.5.21 Verify that output from **EGenTester** meets the requirements (see Clause 6.6.3).

10.2.6 Auditing the ACID Tests

- 10.2.6.1 Verifying that the implementation of the ACID tests sufficiently demonstrates compliance with the ACID requirements of the benchmark may require the **Auditor** to review the source code implementing these tests and any associated scripts or programs.
- 10.2.6.2 Verify that the atomicity test is implemented as specified (see Clause 7.2.2).
- 10.2.6.3 Verify that the atomicity property is successfully demonstrated by the test (see Clause 7.2.2).
- 10.2.6.4 Verify that the consistency tests are implemented as specified (see Clause 7.3.1.1).
- 10.2.6.5 Verify that the consistency conditions are successfully demonstrated by the tests (see Clause 7.3.1.1)
- 10.2.6.6 Verify that the isolation tests are implemented as specified (see Clause 7.4.2).

- 10.2.6.7 Verify that the isolation requirements are successfully demonstrated by the tests (see Clause 7.4.2).
- 10.2.6.8 Verify that the durability tests for **Data Accessibility** is implemented as specified (see Clause 7.5.2.2).
- 10.2.6.9 Verify that the Redundancy Level chosen by the **sponsor** is successfully demonstrated by the **Data Accessibility** test (see Clause 7.5.2).
- 10.2.6.10 Verify that the Redundancy Level chosen by the **sponsor** is correctly **reported** in the **Report** (see Clause 7.5.2.3).
- 10.2.6.11 Verify that a **Data Accessibility** Graph is generated as specified and included in the **Report** (see Clause 7.5.2.4).
- 10.2.6.12 Verify that the durability tests for **Business Recovery** is implemented as specified (see Clause 7.5.3).
- 10.2.6.13 Verify that recovery from each required single failure scenario is successfully demonstrated by one or more **Business Recovery** tests (see Clause 7.5.4).
- 10.2.6.14 Verify that the measured **Business Recovery Time** is correctly **reported** in the **Report** and **Executive Summary Statement** (see Clause 7.5.4).
- 10.2.6.15 Verify that a **Business Recovery** Graph is generated as specified and included in the **Report** (see Clause 7.5.5).
- 10.2.7 Auditing the Pricing**
- 10.2.7.1 Rules for auditing Pricing information are included in the current revision of the TPC Pricing Specification, located at www.tpc.org.
- 10.2.7.2 Verify that the greater of the **60-Day-Space** or the data storage configured during the measurement is included in the priced configuration (see Clause 8.2.2).
- 10.2.7.3 Verify that additional operational components or additional software that might be customary on a customer installed configuration or might be necessary to build and run the application are included (see Clause 8.3.1 and Clause 8.3.2).
- 10.2.7.4 Verify that all component substitutions are compliant with the TPC Pricing Specification and with the TPC-E specific restrictions (see Clause 8.4).
- 10.2.8 Auditing the FDR**
- 10.2.8.1 Verify that the **Executive Summary Statement** is accurate and complies with the reporting requirements.
- 10.2.8.2 Verify that the following sections of the **Report** are accurate and comply with the reporting requirements:
- The diagrams of both measured and **Priced Configurations** and the textual description of the differences between the two configurations;

- The graph of throughput versus elapsed time for the Trade-Result **transaction**;
- The recorded averages over the **Measurement Interval** for the parameters listed in Clause 9.2.5.10;
- The **Business Recovery Time** Graph for all **Business Recovery** tests;
- The textual description of the physical organization of tables and indices, within the database;
- The textual description of any partitioning or replication within the database;
- The cardinality for each table within the database;
- The textual description of the distribution of tables, partitions and logs across all media for both measured and **Priced Configurations** and the textual description of the differences between the two distributions; and
- Any third party price quotations.

10.2.8.3 A complete review of the **Report** by the **Auditor**, beyond the sections listed above, can be requested by the **sponsor**, but is not required.

APPENDIX A. EGEN USER'S GUIDE

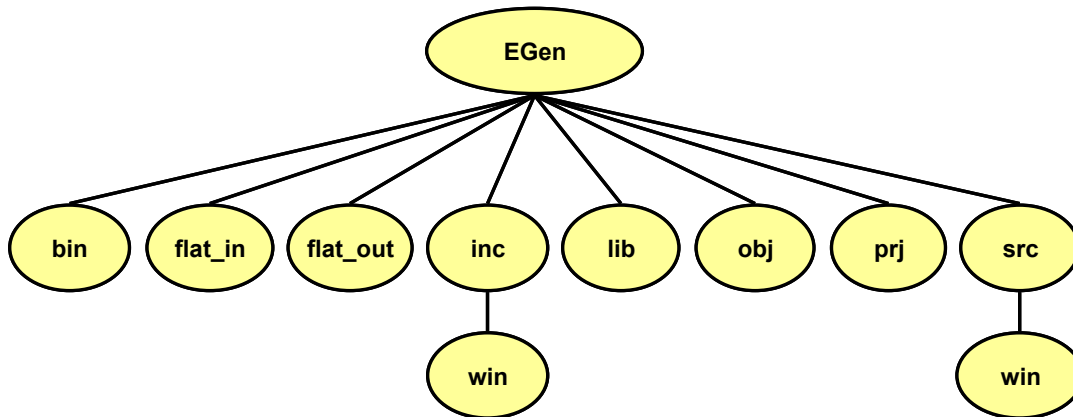
A.1 Overview

EGen is a TPC provided software package. It is designed to facilitate the implementation of TPC-E. This appendix provides information on how a **test sponsor** is to use the features and functionality of **EGen**. The definitions, descriptions, constraints and regulations governing the use of **EGen** are captured in Clause 5 -- .

Comment: Some of the following sections assume the reader has a good understanding of object-oriented design and programming techniques using ANSI C++.

A.2 EGen Directory

A.2.1 **EGen** is distributed in a single directory hierarchy. The following diagram shows the overall **EGen** directory hierarchy.



Picture A.a - Hierarchy of EGen Directory

- bin – default target directory for executable binary files
- flat_in - contains flat input files
- flat_out - default target directory for flat file output
- inc – contains header files
- inc/win – Windows specific header files
- lib – default target directory for library files
- obj – default target directory for object files
- prj – contains project files
- src – contains source files
- src/win – Windows specific source files

A.3 EGenProjectFiles

A.3.1 **EGenProjectFiles** are located in the EGen/prj directory. These files are used to facilitate building **EGen** components in various environments.

- **Windows**
A set of Visual Studio 2003 files are provided. EGen.sln is the top level solution file and brings in all of the necessary .prj files.
- **U*x**
A make file (makefile) is provided to facilitate building the **EGen** components using a make utility.

A.4 EGenInputFiles

A.4.1 **EGenInputFiles** are located in the EGen/flat_in directory. These files are text files containing rows of tab-separated data. The files are used by various **EGen** components as “raw” material for data generation.

A.5 EGenSourceFiles

A.5.1 **EGenSourceFiles** are located in EGen/inc, EGen/src and their associated sub-directories.

A.5.2 **EGenSourceFiles** contain TPC-provided ANSI C++ code to be used in a compliant TPC-E implementation. Functionality is provided to facilitate:

- population of a TPC-E complaint database
- implementation of a TPC-E compliant runtime environment

This functionality is described in subsequent sections.

A.6 EGenLogger

A.6.1 **EGenLogger** is used by **EGenDriver** and **EGenLoader** to log their configuration and any re-configuration. Although not strictly required, the **Test Sponsor** is expected to override/provide a SendToLoggerImpl implementation for recording **EGenLogger's** output. For details see EGen/inc/EGenLogger.h.

A.7 EGenLoader

A.7.1 The task of populating a compliant TPC-E database can be broken into two parts:

- generating compliant data records
- loading the records into the database

- A.7.2 Data generation is a database-neutral task, whereas database population is obviously very database-specific. Therefore, **EGenLoader** is architected honoring this separation as follows. **EGenSourceFiles** contain class definitions that provide abstractions of the TPC-E tables. These table classes are known collectively as **EGenTables** and they encapsulate the functionality needed to generate the data for each of the TPC-E tables. Many of the classes in **EGenTables** are dependent on **EGenInputFiles** for “raw material” used in data record generation. **EGenLoader** therefore makes **EGenInputFiles** available to **EGenTables**, and uses **EGenTables** to generate TPC-E compliant data records.
- A.7.3 In order to support the database-specific nature of loading the generated data, **EGenLoader** makes use of a virtual base class **CBaseLoader** to “load” the data. This provides a controlled interface from the database-neutral data generation portion of **EGenLoader** to the database-specific data loading portion of **EGenLoader**. Database-specific code is encapsulated in subclasses that inherit from and provide an implementation of the virtual **CBaseLoader** class. (**Note:** **CBaseLoader** is actually a template, where the one template parameter is the row type corresponding to the particular table being loaded.) **EGenLoader** provides two alternative implementations of **CBaseLoader**.
- A.7.4 The first loader functionality provided by **EGenLoader** doesn’t actually load a database directly, but rather produces output flat files. One text file is produced for each table. These files contain rows of data fields, where the data fields are separated by “|”. To use this functionality define the compile-time variable **COMPILE_FLAT_FILE_LOAD** when building **EGenLoader** and use the “-l FLAT” switch when running **EGenLoader**.
- A.7.5 The second loader functionality provided by **EGenLoader** is for direct loading of a Microsoft SQL Server database via the ODBC interface. To use this functionality define the compile-time variable **COMPILE_ODBC_LOAD** when building **EGenLoader** and use the “-l ODBC” switch when running **EGenLoader**.
- A.7.6 **EGenLoader** can be extended by providing an implementation of the **CBaseLoader** template class in a sub-class named **CCustomLoader**. To use this functionality define the compile-time variable **COMPILE_CUSTOM_LOAD** and link with sponsor-provided code that implements the **CCustomLoader** class when building **EGenLoader**, and use the “-p” option to pass parameters to the custom loader.

A.8 EGenDriver

- A.8.1 A TPC-E **test sponsor** is responsible for implementing a compliant TPC-E **Driver** (Clause 4 --). The TPC provides **EGenDriver** to facilitate implementation of a compliant **Driver** and to standardize certain key platform-independent parts of the **Driver**.

A.8.2 **EGenDriver** comprises the following three parts.

- **EGenDriverCE** - any and/or all instantiations of the CCE class (see **EGenSourceFiles** CE.h and CE.cpp).
- **EGenDriverMEE** - any and/or all instantiations of the CMEE class (see **EGenSourceFiles** MEE.h and MEE.cpp).
- **EGenDriverDM** - the single instantiation of the CDM class (see **EGenSourceFiles** DM.h and DM.cpp).

A.8.3 **EGenDriver**, like **EGenLoader**, makes use of **EGenInputFiles** and **EGenTables** in data generation. This provides data generation coherency between database population time and **Test Run** time.

A.9 Implementing a CE using EGenDriverCE

A.9.1 Sending data to and receiving data from the **SUT** is very platform-specific functionality. Its implementation depends on the underlying communication protocol and hardware used. Likewise, measuring the **transaction's Response Time** is also platform-specific - depending on what timing mechanisms are provided by the underlying software and hardware.

However, the **transaction mix** (deciding which **transaction** to perform next) and generating the **transaction** input data is very platform-neutral. Therefore, **EGenDriverCE** encapsulates this functionality and provides a standardized implementation for it across all TPC-E implementations.

A.10 Implementing a MEE using EGenDriverMEE

A.10.1 Sending data to and receiving data from the **SUT** is very platform-specific functionality. Its implementation depends on the underlying communication protocol and hardware used. Likewise, measuring the **transaction's Response Time** is also platform-specific - depending on what timing mechanisms are provided by the underlying software and hardware.

However, emulating the internal stock exchange functionality, and generating the **transaction** input data for Trade-Result and Market-Feed is very platform-neutral. Therefore, **EGenDriverMEE** encapsulates this functionality and provides a standardized implementation for it across all TPC-E implementations.

Comment: A proper **MEE** implementation must be able to adjust to changing rates of trade requests and be able to turn-around trade requests into new Trade-Result transactions in a timely fashion. Similarly, a proper **MEE** implementation must be able to adjust to changing rates of Trade-Results and must initiate Market-Feed transactions in a timely fashion.

A.11 Implementing a Data-Maintenance Generator using EGenDriverDM

- A.11.1 Sending data to and receiving data from the **SUT** is very platform-specific functionality. Its implementation depends on the underlying communication protocol and hardware used. Likewise, measuring the Data-Maintenance **transaction's Response Time** is also platform-specific – depending on what timing mechanisms are provided by the underlying software and hardware.

However, generating the **transaction** input data for the Data-Maintenance **transaction** is very platform-neutral. Therefore, **EGenDriverDM** encapsulates this functionality and provides a standardized implementation for it across all TPC-E implementations.

A.12 EGenTxnHarness

EGenTxnHarness comprises any and/or all instantiations of:

- CBrokerVolume class excluding the **sponsor** provided implementation of CBrokerVolumeDB (see **EGenSourceFile** TxnHarnessBrokerVolume.h)
- CCustomerPosition class excluding the **sponsor** provided implementation of CCustomerPositionDB (see **EGenSourceFile** TxnHarnessCustomerPosition.h)
- CDataMaintenance class excluding the **sponsor** provided implementation of CDataMaintenanceDB (see **EGenSourceFile** TxnHarnessDataMaintenance.h)
- CMarketFeed class excluding the **sponsor** provided implementation of CMarketFeedDB (see **EGenSourceFile** TxnHarnessMarketFeed.h)
- CMarketWatch class excluding the **sponsor** provided implementation of CMarketWatchDB (see **EGenSourceFile** TxnHarnessMarketWatch.h)
- CSecurityDetail class excluding the **sponsor** provided implementation of CSecurityDetailDB (see **EGenSourceFile** TxnHarnessSecurityDetail.h)
- CTradeCleanup class excluding the **sponsor** provided implementation of CTradeCleanupDB (see **EGenSourceFile** TxnHarnessTradeCleanup.h)
- CTradeLookup class excluding the **sponsor** provided implementation of CTradeLookupDB (see **EGenSourceFile** TxnHarnessTradeLookup.h)
- CTradeOrder class excluding the **sponsor** provided implementation of CTradeOrderDB (see **EGenSourceFile** TxnHarnessTradeOrder.h)
- CTradeResult class excluding the **sponsor** provided implementation of CTradeResultDB (see **EGenSourceFile** TxnHarnessTradeResult.h)
- CTradeStatus class excluding the **sponsor** provided implementation of CTradeStatusDB (see **EGenSourceFile** TxnHarnessTradeStatus.h)
- CTradeUpdate class excluding the **sponsor** provided implementation of CTradeUpdateDB (see **EGenSourceFile** TxnHarnessTradeUpdate.h)

A.13 Functional Implementation

The following diagram gives a high level overview of a sample implementation of the TPC-E environment. A number of details have been omitted for clarity.

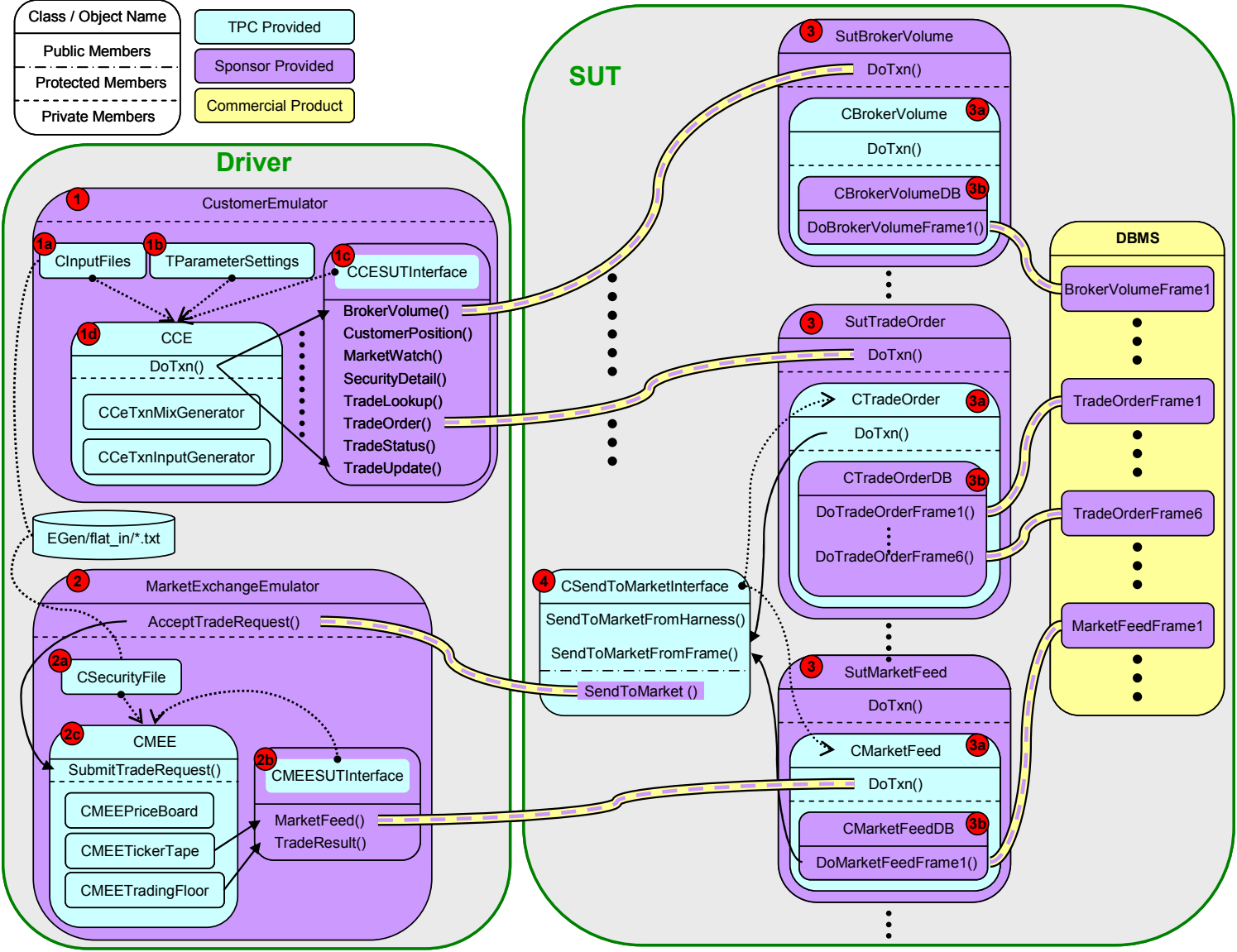


Figure 10.a - High Level Overview of a Sample Implementation

1. The **Test Sponsor** is responsible for implementing a **Customer-Emulator** per Clauses 5.8.4 and A.9.
 - a. CInputFiles is a class provided as part of **EGen** used for loading into memory the **EGenInputFiles** used by other classes in **EGen**. The **Test Sponsor** is responsible for instantiating a CInputFile object correctly and passing a pointer to it into the CCE constructor. See EGen/inc/InputFlatFilesStructure.h.
 - b. TParameterSettings is a TPC provided structure that can be used to alter the runtime behavior of **EGenDriver**. Use of this structure for a compliant run is not required; it is provided to facilitate prototyping and engineering work. See EGen/inc/DriverParamSettings.h.
 - c. CCESUTInterface is a TPC provided pure virtual class that defines an interface used by the CCE class. It is the **sponsor's** responsibility to subclass CCESUTInterface and provide the necessary implementation. This implementation is responsible for sending a **transaction** request to the **SUT**, measuring the **transaction's Response Time** and logging all necessary data. A pointer to the **sponsor's** implementation of the CCESUTInterface must be passed into the CCE constructor. See EGen/inc/CESUTInterface.h.
 - d. CCE is a TPC provided class that must be used when implementing a **Customer-Emulator**. It is the **sponsor's** responsibility to provide pointers to a CInputFile object and CCESUTInterface object when constructing the CCE object. The process of running a test is effectively looping around a call to CCE::DoTxn(). When DoTxn() is called, the CCE object will determine which **transaction** to perform, generate the necessary input data for the **transaction** and pass that data to the **sponsor's** implementation of CCESUTInterface for execution. See EGen/inc/CE.h.
2. The **Test Sponsor** is responsible for implementing a **Market-Exchange-Emulator** per Clauses 5.8.5 and A.10.
 - a. CSecurityFile is a class provided as part of **EGen** used for loading EGen/flat_in/SecurityFile.txt into memory. The **Test Sponsor** is responsible for instantiating a CSecurityFile object and passing a pointer to it into the CMEE constructor. See EGen/inc/SecurityFile.h.
 - b. CMEEESUTInterface is a TPC provided pure virtual class that defines an interface used by the CMEE class. It is the **sponsor's** responsibility to subclass CMEEESUTInterface and provide the necessary implementation. This implementation is responsible for sending a **transaction** request to the **SUT**, measuring the **transaction's Response Time** and logging all necessary data. A pointer to the **sponsor's** implementation of the CMEEESUTInterface must be passed into the CMEE constructor. See EGen/inc/MEESUTInterface.h.
 - c. CMEE is a TPC provided class that must be used when implementing a **Market-Exchange-Emulator**. It is the **sponsor's** responsibility to provide pointers to a CSecurityFile object and CMEEESUTInterface object when constructing the CMEE object. During a **Test Run**, the **sponsor's Market-Exchange-Emulator** is responsible for accepting requests from the **sponsor's** SendToMarket implementation running on the **SUT** and passing these requests to the CMEE object via SubmitTradeRequest(). In addition, the **sponsor's Market-Exchange-Emulator** is responsible for keeping a timer and calling CMEE::GenerateTradeResult() as necessary. See EGen/inc/MEE.h.

3. The **Test Sponsor** is responsible for implementing functionality on the **SUT** to accept **transaction** request over a network connection from the **sponsor's** **CCESUTInterface** and **CMEESUTInterface** implementations. Note that the diagram depicts individual network connections for each **transaction** type but the **sponsor** is free to implement a single connection capable of handling any/all types of **transactions**. Upon receiving a **transaction** request from the **Driver**, the **sponsor's** code is responsible for calling **DoTxn()** on the appropriate **EGenTxnHarness** object (3a). After returning from the call to **DoTxn()** the **sponsor's** code is responsible for sending the **transaction's** output back to the **Driver**.

See **EGen/inc/TxnHarnessBrokerVolume.h** – **TxnHarnessTradeUpdate.h**.

The **sponsor** is responsible for providing implementations for the following classes used by **EGenTxnHarness**.

- **CBrokerVolumeDB**
 - **CCustomerPositionDB**
 - **CMarketFeedDB**
 - **CMarketWatchDB**
 - **CSecurityDetailDB**
 - **CTradeLookupDB**
 - **CTradeOrderDB**
 - **CTradeResultDB**
 - **CTradeStatusDB**
 - **CTradeUpdateDB**
- These classes are responsible for implementing the **Frames** invoked by **EGenTxnHarness**.
4. **CSendToMarketInterface** is a TPC provided class that includes a pure virtual member function **SendToMarket()**. The **sponsor** is responsible for subclassing **CSendToMarketInterface** and providing an implementation for **SendToMarket()**. This implementation is responsible for sending trade requests to the **sponsor's MEE** implementation running on the **Driver**. A pointer to the **sponsor's** implementation of **CSendToMarketInterface** must be passed into the constructor for the **EGenTxnHarness** objects **CTradeOrder** and **CMarketFeed**.


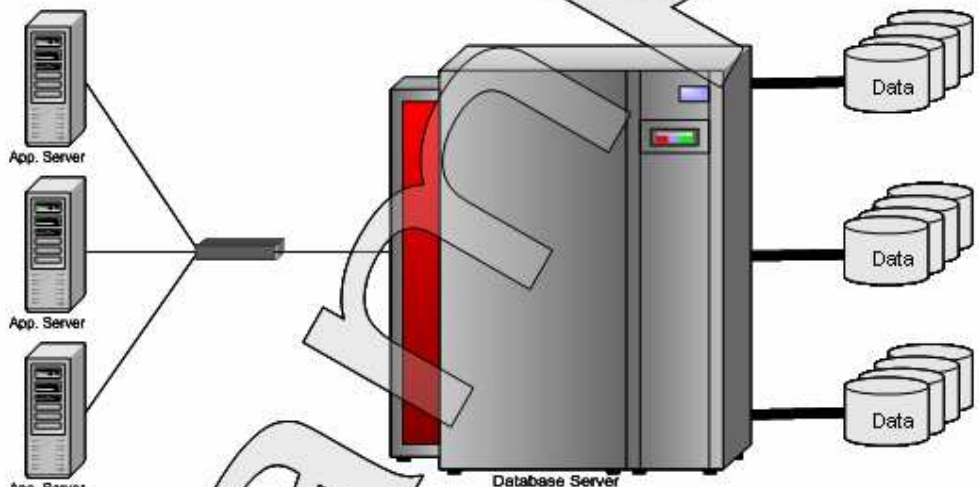
APPENDIX B. EXECUTIVE SUMMARY STATEMENT


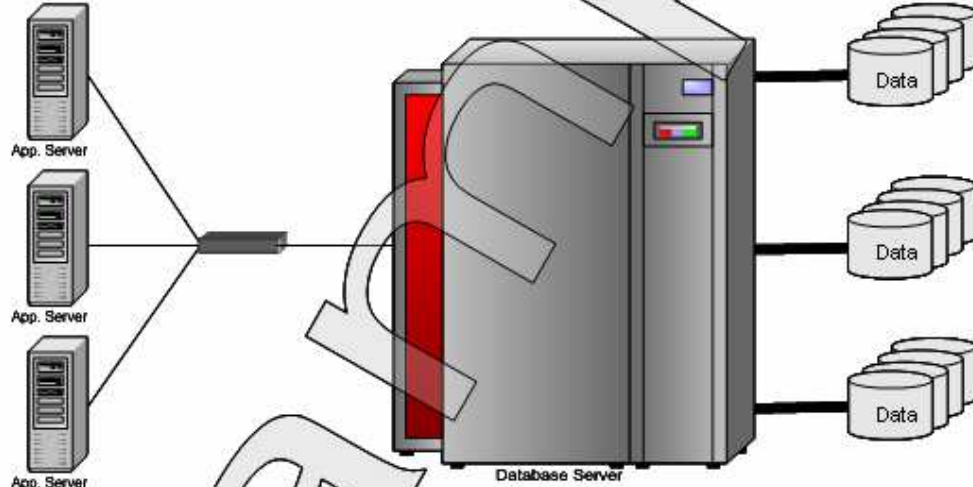
B.1 Layout Requirements

Sponsor (3b)	System Configuration (3b)		TPC-E xx.y.z (2b) TPC Pricing xx.y.z (2b)	
			Report Date January XX, XXXX (1) Revised Date May XX, XXXX (1)	
TPC-E Throughput (2) XX,XXX tpsE (2b)	Price/Performance (2) \$ X.XX USD per tpsE (2b)	Availability Date (2) August XX, XXXX (2b)	Total System Cost (2) \$ XXX,XXXX (2b)	
Database Server Configuration				
Operating System (2) (2b)	Database Manager (2) (2b)	Processors/Cores/ Threads (2) (2b)	Memory (2) (2b)	
<div style="display: flex; justify-content: space-between;"> <div style="width: 60%;"> <p>-Place Configuration Diagram Here</p> <p>-Diagram must include the following:</p> <ul style="list-style-type: none"> - Graphic representation of Database Server - Graphic representation of the disk subsystem - Graphic representation of the Client System(s) - List of system components in Database Server <ul style="list-style-type: none"> - Processors (quantity and type) - Disk Controllers (quantity and type) - Network Interface Cards (quantity and type) - List of system components included in Client System(s) <ul style="list-style-type: none"> - Processors (quantity and type) - Memory - Disk Controllers (quantity and type) - Disk Drives (quantity and type) - Other components <p>- Font</p> <ul style="list-style-type: none"> - Times New Roman or Arial <p>- Size</p> <ul style="list-style-type: none"> - Minimum 12 point type, normal </div> <div style="width: 35%; background-color: #f2f2f2; padding: 10px; border: 1px solid black;"> <p><u>Style Legend:</u></p> <p>Font - Times New Roman or Arial</p> <p>Size - (1) 10 point type, normal</p> <ul style="list-style-type: none"> - (2) 12 point type, normal - (2b) 12 point type, bold - (3) 14 point type, normal - (3b) 14 point type, bold <p>Outside box is 2 points wide</p> <p>Interior lines are 1 point wide</p> </div> </div>				
Initial Database Size (2) (2b)	Redundancy Level: (2) (2b)	Storage (2) (2b)		

Sponsor (3b)	System Configuration (3b)	TPC-E xx.y.z (2b) TPC Pricing xx.y.z (2b)
		Report Date January XX, XXXX (1) Revised Date May XX, XXXX (1)
Primary Metrics TPC-E Throughput: XX,XXX tpsE (2) (2b) Availability Date: August XX, XXXX (2) (2b) Price/Performance: \$ X.XX USD per tpsE (2) (2b) Total System Cost: \$ XXX,XXXX (2) (2b)		
Database Server Configuration Operating System: (2) (2b) Processors: (2) (2b) Database Manager: (2) (2b) Memory: (2) (2b)		
<p>-Place Configuration Diagram Here</p> <p>-Diagram must include the following:</p> <ul style="list-style-type: none"> - Graphic representation of Database Server - Graphic representation of the disk subsystem - Graphic representation of the Client System(s) - List of system components in Database Server <ul style="list-style-type: none"> - Processors (quantity and type) - Disk Controllers (quantity and type) - Network Interface Cards (quantity and type) - List of system components included in Client System(s) <ul style="list-style-type: none"> - Processors (quantity and type) - Memory - Disk Controllers (quantity and type) - Disk Drives (quantity and type) - Other components <p>- Font</p> <ul style="list-style-type: none"> - Times New Roman or Arial <p>- Size</p> <ul style="list-style-type: none"> - Minimum 12 point type, normal <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p><u>Style Legend:</u></p> <p>Font - Times New Roman or Arial</p> <p>Size - (1) 10 point type, normal</p> <ul style="list-style-type: none"> - (2) 12 point type, normal - (2b) 12 point type, bold - (3) 14 point type, normal - (3b) 14 point type, bold <p>Outside box is 2 points wide Interior lines are 1 point wide</p> </div>		
Initial Database Size (2) (2b)	Redundancy Level: (2) (2b)	Storage (2) (2b)

B.2 Sample Executive Summary Statement

		Super Widget Model RY01234		TPC-E 1.0.0 TPC Pricing 1.0.0 Report Date May 5, 2005
TPC-E Throughput 396.04 tpsE	Price/Performance \$143.10 USD per tpsE	Availability Date June 5, 2005	Total System Cost \$ 58,672 USD	
Database Server Configuration				
Operating System ACME OS V2.4	Database Manager ACME RDBMS with Fix Pack 2	Processors/Cores/ Threads 16/16/16	Memory 32GB	
<div></div> <div><div>3 x Client Systems each with - Hardware Components<ul style="list-style-type: none">- 2 x 3.0 GHz CPU w/1MB L2 Cache- 2 x 256MB DDR-RAM- 1 x 18GB 10K Internal Drives- 1 x Internal SCSI Controllers- 2 x Gigabit Ethernet NIC- Software Components<ul style="list-style-type: none">- ACME Client OS V2.4</div><div>Super Widget Server - Hardware Components<ul style="list-style-type: none">- 16 x 3.0 GHz CPU w/1MB L2- 1 x Internal SCSI Controllers- 2 x Gigabit Ethernet NIC</div></div>				
Initial Database Size 5,432 GB		Redundancy Level: 1 RAID-5		Storage 12 x 72GB 15K

	Super Widget Model RY01234		TPC-E 1.0.0 TPC Pricing 1.0.0 Report Date: May 5, 2005
Primary Metrics TPC-E Throughput: 396.04 tpsE Price/Performance: \$ 143.10 USD per tpsE			
		Availability Date:	June 5, 2005
		Total System Cost:	\$ 56,672 USD
Database Server Configuration Operating System: ACME OS V2.4 Database Manager: ACME RDBMS with Fix Pack 2			
		Memory:	32GB
		Processors/ Cores/Threads:	16/16/16
			
3 x Client Systems each with - Hardware Components <ul style="list-style-type: none">- 2 x 3.0 GHz CPU w/1MB L2 Cache- 2 x 256MB DDR RAM- 1 x 18GB 10K Internal Drives- 1 x Internal SCSI Controllers- 2 x Gigabit Ethernet NIC - Software Components <ul style="list-style-type: none">- ACME Client OS V2.4		Super Widget Server - Hardware Components <ul style="list-style-type: none">- 16 x 3.0 GHz CPU w/1MB L2- 1 x Internal SCSI Controllers- 1 x 18GB 10K Internal Disk- 2 x Gigabit Ethernet NIC	
Initial Database Size 5,432 GB	Redundancy Level: 1 RAID-5	Storage 12 x 72GB 15K	

The Really Big Computer Company	Super Widget Model RY01234				TPC-E 1.0.0 TPC Pricing 1.0.0		
					Report Date	December 1, 2005	
					Revision Date	February 1, 2006	
					Availability Date	March 1, 2006	
Description	Part Number	Price Source	Unit Price	Qty	Extended Price	Discounted Price	3 Yr. Maint. Price
Server Hardware							
R090214,0MB,CD-ROM,Mouse	201-A	1	2,761	1	2,761	2,761	1,254
R090214 Dual CPU Upgrade card, 1MB Cache	25657	1	376	8	3,008	3,008	479
4GB DIMMs (4x1GB) For ECC Memory Board	82038	1	753	8	6,024	6,024	0
Dual Port Gigabit Ethernet NIC	C654	1	50	2	100	100	34
PCI to SCSI Dual-Channel Host Bus Adapter	1111	1	175	2	350	350	155
ECC Memory Board	77016	1	116	1	116	116	50
Sub-Total					12,359	12,359	1,972
Server Storage							
18GB 10,000 RPM Disk (Included in Server)	1134	1	0	1	0	0	0
72GB 15,000 RPM Disk	12009-HB	1	856	12	10,272	10,272	4,093
Disk Enclosure	78900D	1	340	3	1,020	1,020	405
SCSI Cable, 68P HD-68P HD, 10FT	1E210	3	22	4	88	88	0
Sub-Total					11,380	11,380	4,498
Server Software							
ACME Operating System V2.4	K985	2	1,550	1	1,550	1,550	0
ACME DBMS	K743	2	9,328	1	9,328	9,328	0
ACME ServicePlus for DBMS (with 45% Discount)	K467	2	0	1	0	0	5,947
Sub-Total					10,878	10,878	5,947

Client HardwareR2385,2x3.0GHz CPU,512MB,18GB
Disk,CD-ROM,Mouse

4428 1 1,324 3 3,972 3,972 165

Gigabit Ethernet NIC A427D 1 35 6 210 210 0

Sub-Total 4,182 4,182 165**Client Software**

ACME Client Operating System V2.4 K986 1 976 3 2,928 2,928 2,549

Sub-Total 2,928 2,928 2,549**Infrastructure**

14" EPA SVGA Monitor 26019 1 88 4 352 352 72

101-Key Keyboard, Power Cord G6001A-A 1 13 4 52 52 0

8-Port Gigabit Switch A2736 4 835 1 835 835 278

Sub-Total 1,239 1,239 350**Other Discounts*** (1,365) (464)**Total** 42,966 41,601 15,017**Notes:**

* Basis for discounts: Server storage discounted 12% by dollar volume; All Acme sourced products and services discounted by 3%.

Price Source: 1=Really Big Computer Company, 2 =Acme, 3=Cables R Us, 4=Networks and More

Audited by Howe, Dewey, Cheatem Auditors, LLP

Three-Year Cost of Ownership: \$ 56,672 USD**TPC-E Throughput: 396.04 tpsE****Price/Performance: \$ 143.10 USD**

Prices used in TPC benchmarks reflect the actual prices a customer would pay for a one-time purchase of the stated components. Individually negotiated discounts are not permitted. Special prices based on assumptions about past or future purchases are not permitted. All discounts reflect standard pricing policies for the listed components. For complete details, see the pricing sections of the TPC benchmark specifications. If you find that the stated prices are not available according to these terms, please inform the TPC at pricing@tpc.org. Thank you.

Numerical Quantities Summary				
Reported Throughput:			396.04 tpsE	
Response Times (in seconds)	Minimum	Average	90 th %tile	Maximum
Broker-Volume	0.000	0.291	0.441	1.261
Customer-Position	0.012	0.333	0.393	1.443
Market-Feed	0.221	0.452	0.552	0.872
Market-Watch	0.324	0.692	0.731	2.011
Security-Detail	0.220	0.361	0.671	1.542
Trade-Lookup	0.251	0.374	0.704	2.111
Trade-Order	0.011	0.362	0.452	0.934
Trade-Result	0.202	0.401	0.481	1.932
Trade-Status	0.001	0.152	0.300	0.552
Trade-Update	0.334	0.401	0.561	0.931
Data-Maintenance	0.492	0.531	0.662	11.171
Transaction Mix (in percent of total transactions)		Transaction Count		Mix
Broker-Volume		256,639		4.90
Customer-Position		3,992,162		13.00
Market-Feed		285,154		1.00
Market-Watch		5,417,934		18.00
Security-Detail		4,277,316		14.00
Trade-Lookup		2,281,235		8.00
Trade-Order		2,880,060		10.10
Trade-Result		2,851,544		10.00
Trade-Status		5,703,088		19.00
Trade-Update		570,309		2.00
Data-Maintenance		120		N/A
Test Duration and Timings				
Ramp-up Time (hh:mm:ss)		00:15:40		
Measurement Interval (hh:mm:ss)		2:00:00		
Business Recovery Time (hh:mm:ss)		2:35:40		
Total Number of Transactions Completed in Measurement Interval		28,515,441		