TPC EXPRESS BENCHMARK ™ V (TPCx-V)

Specification

Revision 1.0.1

February 2016

Transaction Processing Performance Council (TPC)

www.tpc.org

info@tpc.org

Legal Notice

The TPC reserves all right, title, and interest to this document and associated source code as provided under U.S. and international laws, including without limitation all patent and trademark rights therein.

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

No Warranty

Trademarks

Acknowledgments

TPC Membership
(as of February 2016)

Full Members

Associate Members

Document Revision History

Date	Version	Description
12-Nov-2015	1.0.0	Initial TPCx-V standard
24-Feb-2016	1.0.1	Add definitions for VM2 and VM3; minor general cleanups

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
Italics	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.
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 with down diagonal hashing
Broker	Pale Blue with up diagonal hashing
Market	Rose with horizontal hashing
Implementation	
TPC Provided Code	*Turquoise Italics*
Sponsor Provided Code	Lavender Underline
Commercially Available Product	Light Yellow

Table of Contents

Table of Figures

CLAUSE 0 -- <u>PREAMBLE</u>

0.1   Introduction

TPC Express Benchmark V (TPCx-V) is an On-Line Transaction Processing (OLTP) workload utilizing the latest technology for providing multiple concurrent operating and application environments running on a platform. It is a mixture of read-only and update intensive transactions distributed across multiple computing environments simulating the activities found in a conglomeration of complex OLTP application environments. The database schema, data population, transactions, and implementation rules have been designed to be broadly representative of modern OLTP systems running in complex virtualized environments. 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;
- Multiple concurrently executing and isolated operating environments;
- Heterogeneous resource requirements across operating environments;
- Dynamic workload requirements across operating environments;
- Flexible resource allocation;
- 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;
- A mixture of heterogeneous and homogenous database and application environments;
- Multiple databases with many tables with a wide variety of sizes, attributes, and relationships with realistic content;
- Contention on data access and update;
- Stringent Quality of Service requirements.

The TPCx-V operations are modeled as follows:

- The operating environments and their databases are continuously available 24 hours a day, 7 days a week, for data processing from multiple **Sessions** with full access to the data in all tables, except possibly during infrequent maintenance **Sessions**.
- Consolidation of multiple database and application environments utilizing virtual operating environments to fully utilize system capabilities while limiting operating costs.
- Due to the worldwide nature of the application modeled by the TPCx-V benchmark, any of the transactions may be executed against its database at anytime.

0.1.1   Goal of the TPCx-V benchmark

The TPCx-V 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 TPCx-V is a "business throughput" measure of the number of completed Trade-Result transactions processed per second (see Clause 5.7.1). Multiple **Transactions** are used to simulate the business activity of processing a trade, and each **Transaction** is subject to a **Response Time** constraint. The **Performance Metric** for the benchmark is expressed in transactions-per-second-V (tpsV). To be compliant with the TPCx-V standard, all references to tpsV **Results** must include the tpsV rate, the associated **price-per-tpsV**, and the **Availability Date** of the **Priced Configuration** (See Clause 5.7.3 for more details).

Although this specification defines the implementation in terms of a relational data model, 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.

TPCx-V uses terminology and metrics that are similar to other benchmarks, originated by the TPC and others. Such similarity in terminology does not imply that TPCx-V **Results** are comparable to other benchmarks. The only benchmark **Results** comparable to TPCx-V are other TPCx-V **Results** that conform to a comparable version of the TPCx-V 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 TPCx-V 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, TPCx-V should not be used as a substitute for specific customer application benchmarking when critical capacity planning and/or product evaluation decisions are contemplated.

Benchmark **Sponsors** are permitted various possible implementation 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 Clause9, 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., TPCx-V 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 effective version 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 **Results**.
- 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.

0.4   TPCx-V Kit and Licensing

The TPCx-V kit is available from the TPC. User must sign-up and agree to the TPCx-V User Licensing Agreement (ULA) to download the kit. Re-distribution of the kit is prohibited. All related work (such as collaterals, papers, derivatives) must acknowledge the TPC and include TPCx-V copyright. The TPCx-V Benchmark includes: TPCx-V Specification document (this document), TPCx-V Users Guide documentation, and the TPCx-V **Benchmark Kit**, which consists of Java and C++ code to execute the benchmark load, and various scripts to set up the benchmark environment. The test sponsor is required to run the TPC-provided kit as per Section 12 of TPC policies, which describes the requirements for Express Benchmarks. See Clause 1.5 for details.

CLAUSE 1 -- <u>BENCHMARK OVERVIEW</u>

1.1   Definitions

GENERAL _____

tpsV

tpsV is the primary performance metric for TPCx-V.

A _____

ACID

ACID stands for the transactional properties of Atomicity, Consistency, Isolation and Durability.

Add

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

Application

The term **Application** or **Application Program** refers to code that is not part of the commercially available components of the **SUT**, but used specifically to implement the **Transactions** (see Clause 3.3) of this benchmark. For example, stored procedures, triggers, and referential integrity constraints are considered part of the **Application Program** when used to implement any portion of the **Transactions**, but are not considered part of the **Application Program** when solely used to enforce integrity rules (see Clause 10.4) or transparency requirements (see Clause 10.5) independently of any **Transaction**.

Application Recovery

Application Recovery is the process of recovering the business application after a **Single Point of Failure** and reaching a point where the business meets certain operational criteria.

Application Recovery Time

Application Recovery Time is the elapsed time between the start of **Application Recovery** and the end of **Application Recovery** (see Clause 6.5.5.5).

Arbitrary Transaction

An **Arbitrary Transaction** is a **Database Transaction** that executes arbitrary operations against the database at a minimum isolation level of L0 (see Clause 6.4.1.3).

Attestation Letter

The **Auditor**'s opinion regarding the compliance of a **Result** must be consigned in an **Attestation Letter** delivered directly to the **Sponsor**.

Audit Tools

A set of Java applications included in the **Benchmark Kit** that are run by the **Test Sponsor** to produce reports that facilitate the independent audit process.

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

Benchmark Kit

The TPCx-V **Benchmark Kit** is an Express benchmarking kit that conforms to the TPC policies, which describe the requirements for Express Benchmarks. The **Benchmark Kit** is a complete application that builds the schema, populates the database, runs the transactions, records complete run time data, post-processes the logged records to generate performance results, and validates the results against this specification. **Test Sponsors** are required to use the TPCx-V **Benchmark Kit** for reporting TPCx-V results.

BLOB(n)

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

BLOB_REF

BLOB_REF is a data type capable of referencing a **BLOB(n)** object that is stored outside the table on the **SUT**.

BOOLEAN

BOOLEAN is a data type capable of holding at least two distinct values that represent FALSE and TRUE.

Brokerage Initiated

Brokerage Initiated Transactions simulate broker interactions with the system and are initiated by the **Customer Emulator** component of the benchmark **Driver**.

Broker Tables

Broker Tables include 9 tables that contain information about the brokerage firm and broker related data.

Business Day

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

Business Recovery

Business Recovery is the process of recovering from a **Single Point of 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** (see Clause 6.5.5.9).

C _____

Catastrophic

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

CE

See **Customer Emulator**.

CHAR(n)

CHAR(n) means a character string that can hold up to n single-byte characters. Strings may be padded with spaces to the maximum length. **CHAR(n)** must be implemented using a **Native Data Type**.

Commit / Committed

Commit is a control operation that:

- Is initiated by a unit of work (a **Transaction**)
- Is implemented by the **DBMS**
- Signifies that the unit of work has completed successfully and all tentatively modified data are to persist (until modified by some other operation or unit of work)

Upon successful completion of this control operation both the **Transaction** and the data are said to be **Committed**.

Configured Customers

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

Customer Emulator

One key piece of a compliant TPCx-V **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 Transactions simulate customer interactions with the system and are initiated by the **Customer Emulator** component of the benchmark **Driver**.

Customer Tables

Customer Tables include 9 tables that contain information about the customers of the brokerage firm.

D _____

Data Accessibility

Date Accessibility is the ability to maintain database operations with full data access after the permanent irrecoverable failure of any single **Durable Medium** containing database tables, recovery log data, or **Database Metadata**.

Data-Maintenance Generator

Another key piece of a compliant TPCx-V **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**'s 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**.

Database Footprint

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

Database Interface

Database Interface is a 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 Logic

Database Logic is TPC provided **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 **DBMS**s, ranging from small systems that run on personal computers to huge systems that run on mainframes. From a technical standpoint, **DBMS**s 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 **DBMS**s include a report writer program that enables you to output data in the form of a report.

Database Metadata

Database Metadata is information managed by the **DBMS** and stored in the database to define, manage and use the database objects, e.g. tables, views, synonyms, value ranges, indexes, users, etc.

Database Recovery

Database Recovery is the process of recovering the database from a **Single Point of Failure** system failure.

Database Recovery Time

Database Recovery Time is the duration from the start of **Database Recovery** to the point when database files complete recovery.

Database Server

A **Database Server** is a commercially available product(s). **TPC** 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 an **ACID** unit of work.

Data Growth

Data Growth is 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* * tpsV * **Business Day** duration in seconds

DATE

DATE represents the data type of date with a granularity of a day and 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**.

Comment: A time component is not required but may be implemented.

DATETIME

DATETIME represents the data type for a date value that includes a time component. The date component must meet 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

See **Database Management System**

Digit

Digit means decimal digit.

Dimension Tables

Dimension Tables include 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

See **Durable**.

Durable / Durability

In general, state that persists across failures is said to be **Durable** and an implementation that ensures state persists across failures is said to provide **Durability**. In the context of the benchmark, **Durability** is more tightly defined as the **SUT**'s ability to ensure all **Committed** data persist across any **Single Point of Failure**.

Durable Medium

Durable Medium is a data storage medium that is inherently non-volatile such as a magnetic disk or tape. **Durable Media** is the plural of **Durable Medium**.

E _____

Elasticity Phase

Elasticity Phase is any one of the ten 12-minute load variation periods defined in Clause 5.2.

ENUM

ENUM(m[,n]) or **SENUM(m[,n])** means an exact numeric value (unsigned or signed, respectively). **ENUM** and **SENUM** are identical to **NUM** and **SNUM**, respectively, except that they must be implemented using a **Native Data Type** that provides exact representation of at least n **Digits** of precision after the decimal place.

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,

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

Fixed Space

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

Fixed Tables

Fixed Tables are tables that 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 **TPC-provided Transaction** logic, which is invoked as a unit of execution by the **VGenTxnHarness**. The database interactions of a **Transaction** are all initiated from within its **Frames**.

Frame Implementation

Frame Implementation is **TPC** provided functionality that accepts inputs from, and provides outputs to, **VGenTxnHarness** 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 is 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, **Database Metadata**) or not already used as formatting overhead by the **DBMS**.

Full Disclosure Report (FDR)

See **FDR**.

G _____

Group

Each **Tile** has four **Groups**, with **Groups** 1, 2, 3, and 4 contributing an average of 10%, 20%, 30%, and 40% of the total throughput of the **Tile**, respectively. Each **Group** consists of one **Tier A Virtual Machine** and two transaction-specific **Tier B Virtual Machines**.

Growing Space

Growing Space is any space used to store initially-loaded rows from the **Growing Tables** and their associated **User-Defined Objects**. It also 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 each have an initial cardinality that has a defined 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.

H _____

I _____

IDENT_T

IDENT_T is defined as **NUM(11)** and is used to hold non-trade identifiers.

Initial Database Size

Initial Database Size is any space allocated to the test database that is used to store the initial population, **Database Metadata**, **User-Defined Objects**, and any space used as formatting overhead by the **DBMS**. **Initial Database Size** is measured after the database is initially loaded with the data generated by **VGenLoader**.

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 125.

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**.

Log Growth

Log Growth is 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* * tpsV * **Business Day** duration in seconds

M _____

Market Exchange Emulator

A key piece of a compliant TPCx-V **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 include 11 tables that contain information about companies, markets, exchanges, and industry sectors.

Market Triggered

Market Triggered 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 Configuration

See **System Under Test.**

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.

Measurement Interval

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

MEE

See **Market Exchange Emulator**

Modify

The word "**Modify**" indicates that the content of a TPCx-V 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 a particular type described in the specification. For example, **DATETIME** must be implemented with a built-in data type of the **DBMS** designed to store date-time information.

Network

A **Network** is defined as **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

Nominal Throughput is defined to be 2.00 **Transactions**-Per-Second-V 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 **Digits**, of which n **Digits** are to the right (after) the decimal point. The data type must be able to hold all possible values that 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 _____

On-Line

A storage device is considered **On-Line** if it is capable of providing an access time to data, for random read or update, of one second or less by the **Operating System**.

Comment: Examples of **On-Line** storage may include magnetic disks, optical disks, solid-state storage, or any combination of these, provided that the above mentioned access criteria is met.

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. Other 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
- Unixes (Solaris, AIX)
- Linux
- MS-DOS
- Mac OS
- VMS
- Netware

P _____

Part Number

See the definition of **Part Number** in the TPC Pricing Specification.

Performance Metric

The TPCx-V **Reported Throughput** as expressed in tpsV.

Priced Configuration

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

Price/Performance Metric

The TPCx-V total 3-year pricing divided by the **Reported Throughput** is price/tpsV. This is also known as the **Price/Performance Metric**.

Primary Key

A **Primary Key** is a single column or combination of columns that uniquely identifies a row. None of the columns that are part of the **Primary Key** may be nullable. A table must have no more than one **Primary Key**.

Profile

A **Profile** is the characteristics of a **Transaction**, as defined by the **Pseudo-code** and summarized by the **Database Footprint**.

Pseudo-code

Pseudo-code is a description of an algorithm that uses the structural conventions of programming languages, but omits language-specific syntax.

Q _____

R _____

Ramp-down

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

Ramp-up

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

Redundancy Level One

Redundancy Level One (Durable Media Redundancy) guarantees access to the data on **Durable Media** when a single **Durable Media** failure occurs.

Redundancy Level Three

Redundancy Level Three (Full Redundancy) includes **Redundancy Level Two** and guarantees access to the data on **Durable Media** when a single failure occurs within the **Durable Media** system, including communications between **Tier B** and the **Durable Media** system.

Redundancy Level Two

Redundancy Level Two (Durable Media Controller Redundancy) includes **Redundancy Level One** and guarantees access to the data on **Durable Media** when a single failure occurs in the storage controller used to satisfy the redundancy level or in the communication media between the storage controller and the **Durable Media**.

Reference

The word "**Reference**" indicates that the TPCx-V 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 **VGenTxnHarness**.

Referential Integrity

Referential Integrity preserves the relationship of data between tables, by restricting actions performed on **Primary Keys** and **Foreign Keys** in a table.

Remove

The word "**Remove**" indicates that a number of rows are removed from the TPCx-V 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

The **Performance Metric** reported by TPCx-V is the **Reported Throughput**. The name of the metric used for the TPCx-V **Reported Throughput** of the **SUT** is tpsV**.** The value of this metric is based on the **Measured Throughput** and is bound by the limits defined in Clause 5.7.1.2.

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

Results

TPCx-V **Results** are the **Performance Metric**, **Price/Performance Metric**.

Return

The word "**Return**" indicates that the TPCx-V table column is referenced and that its content is retrieved from the database and passed to the **VGenTxnHarness**. 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

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 **ENUM(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** is the number of required customer rows per single **Transactions-Per-Second-V.** The **Scale Factor** for **Nominal Throughput** is 500.

Scaling Tables

Scaling Tables each have a defined 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.

SENUM

ENUM(m[,n]) or **SENUM(m[,n])** means an exact numeric value (unsigned or signed, respectively). **ENUM** and **SENUM** are identical to **NUM** and **SNUM**, respectively, except that they must be implemented using a **Native Data Type** that provides exact representation of at least n **Digits** of precision after the decimal place.

Session

See **Database Session**.

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**.

Comment: 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 is the period of time from the end of the **Ramp-up** to the start of the **Ramp-down**.

Substitution

Substitution is defined as a deliberate act to replace components of the **Priced Configuration** by the **Test Sponsor** as a result of failing the availability requirements of the TPC Pricing Specification or when the **Part Number** for a component changes.

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

Performance over a given period of time (computed as the average throughput over that time) is considered **Sustainable** if it shows no significant variations.

SUT

See **System Under Test**.

System Under Test

System Under Test (SUT) is the total collection of all hardware and software components in all **Tiles**, to include their **Tier A** and **Tier B** Virtual Machines.

T _____

Test Run

A **Test Run is** 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 one-minute average tpsV 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 one-minute average throughput in tpsV (computed as the total number of Trade-Result Transactions that complete within each one-minute interval divided by 60). A plot interval size of 1 minute must be used. The Ramp-up, **Steady State**, **Measurement Interval**, and **Ramp-down** must be identified on the graph. The **Test Run Graph** must be reported in the **Report**.

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**.

Tier A

Tier A consists of all hardware and software needed to implement the down-stream Connector, **VGenTxnHarness**, **Frame Implementation** and **Database Interface** functional components. The **VM** that implements **Tier A** is referred to as **VM1**.

Tier B

Tier B consists of all hardware and software needed to implement the **Database Server** functional components, encapsulated within two transaction-specific **Virtual Machines**, contained within the same **Group**. This includes data storage media sufficient to satisfy the initial database population requirements of Clause 2.4.1 and the **Business Day** growth requirements of Clause 5.6.6.4 and Clause 5.6.6.5. **Tier B** is implemented in two **VMs**: **VM2** receives the two Decision Support-type queries, and **VM3** receives the 7 remaining OLTP transactions.

Tile

Tile is the unit of replication of TPCx-V configuration and load distribution. Each **Tile** consists of 4 **Groups**. A valid TPCx-V configuration may have between 1 and 6 **Tiles**, with all **Tiles** contributing identical proportions of the total load. The number of **Tiles** and the number of **Load Units** configured in the initial populations of the databases in each **Group** are dependent on the **Nominal Throughput**, and are determined by a formula defined in Clause 4.3.4.

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.)

TPC Defined Interface

A **TPC Defined Interface** is a C++ class member that is designed to exchange data (and transfer execution control) between various components of the TPC provided **Benchmark Kit**.

TRADE_T

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

Transaction(s)

The TPCx-V **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 **VGenTxnHarness**.

Transaction Mix

The Transaction Mix is composed of all Customer Initiated, Brokerage Initiated and Market Triggered Transactions.

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

The **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.

User-Defined Object

Any object defined in the database is considered a **User-Defined Object**, except for the following:
- a TPCx-V Table (see clause 2.2.3)
- a required **Primary Key** (see clause 2.2.3.1)
- a required **Foreign Key** (see clause 2.2.3.2)
- a required constraint (see clause 2.2.3.3)
- **Database Metadata**

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**.

VALUE_T

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

VGen

VGen is a TPC provided software environment that is used in the TPC provided **Benchmark Kit** implementation of the TPCx-V benchmark. The software environment is logically divided into three packages: **VGenProjectFiles**, **VGenInputFiles**, and **VGenSourceFiles**. The software packages provide functionality to use: **VGenLoader** to generate the data used to populate the database, **VGenDriver** to generate transactional data and **VGenTxnHarness** to control frame invocation.

VGenDriver

VGenDriver comprises the following parts:

- **VGenDriverCE** provides the core functionality necessary to implement a **Customer Emulator**.
- **VGenDriverMEE** provides the core functionality necessary to implement a **Market Exchange Emulator**.
- **VGenDriverDM** provides the core functionality necessary to implement the **Data-Maintenance Generator**.

VGenDriver provides core transactional functionality (e.g. **Transaction Mix** and input generation) necessary to implement a **Driver**.

VGenDriverCE

VGenDriverCE is any and/or all instantiations of the CCE class (see **VGenSourceFiles** CE.h and CE.cpp).

VGenDriverDM

VGenDriverDM is the single instantiation of the CDM class (see **VGenSourceFiles** DM.h and DM.cpp).

VGenDriverMEE

VGenDriverMEE is any and/or all instantiations of the CMEE class (see **VGenSourceFiles** MEE.h and MEE.cpp).

VGenInputFiles

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

VGenLoader

VGenLoader is a binary executable, generated by using the methods described in **VGenProjectFiles** with source code from **VGenSourceFiles**, including any extensions by a **Test Sponsor** (see Clause 10.7.6.4). When executed, **VGenLoader** uses **VGenInputFiles** to produce a set of data that represents the initial state of the TPCx-V database.

VGenLogger

VGenLogger logs the initial configuration and any re-configuration of **VGenDriver** and **VGenLoader**, and compares current configuration with the TPCx-V prescribed defaults.

VGenProjectFiles

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

VGenSourceFiles

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

VGenTables

VGenSourceFiles contain class definitions that provide abstractions of the TPCx-V tables. These table classes are known collectively as **VGenTables** and they encapsulate the functionality needed to generate the data for each of the TPCx-V tables.

VGenValidate

VGenValidate is a binary executable, generated by using methods described in **VGenProjectFiles** with source code from **VGenSourceFiles**. The **Benchmark Kit** uses **VGenValidate** to validate that the **Sponsor's Measurement Interval** had compliant Trade-Results per **Load Unit**.

VGenTxnHarness

VGenTxnHarness defines a set of interfaces that are used to control the execution of, and communication of inputs and outputs, of **Transactions** and **Frames**.

Virtual Machine (VM)

A **Virtual Machine (VM)** is a self-contained operating environment, managed by the **VMMS**, and that behaves as if it were a separate computer (as defined in Clause 10.1.1.3). TPCx-V requires that there shall be three **VM**s per **Group**: one **Tier A VM** and two transactional specific **Tier B VM**s.

Virtual Machine Management Software (VMMS)

Commonly referred to as a Hypervisor, **Virtual Machine Management Software (VMMS)** is a commercially available framework or methodology of dividing the resources of a system into multiple computing environments. Each of these computing environments allows a completely isolated software stack including an operating system to run in complete isolation from anything else running on the system. The **VMMS** allows for the creation of multiple computing environments on the same system.

Comment: The term **VMMS** is not meant to include the static partitioning of a system that occurs at boot time or any dynamic partitioning that may take place through operator intervention

VM1

A **Virtual Machine (VM)** that implement the **Tier A** functionality of a **Group**.

VM2

A **Virtual Machine (VM)** that is a component of the **Tier B** of a **Group,** and executes the two Decision Support queries.

VM3

A **Virtual Machine (VM)** that is a component of the **Tier B** of a **Group**, and executes the 7 OLTP transactions.

W _____

X _____

Y _____

Z _____

1.2 Business and Application Environment

TPC Express Benchmark™ V 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 to help users relate intuitively to the components of the benchmark.

A typical business requires multiple applications to manage a variety of operations. Often these applications have been located on separate systems. With advances in virtualization technologies and in the strength of computing resources, it is now possible to co-locate these applications on the same system.

While it may be possible to install and use multiple applications in a single system image, there can be advantages to maintaining the applications in distinct virtual machines (VMs):

- Duplicate applications may require separation of data to serve multiple regions or customer sets;
- Dissimilar applications may have some duplicate naming challenges where separation is desirable;
- It may be desirable to restrict the user group of one application from accessing data used by another application;
- There may be accounting reasons for identifying the amount of computing resources required by each application;
- There may be a desire to isolate maintenance operations of each application, so as not to disrupt service on other applications;
- There may be a need to separate the application interface to end users from the interface to the database, as is found in many 3-tiered application environments.

In short, depending on the size of the business and the size of the system used, the business model of TPCx-V may be viewed as a "Data Center in a Box", with a wide variety of applications, including both database tiers and application-management tiers all residing on logically distinct VMs within a single computer system. The following diagram illustrates the potential complexity of the business model portrayed in the benchmark.
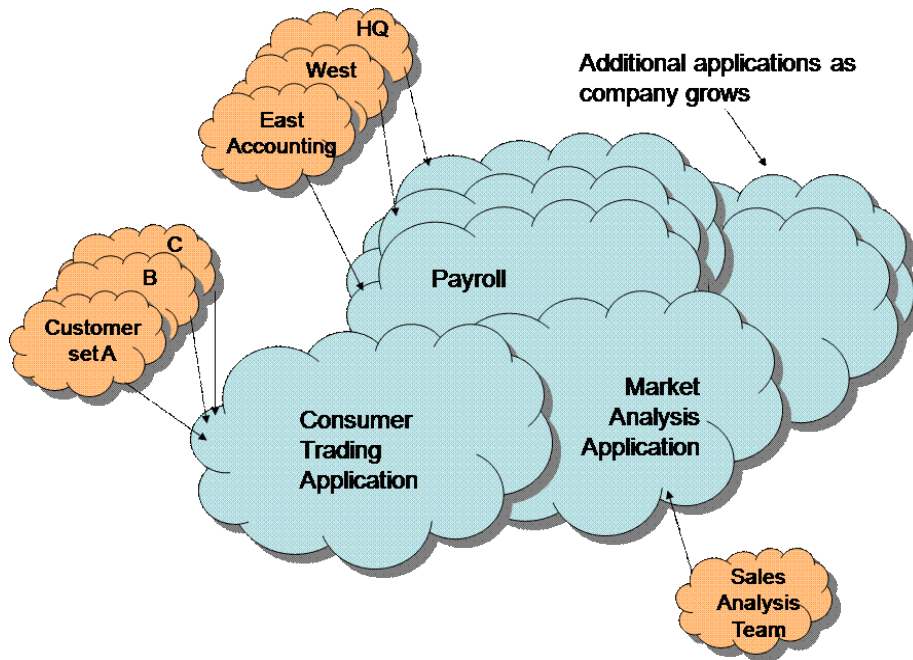
Business Model: Data Center in a Box

However, the complexities of the modeled environment do not lend itself well for a measureable, repeatable performance benchmark. Consequently, the TPCx-V benchmark application is a simplified view of this complex environment – retaining most of the key features of the business model, while enhancing the ability to provide meaningful and comparable benchmark results.

The following diagram represents this simplified view:
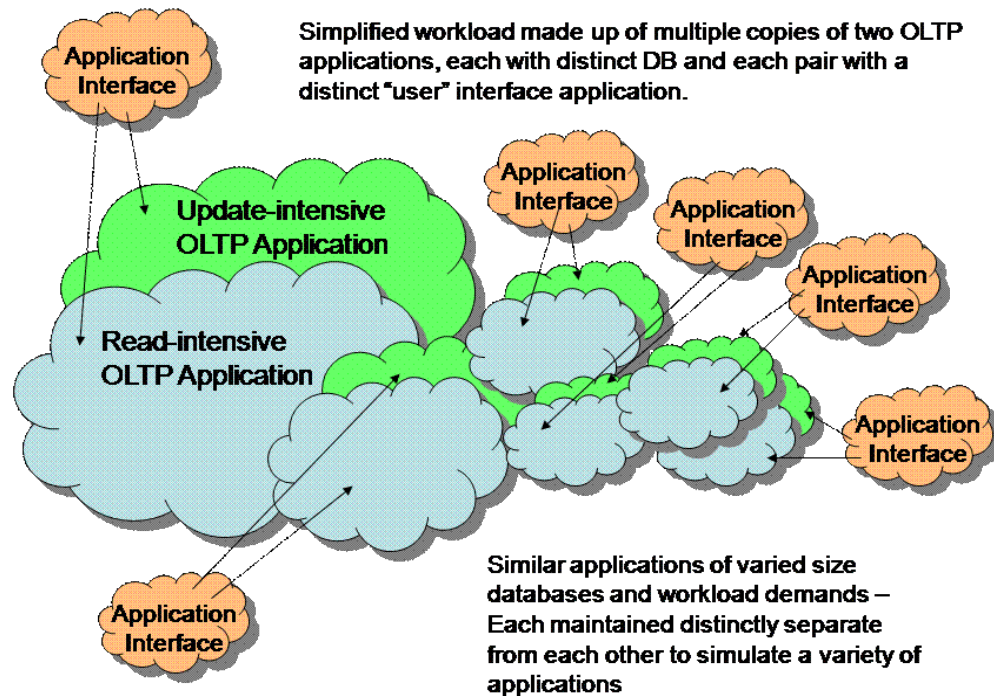
Simplified VM Components

The benchmark has been reduced to simplified form of the virtualized environment. Each group of Application Interface, Update-Intensive and Read-Intensive VMs is a distinct "**Group**". A **Tile** comprises four **Groups**, with 1 to 6 identical **Tiles** per configuration. The total load on the system determines the size of each **Tile** and the number of **Tiles**. **Tiles** are logically distinct from each other from an application perspective, although the benchmark driver may coordinate the amount of work being required of each **Tile**.

Note: To provide a meaningful application environment with database components and transactions that are relevant and understandable, the application environment defined for the TPC-E benchmark is employed. TPC-E is altered to provide the desired read-intensive and update-intensive environments, shown above. While TPC-E uses a business model of a brokerage house with transactions driven from multiple sources, the deployment of the adjusted application in TPCx-V is intended to represent a wide variety of OLTP-based applications that could be employed in a virtualized computing environment.

There is one other critical aspect to the business model for a virtualized environment. This is the concept of workload dynamics. Performance benchmarks are typically measured in "steady state", where the flow of work requests is adjusted to meet the capabilities of the system. For a single application, this can provide a satisfactory answer, but not for a virtualized environment.

The following diagram illustrates the existence of workload dynamics in the business model for TPCx-V. Each application may vary between the minimum and maximum requirements, depending on such things as time zone, time of day, time of year or introduction of a new product. To accommodate each of the four applications represented on separate systems, the total compute power required is represented by the "Total Separate" bar. However, in the chosen business model, the peak workload demands for each application are not simultaneous. One workload may be at a peak when another is at a valley, allowing computer resources to be shifted from the low-use application to the high-use one for some period of time, and shifting the resources to another high-demand application at a subsequent point. This allows the total configured capacity to be more like the bar marked "Virtualized."
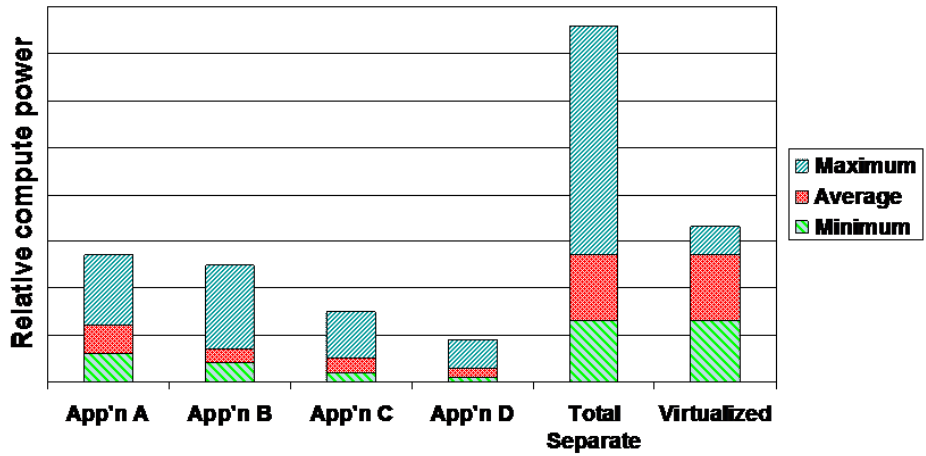
Demands by workload

In the environment modeled by the benchmark, the dynamic nature of each workload could be dictated by a wide variety of influences that result in an unpredictable shifting of resources and an equally unpredictable amount of overall system output. As with the complexity of the modeled application environment, this level of workload dynamics is not easily repeated to deliver comparable measurements. Since the primary requirement of the virtualized environment for this situation is the ability to dynamically allocate resources to the VMs that are in high demand, it is sufficient to define a workflow time line that shifts workload demands among the VMs in a predictable manner, as illustrated, below. 0 is for demonstration purposes. Clause 5.2 specifies the actual number and properties of the **Elasticity Phases**.
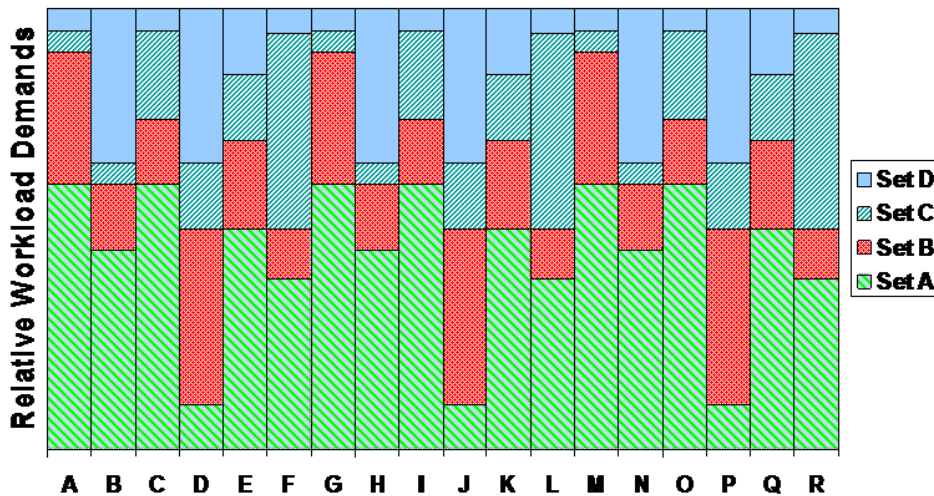
Elasticity Phases

TPCx-V 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. TPCx-V 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:
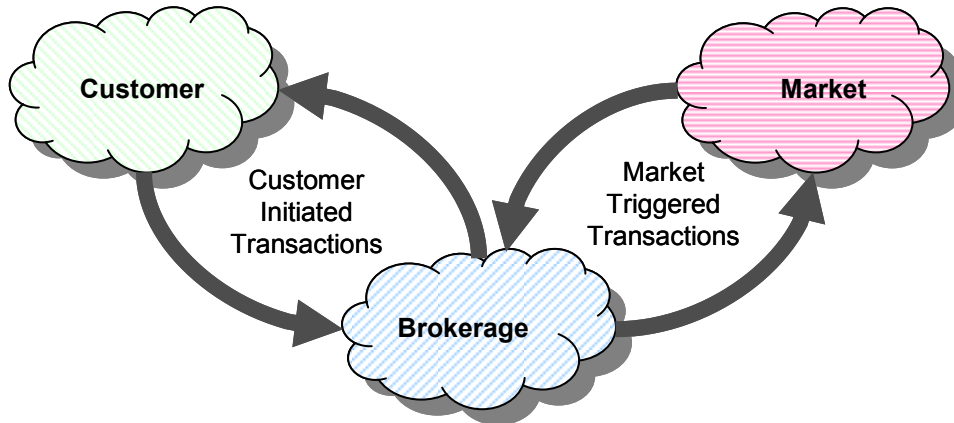
Business Model Transaction Flow

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 TPCx-V.

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 turn 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.

The TPCx-V 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:

Invoke the following transactions …

READ-WRITE	READ-ONLY	
•Market-Feed	•Broker-Volume	•Security-Detail
•Trade-Order	•Customer-Position	•Trade-Lookup
•Trade-Result	•Market-Watch	•Trade-Status
•Trade-Update		

… against the following data

Application Components
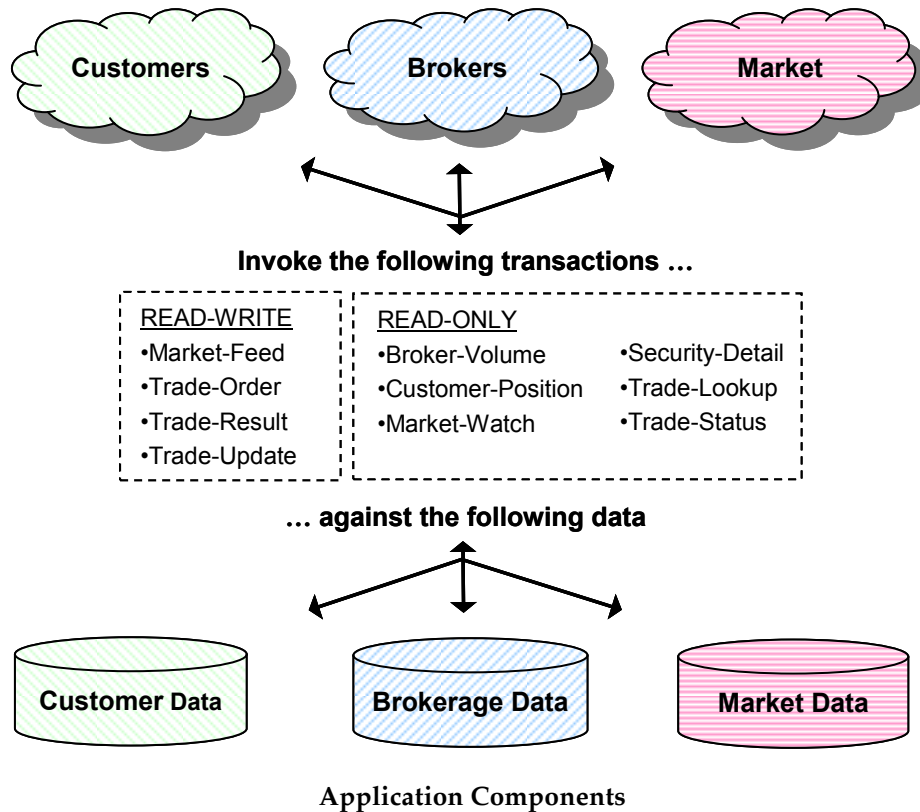
The benchmark has been reduced to simplified form of the application environment. To measure the performance of the OLTP system, a simple **Driver** generates **Transactions** and their inputs, submits them to the **System Under Test**, and measures the rate of completed **Transactions** being returned. 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 server side of a transaction monitor or application server.

1.3   Transaction Summary

1.3.1   Broker-Volume

The Broker-Volume **Transaction** is designed to emulate a brokerage house's "up-to-the-minute" internal business processing. An example of a Broker-Volume **Transaction** would be a manager generating a report on the current performance potential of various brokers.

1.3.2   Customer-Position

The Customer-Position **Transaction** is designed to emulate the process of retrieving the customer's profile and summarizing their overall standing based on current market values for all assets. This is representative of the work performed when a customer asks the question "What am I worth today?"

1.3.3   Market-Feed

The Market-Feed **Transaction** is designed to emulate the process of tracking the current market activity. This is representative of the brokerage house processing the "ticker-tape" from the market exchange.

1.3.4 Market-Watch

The Market-Watch **Transaction** is designed to emulate the process of monitoring the overall performance of the market by allowing a customer to track the current daily trend (up or down) of a collection of securities. The collection of securities being monitored may be based upon a customer's current holdings, a customer's watch list of prospective securities, or a particular industry.

1.3.5 Security-Detail

The Security-Detail **Transaction** is designed to emulate the process of accessing detailed information on a particular security. This is representative of a customer doing research on a security prior to making a decision about whether or not to execute a trade.

1.3.6 Trade-Lookup

The Trade-Lookup **Transaction** is designed to emulate information retrieval by either a customer or a broker to satisfy their questions regarding a set of trades. The various sets of trades are chosen such that the work is representative of:

- performing general market analysis
- reviewing trades for a period of time prior to the most recent account statement
- analyzing past performance of a particular security
- analyzing the history of a particular customer holding

1.3.7 Trade-Order

The Trade Order **Transaction** is designed to emulate the process of buying or selling a security by a Customer, Broker, or authorized third-party. 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 at a requested price. 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 before either submitting or canceling the trade.

1.3.8 Trade-Result

The Trade-Result **Transaction** is designed to emulate the process of completing a stock market trade. This is representative of a brokerage house receiving from the market exchange the final confirmation and price for the trade. The customer's holdings are updated to reflect that the trade has completed. Estimates generated when the trade was ordered for the broker commission and other similar quantities are replaced with the actual numbers and historical information about the trade is recorded for later reference.

1.3.9 Trade-Status

The Trade-Status **Transaction** is designed to emulate the process of providing an update on the status of a particular set of trades. It is representative of a customer reviewing a summary of the recent trading activity for one of their accounts.

1.3.10 Trade-Update

The Trade-Update **Transaction** is designed to emulate the process of making minor corrections or updates to a set of trades. This is analogous to a customer or broker reviewing a set of trades, and discovering that some minor editorial corrections are required. The various sets of trades are chosen such that the work is representative of:

- reviewing general market trends
- reviewing trades for a period of time prior to the most recent account statement
- reviewing past performance of a particular security

1.3.11 Data-Maintenance

The Data-Maintenance **Transaction** is designed to emulate the periodic modifications to data that is mainly static and used for reference. This is analogous to updating data that seldom changes.

1.3.12 Trade-Cleanup

The Trade-Cleanup **Transaction** is used to cancel any pending or submitted trades from the database.

1.4 Model Description

1.4.1 Entity Relationships

1.4.1.1 Trading in TPCx-V is done by Accounts. Accounts belong to Customers. Customers are serviced by Brokers. Accounts trade Securities that are issued by Companies.

1.4.1.2 The total set of Securities that can be traded is 6,850 and the total set of Companies is 5,000. For each Company, there is one common share, plus 0-4 preferred shares.

1.4.1.3 All Companies belong to one of the 102 Industries. Each Industry belongs to one of the 12 market Sectors.

1.4.1.4 Each Account picks its average of ten Securities to trade from across the entire range of Securities.

1.4.1.5 Securities to be traded can be identified by the security symbol or by the company name and security issue.

1.4.2 Differences between Customer Tiers

1.4.2.1 The basic scaling unit of a TPCx-V 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.

1.4.2.2 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.

1.4.2.3 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 any **Test Run**.

1.4.3 Customer Partitioning

1.4.3.1 TPCx-V scales with Customers. It is conceivable that Customer information could be partitioned into groups of related Customers. This is called Customer Partitioning. The advantage of Customer Partitioning is that it increases locality of reference within each sub-group of Customers. Transactions can be directed to a subset of Customers or to the entire set of Customers, in a defined proportion.

1.4.4 Trade Types

1.4.4.1 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%).

1.4.4.2 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%).

1.4.4.3   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.

1.4.4.4   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 6 minutes based on **VGenDriverMEE** 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.

1.4.5   Effects of Trading on Holdings

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

1.4.5.2   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).

1.4.5.3   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.

1.4.5.4   Before **VGenLoader** runs, there are no trades and no positions in any security for any account. **VGenLoader** simulates running the benchmark for 125 **Business Days** of initial trading, so that the initial database will be ready for benchmark execution.

1.4.5.5   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).

1.4.5.6   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.

1.5   TPCx-V Benchmark Kit

1.5.1   Kit Contents

The TPCx-V kit contains the following components:

- The TPCx-V User's Guide

- Java and C++ code to implement the driver and the database access code in Tier A

- DML (stored procedures) to implement the body of transactions

- DDL (including shell scripts) to create the schema and populate the database

- Various bash scripts to run a test, produce the Executive Summary, validate the results and perform basic tasks outlines in Clause 9 -- , and collect statistics to assist the **test sponsor** in tuning the configuration.

1.5.2 DBMS

PostgreSQL version 9.3 is the database used by the TPCx-V **Benchmark Kit**. The **Test Sponsor** may choose to use newer, released versions of PostgreSQL when they become available.

1.5.3 Kit Usage

To submit a compliant TPCx-V benchmark result, the **test sponsor** is required to use the TPCx-V kit as provided, except for modifications explicitly listed in

The kit must be used as outlined in the TPCx-V User's Guide.

The output of the TPCx-V kit is called the **run report**, which includes the following

1. Executive Summary
2. Validation and audit files
3. Supporting file

1.5.4 If there is a conflict between the TPCx-V specification and the TPC provided code, the TPC provided code prevails.

1.5.5 Configuration Files

The TPCx-V **Benchmark Kit** reads the VM network (NetBIOS) names, port numbers, database sizes, **Measurement Interval** Length, etc. from the configuration file vcfg.properties. The file testbed.properties has the **SUT** information used in producing the **Executive Summary Statement** at the completion of a **Test Run**.

The contents of vcfg.properties and testbed.properties that are included in the **Benchmark Kit** are generic, and need to be changed by the **Test Sponsor** to conform to the actual **System Under Test**. These two files are the only parts of the **Benchmark Kit** that the **Test Sponsor** is permitted to modify.

1.5.6 Addressing Errors in the TPCx-V Benchmark Kit

If a **Test Sponsor** must correct an error in the TPCx-V **Benchmark Kit** 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 1.5.7, no later than the time when the **Result** is submitted.

4. The error and the modification used to correct the error must be **reported** in the **FDR**, as described in clause 8.4.4.1.

5. 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.

1.5.7 Process for Reporting Issues with the TPCx-V Benchmark Kit

The TPCx-V **Benchmark Kit** has been tested on a variety of platforms. None-the-less, it is impossible to guarantee that the TPCx-V **Benchmark Kit** is functionally correct in all aspects or will run correctly on all platforms. It is the **Test Sponsor's** responsibility to ensure the TPCx-V **Benchmark Kit** runs correctly in their environment(s).

1.5.7.1 Portability Issues

If a **Sponsor** believes there is a portability issue with the TPCx-V **Benchmark Kit**, 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 a TPCx-V **Benchmark Kit** 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 the TPCx-V **Benchmark Kit** update
- The TPC issuing a formal waiver documenting the allowed changes to the TPCx-V **Benchmark Kit.** In the event a waiver is issued and used by the **Sponsor**, certain documentation policies apply (see Clause 8.4.4.1).

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.

1.5.7.2   Other Issues

For any other issues with the TPCx-V **Benchmark Kit**, 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 a TPCx-V **Benchmark Kit** issue not related to portability. The **Sponsor** must provide return contact information (e.g. Name, Address, Phone number, Email).

1.5.8   Submitting TPCx-V Benchmark Kit Enhancement Suggestions

As a result of using the TPCx-V **Benchmark Kit**, **Test Sponsors** may have suggestions for enhancements. To submit a suggestion the **Sponsor** must:

1. Document the exact nature of the proposed enhancement
6. Document any proposed implementation for the enhancement
7. Contact the TPC Administrator with the above specified documentation (hard or soft copy is acceptable) and clearly state that this is a TPCx-V **Benchmark Kit** 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**.

1.5.9   Future Kit Releases

If a **test sponsor** would like a future release of the TPCx-V **Benchmark Kit** to include new scripts or changes to existing script, then the **test sponsor** can donate the scripts or script code changes to the TPC, and work with the TPC to incorporate them in the next release.

If a **test sponsor** would like to see changes made to the Java or C++ code of the kit, then the changes should be provided to the TPC for potential inclusion in the next release of the TPCx-V **Benchmark Kit**.

Comment: It is the intention of the TPC to encourage contribution of code that fixes bugs or allows the benchmark to run in new environments, and the Council will strive to release such changes with an accelerated release schedule. Java and C++ code changes that alter the characteristics of the kit will need to go through a rigorous testing and prototyping phase before approval by the Council.

CLAUSE 2 -- <u>DATABASE DESIGN, SCALING & POPULATION</u>

2.1   Introduction

The TPCx-V database is defined to consist of 33 separate and individual tables. Each VM in a **Group** shall contain all of these tables even though some tables may not be referenced by the transactions that are executed on that VM. The tables shall be scaled according to the contribution of that **Group** to the overall throughput as defined in Clause 2.6. Each VM has a schema independent of other VMs. The database schema is organized into four sets of tables:

- **Customer Tables** include 9 tables that contain information about the customers of the brokerage firm.

- **Broker Tables** include 9 tables that contain information about the brokerage firm and broker related data.

- **Market Tables** include 11 tables that contain information about companies, markets, exchanges, and industry sectors.

- **Dimension Tables** include 4 dimension tables that contain common information such as addresses and zip codes.

The relationship between the tables and the requirements governing their use are outlined in the remaining sections of Clause 2.

2.1.1   Definitions

2.1.1.1   A **Primary Key** is a single column or combination of columns that uniquely identifies a row. None of the columns that are part of the **Primary Key** may be nullable. A table must have no more than one **Primary Key**.

2.1.1.2   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.

2.2   TPCx-V Database Schema and Table Definitions

Details of the TPCx-V database schema, the data type requirements, the required structure of each individual table, the entity relationship between tables and the individual column restrictions are defined in this clause.

2.2.1   Data Type Definitions

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

2.2.1.2   **CHAR(n)** means a character string that can hold up to n single-byte characters. Strings may be padded with spaces to the maximum length. **CHAR(n)** must be implemented using a **Native Data Type**.

2.2.1.3   **NUM(m[,n])** means an unsigned numeric value with at least m total **Digits**, of which n **Digits** are to the right (after) the decimal point. The data type must be able to hold all possible values that 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**.

0.1.1.1   **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.**

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

2.2.1.4   **ENUM(m[,n])** or **SENUM(m[,n])** means an exact numeric value (unsigned or signed, respectively). **ENUM** and **SENUM** are identical to **NUM** and **SNUM**, respectively, except that they must be implemented using a **Native Data Type** that provides exact representation of at least n **Digits** of precision after the decimal place.

Comment: A numeric data type provides either exact or approximate representation of numeric values. For example, INTEGER and DECIMAL are exact numeric data types and REAL and FLOAT are approximate numeric data types (based on ANSI SQL definitions).

2.2.1.5   **BOOLEAN** is a data type capable of holding at least two distinct values that represent FALSE and TRUE.

Comment: The convention in this document, as well as the implementation of **VGen**, is that the value zero (0) denotes FALSE and the value one (1) denotes TRUE.

0.1.1.2   **DATE** represents the data type of date with a granularity of a day and 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.**

Comment: A time component is not required but may be implemented.

2.2.1.6   **DATETIME** represents the data type for a date value that includes a time component. The date component must meet 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   **BLOB(n)** is a data type capable of holding a variable length binary object of n bytes.

2.2.1.8   **BLOB_REF** is a data type capable of referencing a **BLOB(n)** object that is stored outside the table on the **SUT**.

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 non-trade identifiers.

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

Trade identifiers have the following characteristics:
- They must be unique.
- They may be sparse.
- At load time they are generated by **VGenLoader**.
- At run time they are generated by **Sponsor** provided code.
- The **VGenLoader** 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 **SENUM(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 **ENUM(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 **SENUM(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 **SENUM(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 TPCx-V 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

Category	Table Name	Table Prefix	Definition
DIMENSION	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 during a **Test Run**. The **Primary Key**s are marked with PK or PK+ in the Relations field 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 during a **Test Run**. The **Foreign Keys** are marked with FK () or 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 constraints defined in this section must be enforced by the database during a **Test Run**. The constraints are listed in the Constraints column for each table definition.

Comment: Unless a Not Null constraint is present, a column must allow Null.

2.2.3.4   For each TPCx-V required table, the columns 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_

Column Name	Data Type	Constraints	Relations	Description
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(25)	Not Null		Last name of the person with access to the customer account.
AP_F_NAME	CHAR(20)	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_

Column Name	Data Type	Constraints	Relations	Description
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(25)	Not Null		Primary Customer's last name.
C_F_NAME	CHAR(20)	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_

Column Name	Data Type	Constraints	Relations	Description
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.

Column Name	Data Type	Constraints	Relations	Description
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 references per customer into the TAXRATE table. One reference is for state/province tax; the other one is for national tax. The TAXRATE table contains the actual tax rates.

Table Prefix: CX_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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 then 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).

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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(49)	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_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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_*

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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.

2.2.5.6 TRADE

The table contains information about trades.

Table Prefix: T_

Column Name	Data Type	Constraints	Relations	Description
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 0, 1		Is this trade a cash (1) or margin (0) 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.

Column Name	Data Type	Constraints	Relations	Description
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(49)	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	Not Null >= 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 0, 1		If this trade is closing an existing position, is it executed against the newest-to-oldest account holdings of this security (1=LIFO) or against the oldest-to-newest account holdings (0=FIFO).

2.2.5.7 TRADE_HISTORY

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

Table Prefix: TH_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
TR_T_ID	TRADE_T	Not Null	PK FK (T_)	Trade request identifier. This value will be used for processing the pending limit order when it is subsequently triggered.
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.

Column Name	Data Type	Constraints	Relations	Description
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_B_ID	IDENT_T	Not Null	FK (B_)	Identifies the broker handling the trade.

2.2.5.9 TRADE_TYPE

The table contains a list of valid trade types.

Table Prefix: TT_

Column Name	Data Type	Constraints	Relations	Description
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 0, 1		1 if this is a "Sell" type transaction. 0 if this is a "Buy" type transaction.
TT_IS_MRKT	BOOLEAN	Not Null in 0, 1		1 if this is a market transaction that is submitted to the market exchange emulator immediately. 0 if this is a limit transaction.

The contents of the TRADE_TYPE table are shown below for readability, since the TT_ID values are used elsewhere in the specification.

TT_ID	TT_NAME	TT_IS_SELL	TT_IS_MRKT
TLB	Limit-Buy	0	0
TLS	Limit-Sell	1	0
TMB	Market-Buy	0	1
TMS	Market-Sell	1	1
TSL	Stop-Loss	1	0

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_

Column Name	Data Type	Constraints	Relations	Description
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.

Column Name	Data Type	Constraints	Relations	Description
CO_SP_RATE	CHAR(4)	Not Null		Company's credit rating from Standard & Poor.
CO_CEO	CHAR(46)	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_

Column Name	Data Type	Constraints	Relations	Description
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. **VGenLoader** will load this table with data for each security for the period starting 3 January 2000 and ending 31 December 2004.

Table Prefix: DM_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
EX_ID	CHAR(6)	Not Null	PK	Exchange identifier. Values are, "NYSE", "NASDAQ", "AMEX", "PCX".
EX_NAME	CHAR(100)	Not Null		Exchange name.

Column Name	Data Type	Constraints	Relations	Description
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. **VGenLoader** 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_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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.

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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	BLOB(100000) or BLOB_REF	Not Null		Large object containing the news item or links to the story.
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_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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 includes 4 dimension tables that contain common information such as addresses and zip codes.

2.2.7.1 ADDRESS

This table contains address information.

Table Prefix: AD_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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".

The contents of the STATUS_TYPE table are shown below for readability, since the ST_ID values are used elsewhere in the specification.

ST_ID	ST_NAME
ACTV	Active
CMPT	Completed
CNCL	Canceled
PNDG	Pending
SBMT	Submitted

2.2.7.3   TAXRATE

The table contains information about tax rates.

Table Prefix: TX_

Column Name	Data Type	Constraints	Relations	Description
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_

Column Name	Data Type	Constraints	Relations	Description
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

The Data Definition Language (DDL) statements contained in the TPCx-V **Benchmark Kit** create the schema to conform to this specification. After creating disk space to hold the data, the test sponsor must run the VDb/pgsql/scripts/linux/setup.sh shell script, which creates and populates the schema on the provided disk space. This section describes what rules are followed by the DDL that implements the schema. The only changes allowed to the implementation rules are those defined in Clauses 1.5.6 and 1.5.7.

For full details of the Implementation Rules, see Appendix 10.1.

2.4 TPCx-V Database Size and Table Cardinality

The transaction load generated to service customer accounts and to interact with financial markets drives the throughput of the TPCx-V 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 TPCx-V database size and scaling. CUSTOMER table cardinality is determined based on the transaction throughput metric requirements defined in Clause 5.6.7.

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

The TPCx-V benchmark has three types of sizing requirements for its tables:

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

- **Scaling Tables** each have a defined 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** each have an initial cardinality that has a defined 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.

Comment: The HOLDING and HOLDING_SUMMARY tables are considered **Growing Tables**. Rows are added to and deleted from the HOLDING and HOLDING_SUMMARY tables during the benchmark execution, but the average size of the tables continues to grow at an insignificant rate during **Steady State**. The TRADE_REQUEST table is also considered a **Growing Table** because it is initially empty and at runtime grows to an average size that is a fixed relationship to the transaction throughput rates and not to the cardinality of the CUSTOMER table.

2.4.1 Initial Database Size Requirements

2.4.1.1 The test database must be initially populated using data generated by **VGenLoader**. By definition, the TPC provided **VGenLoader** produces the correct number of rows for each table. The test database must be built including the initial database population and **User-Defined Objects** present immediately prior to the first **Test Run**.

2.4.1.2   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 5.6.8.5 defines the **Nominal Throughput** for a given number of rows in the CUSTOMER table. The minimum number of rows for the CUSTOMER table in each database 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**.

2.4.1.3   The overall **Load Unit** count, based on Clause 5.6.8.5, shall be proportioned among the **Groups** and **Tiles** as specified in Clause 4.3.4.2. Each of VM2 and VM3 in a **Group** must be initially populated with the same number of **Load Units**. The initial database populations of all **Group** 1 databases in all **Tiles** are required to be equal. The number of **Load Units** in the initial database population in a database in **Groups** 2, 3, and 4 must be 2, 3, and 4 times the number of **Load Units** in a **Group** 1 database, respectively. The minimum aggregate number of **Load Units** is (50 X *Tile count*) with *Tile count* calculated from formulas in Clause 4.3.4.1. Since the size of the CUSTOMER table in a **Group** 1 database may be increased only in increments of 1,000 customers, the aggregate number of **Load Units** may only be increased in increments of (10 X *Tile count*) **Load Units**.

2.4.1.4   The **Growing Tables** are populated with an initial set of rows sufficient to enable all benchmark **Transactions** to run.

2.4.1.5   The **Scale Factor** is the number of required customer rows per single **Transactions-Per-Second-V.** The **Scale Factor** for **Nominal Throughput** is 500.

2.4.1.6   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 125.

2.4.1.7   The number of **Load Units** configured in each database must be equal to the number of **Load Units** actually accessed during the **Test Run.**

2.4.1.8   The following variables are used as an aid in defining TPCx-V table cardinalities:

Variable	Table	Description
customers	CUSTOMER	Number of rows in the CUSTOMER table.
accounts	CUSTOMER_ACCOUNT	Number of rows in the CUSTOMER_ACCOUNT table. Equal to 5 * *customers*.
trades	TRADE	Number of trade rows in the TRADE table. The trades number is equal to 7200 * *customers* (125 days of initial population at **SF** = 500).
settled	SETTLEMENT	Number of settled trade rows in the SETTLEMENT table. The settled number is equal to *trades*.
companies	COMPANY	Number of rows in the COMPANY table. There are a fixed 5,000 companies.
securities	SECURITY	Number of rows in the SECURITY table. There are a fixed 6,850 securities.

2.4.1.9   The following rules are used by **VGenLoader** to calculate the cardinalities of the **Scaling Tables** and **Growing Tables**. The **VGen** package uses random number generators to set the number of rows for relationships such as securities per account and, as a result, the cardinality of some TPCx-V tables can only be approximated.

Table	Variable Used	Rule
ACCOUNT_PERMISSION	*accounts*	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 ~1.42 * *accounts*
ADDRESS	*customers*	*companies*(5,000) + EXCHANGE(4) + *customers*
BROKER	*customers*	0.01 * *customers* (or 1 broker per 100 *customers*)
CASH_TRANSACTION	*settled*	~0.92 * *settled* (84% of buys and 100% of sells are cash)
COMPANY	*companies*	1 * *companies*
COMPANY_COMPETITOR	*companies*	3 * *companies*
CUSTOMER_ACCOUNT	*customers*	5 * *customers*
CUSTOMER_TAXRATE	*customers*	2 * *customers*
DAILY_MARKET	*securities*	*securities* * 1,305 (5 years of 5-day work weeks with two leap years)
FINANCIAL	*companies*	*companies* * 20 quarters (5 years)
HOLDING	*settled*	~0.07955 * *settled* (assumes ITD = 125 and SF = 500)
HOLDING_HISTORY	*settled*	~1.3331 * *settled* (assumes ITD = 125 and SF = 500)
HOLDING_SUMMARY	*accounts*	~ 9.9234 * *accounts* (assumes ITD = 125 and SF = 500)
LAST_TRADE	*securities*	1 * *securities*
NEWS_ITEM	*companies*	2 * *companies*
NEWS_XREF	*companies*	2 * *companies*
SECURITY	*customers*	1 * *Securities*
SETTLEMENT	*settled*	1 * *settled*
TRADE	*customers*	7200 * *customers* = ((ITD * 8 * 3600) / SF) * *customers*

TRADE_HISTORY	*settled*	~((2 rows per market trade) * 0.6)
		  +
		  ((3 rows per limit trade) * 0.4)
		Average is (2.4 * *settled*)

TRADE_REQUEST		0
WATCH_LIST	*customers*	Each customer has one watch list (1 * *customers*)
WATCH_ITEM	*customers*	Average=100 items per watch list * *customers*

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

Fixed Tables	Cardinality	Cardinality Formula
CHARGE	15	5 trade types * 3 customer tiers
COMMISSION_RATE	240	4 rates * 4 exchanges * 5 trade types * 3 customer tiers
COMPANY	5,000	5,000 companies
COMPANY_COMPETITOR	15,000	3 * *companies*
DAILY_MARKET	8,939,250	1,305 days (5 years) * *securities*
EXCHANGE	4	4 exchanges

FINANCIAL	100,000	*companies* * 20
INDUSTRY	102	102 industries
LAST_TRADE	6,850	*securities* * 1
NEWS_ITEM	10,000	*companies* * 2
NEWS_REF	10,000	*companies* * 2
SECTOR	12	12 sectors
SECURITY	6,850	*securities* * 1
STATUS_TYPE	5	5 status types
TAXRATE	320	320 tax rates
TRADE_TYPE	5	5 trade types
ZIP_CODE	14,741	14,741 zip codes

2.4.1.11   The following list contains the cardinality of the **Scaling Tables** for the minimum of 5,000 customers

Scaling Tables	Cardinality	Cardinality Formula
CUSTOMER	5,000	Scaled based on transaction rate
CUSTOMER_TAXRATE	10,000	*customers* * 2
CUSTOMER_ACCOUNT	25,000	*accounts* = (5 * *customers*)
ACCOUNT_PERMISSION	~35,500	*accounts* * (Average of ~1.42 permissions per account)
ADDRESS	10,004	*companies* (5,000) + EXCHANGE (4) + *customers*
BROKER	50	*customers* * 0.01
WATCH_LIST	5,000	*customers* * 1
WATCH_ITEM	~ 500,000	*customers* * (Average of ~100 *securities* per watch list)

2.4.1.12   The following list shows the initial cardinality of the **Growing Tables** for the minimum of 5,000 customers, ITD-125, and SF=500.

Growing Tables	Cardinality	Cardinality Formula
CASH_TRANSACTION	~33,120,000	~0.92 * *settled* (84% of buys & 100% of sells are cash)
HOLDING	~2,844,000	~0.07955 * *settled* (assumes **ITD** = 125 and **SF** = 500)
HOLDING_HISTORY	~47,916,000	~1.3331 * *settled* (assumes **ITD** = 125 and **SF** = 500)
HOLDING_SUMMARY	~248,900	~9.9234 * *accounts*
SETTLEMENT	36,000,000	1 * *settled*
TRADE	36,000,000	((**ITD** * 8hr / day * 3600sec / hr * *customers*) / **SF**)
TRADE_HISTORY	~86,400,000	~(2.4 * *trades*)
TRADE_REQUEST	0	0

2.4.2 Test Run Database Size Requirements

2.4.2.1 The following list shows the increase in rows per second for the **Growing Tables** (except for TRADE_REQUEST) during a **Test Run**. The rate of growth may decline after running for a large number of days.

Table Name	Cardinality Formula
CASH_TRANSACTION	~0.92 * ($customers/$**SF**)
HOLDING	~0.040 * ($customers/$**SF**)
HOLDING_HISTORY	~1.344 * ($customers/$**SF**)
SETTLEMENT	1 * ($customers/$**SF**)
TRADE	1 * ($customers/$**SF**)
TRADE_HISTORY	~2.4 * ($customers/$**SF**)

The TRADE_REQUEST table is empty at the start of a **Test Run** and does grow at first during runtime, but it soon reaches a cardinality that is dependent on recent performance and not on the length of the **Test Run**. The approximate cardinality of TRADE_REQUEST during the **Steady State** portion of a **Test Run** can be estimated as ~24 rows * **Measured Throughput** (see Clause 5.6.8.1). Considerable variation in this cardinality is possible both while running and at the end of a **Test Run**.

2.4.2.2 The test database must be built to sustain the **Reported Throughput** 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 TPCx-V **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 **VGenTxnHarness** 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 **VGenTxnHarness**.

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

3.1.1.3   A **Frame** is the **TPC-provided Transaction** logic, which is invoked as a unit of execution by the **VGenTxnHarness**. The database interactions of a **Transaction** are all initiated from within its **Frames**.
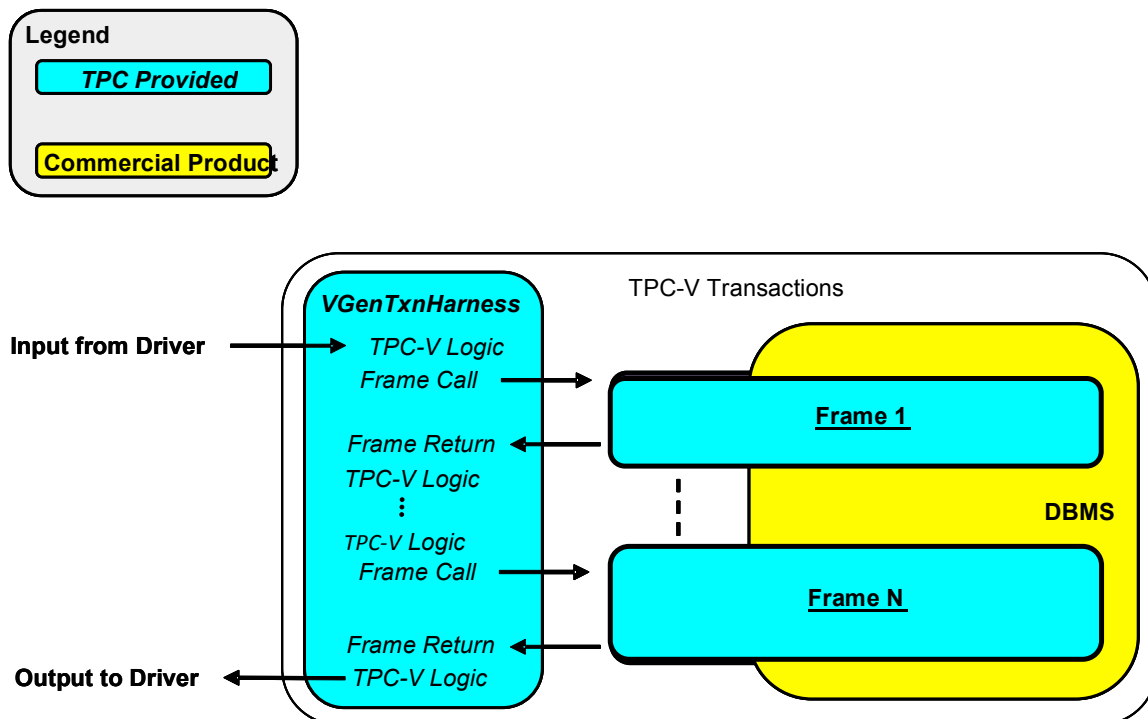
Frames Interfacing with the Harness and the Database

3.1.1.4   A **Database Transaction** is an **ACID** unit of work.

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 words "Transaction Control" that denotes the entire **Transaction**. The last row defines the overall **Transaction**.

- The second column denotes one of the following:

    o   A specific column name of a database table as defined in Clause 2.2.

    o   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.

    o   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 last row of the **Database Footprint** where the words "Transaction Control" 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:

    o   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.

    o   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 a **Database Transaction**. **Commit is** a control operation that:

- Is initiated by a unit of work (a **Transaction**)

- Is implemented by the **DBMS**

- Signifies that the unit of work has completed successfully and all tentatively modified data are to persist (until modified by some other operation or unit of work)

    o   Upon successful completion of this control operation both the **Transaction** and the data are said to be **Committed**. The explicit committing of 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:

    o The word "**Reference**" indicates that the TPCx-V 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 **VGenTxnHarness**.

    o The word "**Return**" indicates that the TPCx-V table column is referenced and that its content is retrieved from the database and passed to the **VGenTxnHarness**. 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.

    o The word "**Modify**" indicates that the content of a TPCx-V 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.

    o The word "**Add**" indicates that a number of rows are added to the TPCx-V table specified by the **Database Footprint**. TPCx-V 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.

    o The word "**Remove**" indicates that a number of rows are removed from the TPCx-V 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 **VGenTxnHarness** 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.

Comment 3: The programming semantics used to implement the required access methods for a given table column is not restricted from performing operations typically associated with a different access method, as long as the implementation of the **Frame** is functionally equivalent to the specified **Pseudo-code**. For example, "select for update" and "select with UPDLOCK" are compliant implementations of a **Reference** access method.

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 **VGen's** data generation methods or values for data elements defined in the database schema for the benchmark, except for the VGen constants listed in the table below.

Comment 1: 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**, via **Transaction** inputs or **Transaction Pseudo-code**, is prohibited.

3.2.1.2 The following table shows **VGen** constants used as limits when generating the number of values for **Transaction** inputs or when accepting **Transaction** outputs. These constant limits are provided in the specification for explicit usage in the corresponding Clause 3.3 **Frame Implementations**.

Description	Constant	Value	VGen 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	25	TxnHarnessStructs.h
Security-Detail			
Minimum number of DAILY_MARKET rows to return	min_day_len	5	TxnHarnessStructs.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-Lookup			
Maximum number of TRADE rows to return for Transaction	TradeLookupMaxRows	20	MiscConsts.h
Maximum number of TRADE rows to return for Frame 1	TradeLookupFrame1MaxRows	20	MiscConsts.h
Maximum number of TRADE rows to return for Frame 2	TradeLookupFrame2MaxRows	20	MiscConsts.h
Maximum number of TRADE rows to return for Frame 3	TradeLookupFrame3MaxRows	20	MiscConsts.h
Maximum number of TRADE _HISTORY rows to return	TradeLookupMaxTradeHistoryRowsReturned	3	MiscConsts.h
Trade-Status			
Maximum number of trade status rows to return	max_trade_status_len	50	TxnHarnessStructs.h

Trade-Update			
Maximum number of TRADE rows to return for Transaction	TradeUpdateMaxRows	20	MiscConsts.h
Maximum number of TRADE rows to return for Frame 1	TradeUpdateFrame1MaxRows	20	MiscConsts.h
Maximum number of TRADE rows to return for Frame 2	TradeUpdateFrame2MaxRows	20	MiscConsts.h
Maximum number of TRADE rows to return for Frame 3	TradeUpdateFrame3MaxRows	20	MiscConsts.h
Maximum number of TRADE _HISTORY rows to return	TradeUpdateMaxTradeHistoryRowsReturned	3	MiscConsts.h

3.2.1.3   All data exchanges between **Frames** must be done by the **VGenTxnHarness** 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 **VGenTxnHarness** and to pack the output parameters returned to the **VGenTxnHarness**.

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

3.2.1.5   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.

3.2.1.6   The implementation of each **Frame** must be functionally equivalent to the **Pseudo-code** provided for that **Frame** in Clause 3.3. Functional equivalence is satisfied when:

- For a given set of inputs the implementation produces the same outputs and causes the same change in database state as the **Pseudo-code**. A change in database state is a change to a TPCx-V Table or TPCx-V Table column, resulting from any **Modify**, **Add** or **Remove** access method defined by the **Transaction's Database Footprint**.
- All access methods in the **Database Footprint** are performed.
- No additional **Add**/**Modify**/**Remove** access methods against any TPCx-V Table are performed.

Comment: Additional **Reference** access methods against any TPCx-V Table may be performed. Additional access methods against any **User-Defined Object** may be performed.

3.2.1.7   The minimum decimal precision for any computation performed as part of the **Frame** must be the maximum decimal precision of all the individual items in that calculation.

3.2.1.8 Each **Frame** and **Transaction** has a status output parameter used to indicate the execution status of the **Frame** or **Transaction**. A status value of 0 indicates success. A negative status value indicates an error that would invalidate a **Test Run**. A positive non-zero integer value for status indicates a warning. Warnings mean that an unexpected result was generated and the **Test Sponsor** and **Auditor** should investigate the unexpected result. The unexpected result may be due to a rare but legal condition or it may be because of an incorrect implementation or run-time problem. If the latter is the cause of the warning, it must be treated as an error that invalidates the **Test Run**.

The following table shows the positive warning numbers and where they may happen in **VGen**.

Transaction	Frame	Warning Status	Reason for Warning
Trade-Lookup	2	+621	num_found == 0
Trade-Lookup	3	+631	num_found == 0
Trade-Lookup	4	+641	num_trades_found == 0
Trade-Update	2	+1021	num_updated == 0
Trade-Update	3	+1031	num_found == 0

3.2.1.9 If a transaction processing monitor (hereinafter referred to as TM) is used it must be commercially available software which provides the following features/functionality:

Operation - The TM must allow for:
- request/service prioritization
- multiplexing/de multiplexing of requests/services
- automatic load balancing
- reception, queuing, and execution of multiple requests/services concurrently

Security - The TM must allow for:
- the ability to validate and authorize execution of each service at the time the service is requested.
- the restriction of administrative functions to authorized users.

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

Recovery - The TM must have the capability to:
- post error codes to an application
- detect and terminate long-running transactions based on predefined time-out intervals

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

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

8. Client and server application programs use the same identifier (e.g., handle or pointer) to maintain the message context for multiple transactions.

9. Change and/or recompilation of the client and/or server application programs is required when the number of queues or equivalent data structures used by the TM to maintain the message context between the client and server application programs is changed by TM administration.

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

3.2.2   Customer Partitioning and Generating Transaction Inputs

3.2.2.1   If customer partitioning is being used and the **Frame** is **Customer Initiated**, **VGenDriverCE** will apply the following rules whenever generating a customer identifier, account identifier or customer tax identifier:

- 50% of the time the data is selected from the partition's range of customers.

- 50% of the time the data is selected from the entire range of customers.

If customer partitioning is not being used, or the **Frame** is not **Customer Initiated**, **VGenDriverCE** will generate customer identifiers, account identifiers and customer tax identifiers from the entire range of customers.

3.3   The Transactions

The TPCx-V 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. Nine of the **Transactions** follow a specific mix to generate the desired workload while keeping the benchmark environment simple, repeatable and easy to execute. Two additional **Transactions** are not part of the **Transaction Mix**, but are executed at fixed intervals. The tenth **Transaction**, called "Market-Feed", simulates a market ticker feed of recent stock trades. The eleventh **Transaction**, called "Data-Maintenance", simulates administrative updates to tables that are not otherwise modified by the **Transactions** in the mix.

An additional 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, TPCx-V has defined **Transactions** with read-only characteristics as well as **Transactions** with read-write characteristics. 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** can be grouped into three categories:

- **Customer Initiated Transactions** simulate customer interactions with the system and are initiated by the **Customer Emulator** component of the benchmark **Driver**.

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

- **Market Triggered Transactions** simulate the behavior of the market and are triggered by the **Market Exchange Emulator** component of the benchmark **Driver**.

Nine **Transactions** are in the mix, and in addition, the benchmark defines two time triggered **Transactions,** the Market-Feed **Transaction** and the Data-Maintenance **Transaction**, which are initiated at fixed time intervals as defined in Clause 5.3.3. Also defined is a Trade-Cleanup **transaction** (see clause 5.3.4), 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**. See Clause 10.6 for full implementation details of the **transactions**, including pseudo-code

Transaction	Weight	Access	Category	Frames	Definition
Broker-Volume	Mid to Heavy	Read-only	Brokerage Initiated	1	Clause 10.6.1
Customer-Position	Mid to Heavy	Read-only	Customer Initiated	3	Clause 10.6.2.1
Market-Feed	Light	Read-write	Market Time Triggered	1	Clause 10.6.3
Market-Watch	Medium	Read-only	Customer Initiated	1	Clause 10.6.4
Security-Detail	Medium	Read-only	Customer Initiated	1	Clause 10.6.5
Trade-Lookup	Medium	Read-only	Brokerage Initiated for Frames 1 & 3 Customer Initiated for Frames 2 & 4	4	Clause 10.6.6
Trade-Order	Heavy	Read-write	Customer Initiated	6	Clause 10.6.7
Trade-Result	Heavy	Read-write	Market Triggered	6	Clause 10.6.8
Trade-Status	Light	Read-only	Customer Initiated	1	Clause 10.6.9
Trade-Update	Medium	Read-write	Brokerage Initiated for Frames 1& 3 Customer Initiated for Frame 2	3	Clause 10.6.10
Data-Maintenance	Light	Read-write	Brokerage Time Triggered	1	Clause 10.6.11
Trade-Cleanup	Medium	Read-write	Run once before Test Run	1	Clause 10.6.12

CLAUSE 4 -- DESCRIPTION OF SUT, DRIVER, AND NETWORK

4.1   Overview

TPCx-V is a distillation of an abstraction of multiple virtualized "real-world" OLTP environment. In order to understand what TPCx-V tests and, as a consequence, what TPCx-V does not test, it is necessary to understand the base "real-world" environment, the abstraction of that base environment , and the distillation of that abstraction. For a complete description of the SUT, Driver, and Network, see Clause 10.1.

4.2   Example Test Configuration Implementations

Figure 4.a - Sample Component of Physical Test Configuration
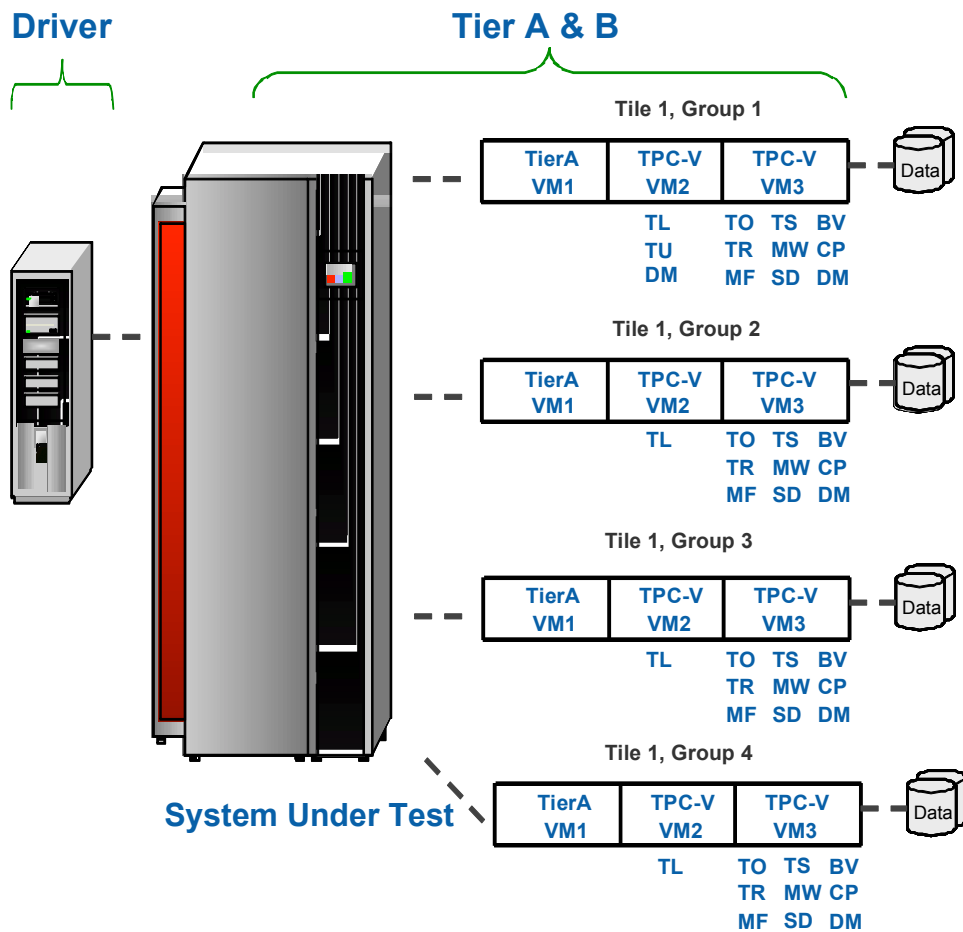
4.2.1   The following figure shows the physical components that could be assembled to implement a hypothetical test configuration.

Comment: 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 Further Requirements for SUT and Driver Implementations

4.3.1 Disclosure of Network Configuration

The **Test Sponsor** shall describe completely the **Network** configurations of both the tested services and the proposed real (target) services that are being represented.

4.3.2 Synchronization of Time

All of the systems used for the **Driver** and **SUT** must have system clocks that are synchronized to within a tolerance of 10 seconds across all systems. The synchronization must be verified once before and once after the **Test Run**.

This clause covers the constraints and regulations governing the use of **Benchmark Kit**. For detailed information on **Benchmark Kit**, what features and functionality it provides and how a **Test Sponsor** is to use those features and functionality see Clause 10 -- .

4.3.3 SUT Implementation Limits on Operator Intervention

Systems must be able to run normal operations for at least a **Business Day** without requiring any operator intervention to sustain the **Reported Throughput**.

Comment: Operator intervention is defined as any activity that requires an operator or an individual to perform a function to enable the **SUT** to continue processing **Transactions**.

4.3.4 Valid Configurations

A TPCx-V configuration is made up of several identical **Tiles**, with each **Tile** having 4 **Groups**. A **Tile** in a valid configuration will have **Groups** 1, 2, 3, and 4 contributing an average of 10%, 20%, 30%, and 40% of the total throughput of the **Tile**, respectively.

4.3.4.1 Calculation of the number of Tiles

The **Tile** counts for various **Load Unit** ranges are listed in the table below, and depicted in Figure 4.f.

Comment: The ranges are overlapping. So when a sponsor chooses the number of **Load Units** based on the corresponding **Nominal Throughput**, the table gives the sponsor either two choices for the number of **Tiles** (for example, at 500 LUs), or a single choice (for example, at 2,000 LUs).

| Aggregate LU range | | Number of | Aggregate LU |
From	To	Tiles	increment size
50	800	1	10
200	12,800	2	20
3,150	204,750	3	30
51,200	3,276,800	4	40
819,200	52,428,800	5	50
13,107,000	Infinity	6	60
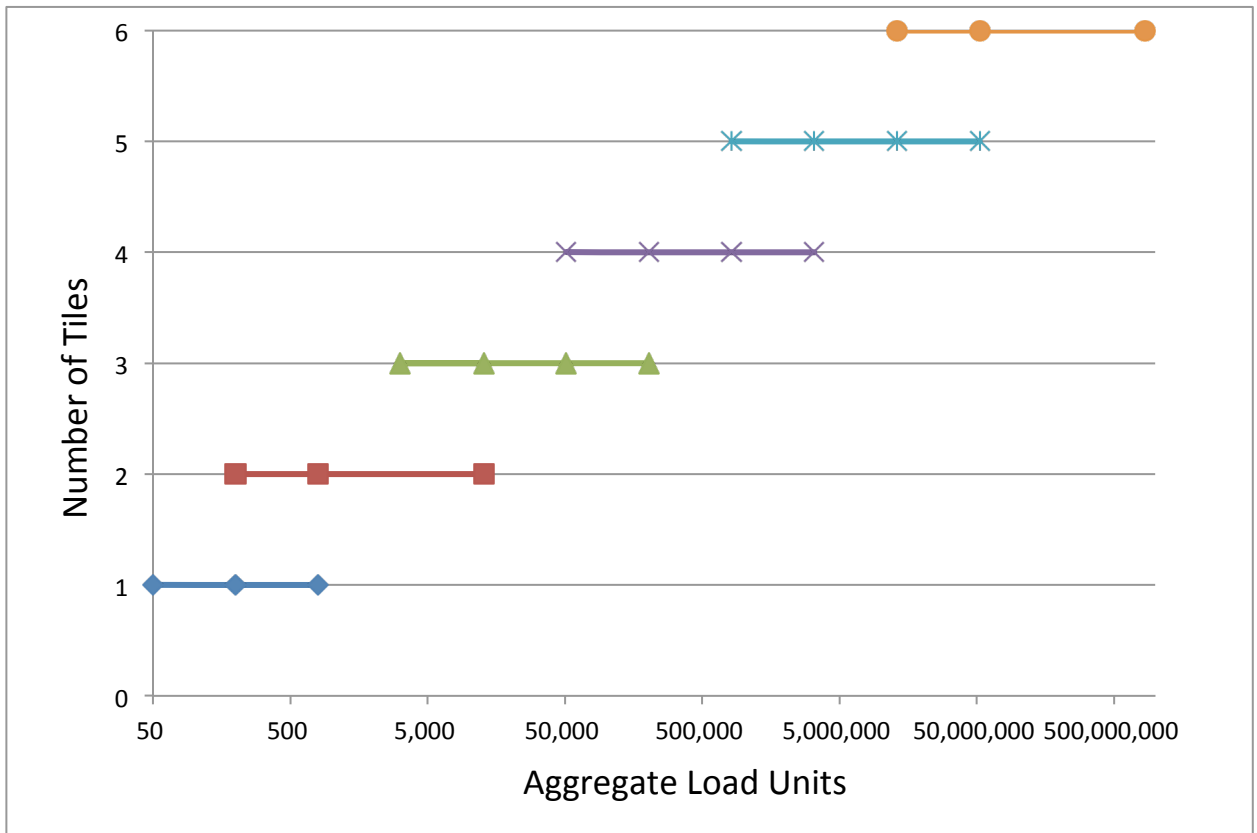
Figure 4.b – Valid number of Tiles versus aggregate LUs

The formulas below were used in the calculation of the values in the table above. However, some of the values were adjusted to make the aggregate number of LUs an integral multiple of the LU increment size. The valid number of **Tiles** for an overall tpsV rate, V, may be either:

Number of **Tiles** = $\text{FLOOR}(\text{MAX}(1,\text{LOG}_2(tpsV/100))/4,1)+1$

or

Number of **Tiles** = $\text{FLOOR}(\text{MAX}(1,\text{LOG}_2(tpsV/25))/4,1)+1$

4.3.4.2 Calculation of the number of Load Units in each Group

The overall number of **Load Units** is determined by Clause 5.6.8.5. The number of **Load Units** in each **Group** 1 in a configuration with n **Tiles** equals (overall number of **Load Units** / n) * 10%. The number of **Load Units** in each **Group** 2-4 is similarly calculated by substituting 20%, 30%, and 40%, respectively, in the formula above.

All **Groups** must be populated in accordance with the requirements in Clauses 2.4.1.2 and 2.4.1.3. Clause 2.4.1.3 specifies the minimum number of **Load Units** and the minimum **Load Unit** increment value.

CLAUSE 5 -- EXECUTION RULES & METRICS

5.1 Introduction

This clause defines the execution rules and the methods for calculating the benchmark metric.

5.1.1 Definition of Terms

5.1.1.1 The term **Reported** refers to an item that is part of the **FDR** (see Clause 8 -- for detailed requirements).

5.1.1.2 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 1: Transaction** errors are not allowed during the **Test Run**. A **Transaction** that never completes is considered an error.

> **Comment 2:** A Trade-Order **Transaction** that requires a rollback that runs successfully and produces the correct output is considered a **Valid Transaction**.

> **Comment 3:** A **Transaction** that aborts and is retried by the **SUT** and ultimately completes successfully and produces the correct output is not an error. A **Transaction** may not be retried by the **Driver**.

5.2 Dynamic Workload Variation

One of the unique features of TPCx-V is that the load of each **Group** rises or falls at every one of the 9 **Phase** changes of the **Measurement Interval**. This is intended to represent the elastic nature of workloads present in virtual systems and the resource allocation policies required to handle such elasticity. The overall load presented to the **System Under Test**, as well as the total load presented to each **Tile**, remains constant throughout the **Measurement Interval**, but the contribution from each **Group** within a **Tile** varies by as much as a factor of 7X between two *consecutive* **Elasticity Phases** (the rise of the contribution of **Group 1** from 5% to 35% in **Elasticity Phase** 7, followed by the dropping back to 5% in **Elasticity Phase** 8). In each **Phase**, all **Group** 1s of all **Tiles** vary to the same degree; and the same applies to **Groups** 2-4. The table and chart below show how much each **Group** contributes to the overall throughput of a **Tile** in each 12-minute **Elasticity Phase**.

The difference between the highest and lowest percentage of load presented to a **Group** across all 10 **Elasticity Phases** can be as much as 16X (the 80% of **Elasticity Phase** 4 of **Group** 4 to the 5% of **Elasticity Phase** 9 of that **Group**).

The Max-to-Min variation for **Group** 1 is from 35% to 5%.

The Max-to-Min variation for **Group** 2 is from 65% to 5%.

The Max-to-Min variation for **Group** 3 is from 70% to 5%.

The Max-to-Min variation for **Group** 4 is from 80% to 5%.

Elasticity Phase	Group 1	Group 2	Group 3	Group 4
1	10%	20%	30%	40%
2	5%	10%	25%	60%
3	10%	5%	20%	65%
4	5%	10%	5%	80%
5	10%	5%	30%	55%
6	5%	35%	20%	40%
7	35%	25%	15%	25%
8	5%	65%	20%	10%
9	10%	15%	70%	5%
10	5%	10%	65%	20%
Average	10%	20%	30%	40%
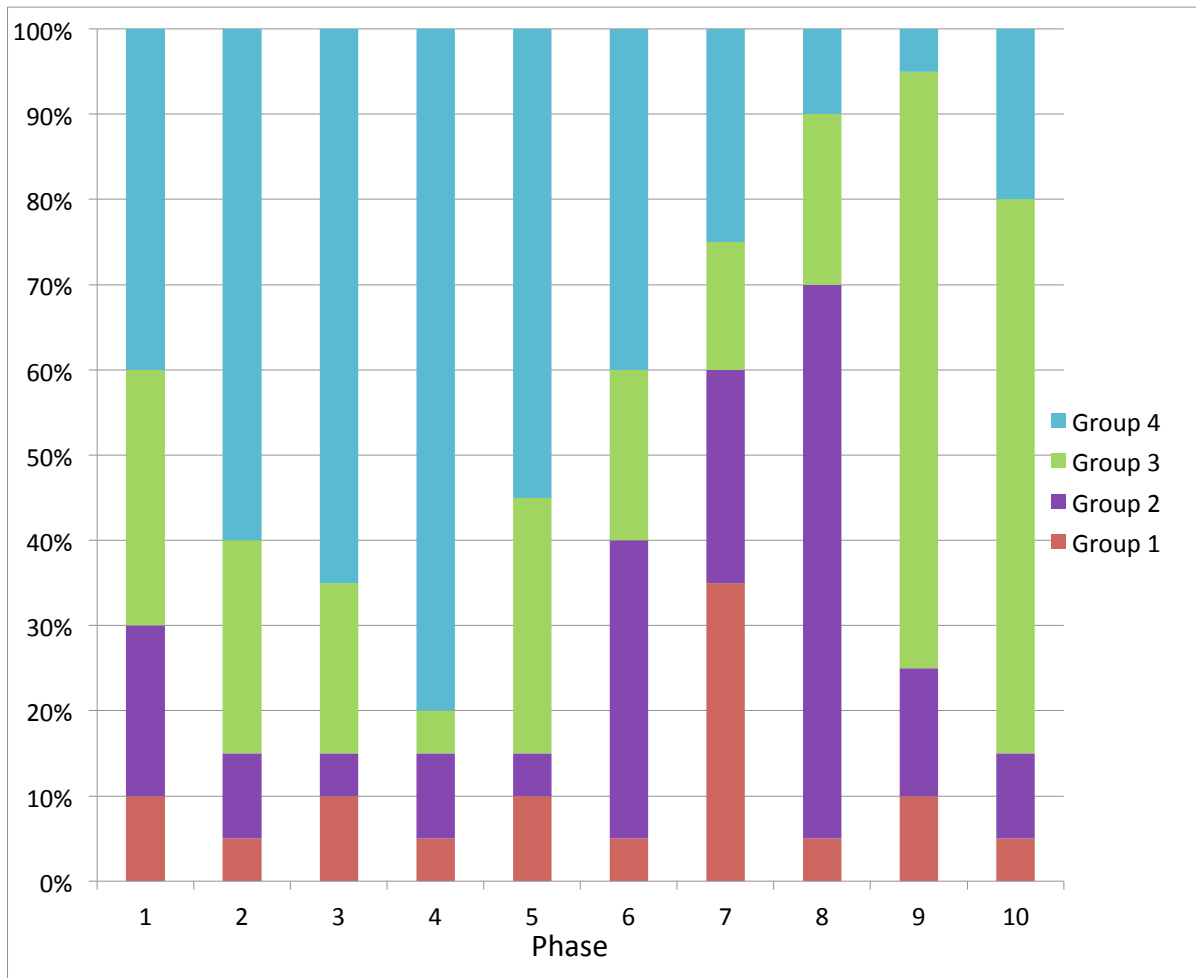
Figure 5.a - Dynamic load variation

5.3 Transaction Mix

The TPCx-V workload is made up of a number of **Transactions** acting against multiple databases following a specified **Transaction Mix**. During the **Test Run**, the CCE code controls the generation of **Brokerage Initiated** and **Customer Initiated Transaction** types via a card deck methodology designed to satisfy the specified mix (see CETxnMixGenerator.cpp). The **Market Triggered Transactions** are not generated by the **CE** but arise from asynchronous actions in the **MEE**.

Since deviations from the specified mix are still possible, it is the **Test Sponsor's** responsibility to make sure that the following criteria were indeed met for the **Measurement Interval** in order for the **Measurement Interval** to be valid. For the purposes of verifying that these criteria are met any and all **Valid Transactions** whose sT_n and eT_n are both within the **Measurement Interval** are to be counted.

5.3.1 Mix Requirements

The following table shows the target mix percentages for the two **Tier B Virtual Machines** of each **Group**. The **Test Sponsor** must show that the actual percentage obtained for each **Transaction** type over the entire **Measurement Interval** is within the specified Required Range.

VM in Group	Transaction	Target Pct	Required Range	Comment
VM2	Trade-Lookup	9%	8.955%-9.045%	
VM2	Trade-Update	1%	0.995%-1.005%	
VM3	Broker-Volume	3.9%	3.881%-3.920%	
VM3	Customer-Position	15%	14.910%-15.090%	
VM3	Market-Watch	17%	16.905%-17.095%	
VM3	Security-Detail	16%	15.905%-16.095%	
VM3	Trade-Order	10.1%	10.049% – 10.151%	~1% of Trade Orders rollback (see Clause 5.4.1, rollback is 1 out of each 101 Trade Orders.). 99% of 10.1% is the 10% for Trade Result.
VM3	Trade-Result	10%	9.950% - 10.050%	There is one Trade-Result per Trade-Order completed by the MEE, but ~1% of Trade-Order Transactions rollback at time of initial processing.
VM3	Trade-Status	18%	17.900%-18.100%	
Total		**100%**		

Comment 1: The number of completed Trade-Results is one per non-rolled-back Trade-Order. However, pending limit orders are delayed until their trigger price is reached. Therefore mix percentages may vary over short periods of time.

Comment 2: Only the first MEE instance issues Market-Feed Transactions, which shall be at the rate of 2 per second.

5.3.2 Required Precision for Mix Percentage Reporting

The **Transaction Mix** percentages must be **reported** to the same precision as shown in the Required Range in the table in Clause 5.3.1.

Computing the mix frequencies actually obtained during the **Measurement Interval** must be done with at least four decimal places and must be rounded to the nearest three decimal places when **reported**. For example, 7.2344 must be **reported** as 7.234 and 7.2345 must be **reported** as 7.235

5.3.3 Data-Maintenance

For each of the two **Tier B Virtual Machines** in each **Group**, a single Data-Maintenance **Transaction** must be invoked every sixty seconds. The Data-Maintenance transaction submitted to each **VM** conforms to the table cardinalities of the database in that **VM**. The actual interval between the executions of two consecutive **Transactions** must be no less than 58 seconds and no more than 62 seconds. Each Data-Maintenance **Transaction** must successfully complete in 55 seconds or less.

5.3.4 Trade-Cleanup

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 be invoked and finish before any other type of **Transaction** can be executed.

5.4 Transaction Parameters

Each **Transaction** type has variable inputs. Some of the **Transactions** have specified percentages (see DriverParamSettings.h) for the possible values of these inputs. During the **Test Run**, the **VGenDriver** code controls the generation of the values for theses inputs using a random number generator in a manner designed to satisfy the specified percentage (see CETxnInputGenerator.cpp). However since deviations from the specified percentage are still possible, it is the **Test Sponsor's** responsibility to make sure that the following criteria were indeed met for the **Measurement Interval** in order for the **Measurement Interval** to be valid. For the purposes of verifying that these criteria are met, inputs for any and all **Valid Transactions**, whose sT_n and eTn are both within the **Measurement Interval,** are to be counted.

5.4.1 Input Value Mix Requirements

The following table shows the target input value percentages. The **Test Sponsor** must show that the actual percentage obtained for each input type over the entire **Measurement Interval** is within the specified Required Range.

Input Parameter	Value	Target Pct	Required Range
Customer-Position			
by_tax_id	1	50%	48% to 52%
get_history	1	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	1	1%	0.9% to 1.1%
Trade-Lookup			
frame_to_execute	1	40%	38%-42%
	2	30%	28.5% to 31.5%
	3	20%	19%-21%
	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%

Input Parameter	Value	Target Pct	Required Range
type_is_margin	1	8%	7.5% to 8.5%
roll_it_back	1	~1%	0.94% to 1.04% (*)
is_lifo	1	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	45%	43%-47%
	2	33%	31% to 35%
	3	22%	20%-24%

(*) **Comment:** The ratio of rolled-back trades to completed trades is 1/100 or 1%, so the ratio of rolled-back 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 on the expected 0.99% value.

5.5 Response Time

5.5.1 Response Time

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

RTn = eTn - sTn

where:

sTn and eTn are measured at the **Driver**;

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

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

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

5.5.1.2 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
Broker-Volume	3 sec.
Customer-Position	3 sec.

Market-Feed	2 sec.
Market-Watch	3 sec.
Security-Detail	3 sec.
Trade-Lookup	3 sec.
Trade-Order	2 sec.
Trade-Result	2 sec.
Trade-Status	1 sec.
Trade-Update	3 sec.

5.5.1.3   The following diagram illustrates where **Response Time**'s are measured for each type of **Transaction**. Time stamps are taken on the **Driver**.
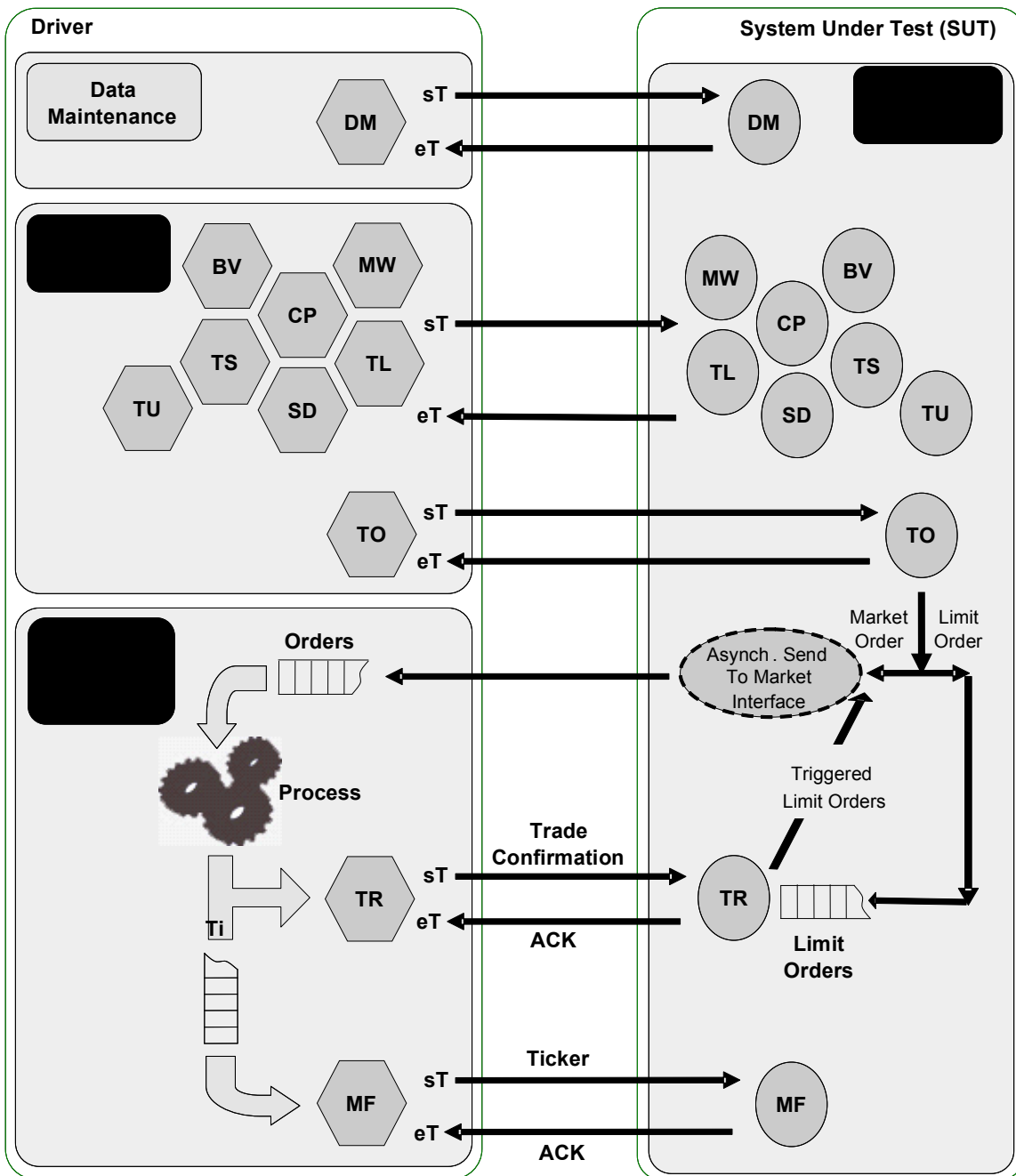
Figure 5.b - Measuring Response Time

5.5.1.4   Over the **Measurement Interval**, the average **Response Time** for each type of **Transaction** that is part of the **Transaction Mix** must not be longer than the 90th percentile **Response Time** for that **Transaction**.

5.5.1.5   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.

5.5.1.6   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.

5.6 Test Run

5.6.1 Definition of Terms

5.6.1.1 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**.

5.6.1.2 The term **Ramp-up** refers to the period of time from the start of the **Test Run** to the start of **Steady State**.

5.6.1.3 The term **Steady State** refers to the period of time from the end of the **Ramp-up** to the start of the **Ramp-down**.

5.6.1.4 The term **Ramp-down** refers to the period of time from the end of **Steady State** to the end of the **Test Run**.

5.6.1.5 The term **Measurement Interval** refers to the period of time during **Steady State** chosen by the **Test Sponsor** to compute the **Reported Throughput**.

5.6.1.6 The term **Business Day** refers to a period of eight hours of transaction processing activity.

5.6.1.7 The performance of the **SUT** is defined to be **Sustainable** if Performance over a given period of time (computed as the average throughput over that time) is considered **Sustainable** if it shows no significant variations.

5.6.2 Database Content

5.6.2.1 Prior to the first **Test Run**, the initial database for each **VM** must satisfy Clause 2.4.1. Prior to any **Test Run**, the database must satisfy Clause **10.4** and Clause 2.4.2.

Comment: Clause 2.4.2 defines cardinality changes as **Transactions** are executed against the database. If no **Transactions** have been executed, then initial cardinalities of Clause 2.4.1 apply.

5.6.2.2 At the start of a **Test Run** each 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.**

5.6.2.3 The only changes (unless otherwise directed by an **Auditor**) that can be made to the content of the TPCx-V database tables between the initial population and a valid **Test Run** must be performed by the running of **Valid Transactions,** as defined in this specification.

5.6.3 Sustainable Performance

5.6.3.1 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**.

5.6.3.2 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: This requirement is met when the aggregate throughput computed over any period of one hour, sliding over the **Steady State** by increments of twelve minutes, varies from the **Reported Throughput** by no more than 2%.

5.6.3.3   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: This requirement is met when the aggregate throughput level computed over any period of twelve minutes, sliding over the **Steady State** by increments of one minute, varies from the **Reported Throughput** by no more than 20%.

5.6.3.4   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**.

5.6.4   Steady State

5.6.4.1   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**. (For example see Clauses 5.6.5.2 and 5.3.3).

5.6.4.2   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,

5.6.5   Measurement Interval

5.6.5.1   The **Measurement Interval** must be an integer multiple of two hours and must occur entirely during **Steady State.** The start of the **Measurement Interval** has to coincide with the start of an **Elasticity Phase.** The **Measurement Interval** may start at the beginning of any of the ten **Elasticity Phases**.

Comment 1: The ten **Elasticity Phases** (see Clause 5.2) take two hours for one complete cycle, so the **Measurement Interval** must cover one or more full repetitions of these workload variations.

Comment 2: The Start of a **Measurement Interval** can be at the beginning of any arbitrary **Elasticity Phase** within the Dynamic Workload Variations that meets all of the other requirements. For example, the **Measurement Interval** may begin at the start of **Elasticity Phase** number 7 and end after 10 Phases at the conclusion of subsequent **Elasticity Phase** number 6.

Comment 3:It is required that the **Measurement Interval** contains exactly an integer multiple of 10 **Elasticity Phases** in the (cyclical) order defined in Clause 5.2. Determining that Start may be done during execution or after the end of the Test Run (e.g., when post-processing Driver log files).

5.6.5.2   During the **Measurement Interval**, the database contents (excluding the transaction log) stored on **Durable Media** cannot be more than 12 minutes older than any **Committed** state of the database.

Comment: This may mean that **Database Management Systems** implementing traditional checkpoint algorithms may need to perform checkpoints twice as frequently (i.e. every 6 minutes) in order to guarantee that the 12-minute requirement is met.

5.6.5.3   For the purposes of calculating **reported Transaction** statistics, all **Transactions** and only those **Transactions** whose sT_n and eT_n are within the **Measurement Interval** are used.

5.6.5.4   A transaction is considered to have taken place in an **Elasticity Phase** if its end time eT_n is within that Elasticity Phase, regardless of when the transaction started as long as both sT_n and eT_n are within the **Measurement Interval.**

5.6.5.5   All threads of execution that perform **CE Transactions** during the **Measurement Interval** must:

- Perform at least one **Transaction** prior to the start of the **Measurement Interval**
- Perform at least one **Transaction** after the end of the **Measurement Interval**

Creating, starting, stopping, or deleting threads of execution that perform **CE Transactions** is not permitted during the **Measurement Interval**.

5.6.6   Database Growth

5.6.6.1   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 (see Clause 5.6.3) 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**).

5.6.6.2   **Initial Database Size** is any space allocated to the test database that is used to store the initial population, **Database Metadata**, **User-Defined Objects**, and any space used as formatting overhead by the **DBMS**. **Initial Database Size** is measured after the database is initially loaded with the data generated by **VGenLoader**.

5.6.6.3   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, **Database Metadata**) or not already used as formatting overhead by the **DBMS**.
- **Growing Space**, which includes any space used to store initially-loaded rows from the **Growing Tables** and their associated **User-Defined Objects**. It also 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 that does not qualify as either **Free Space** or **Growing Space**.

  Comment: While cardinality does not change for non-**Growing Tables**, it is possible that some **Fixed Space** storage could increase for other reasons. If the computed increase for the **Business Day** for any such object would be greater than the 5% cardinality increase already imposed on non-**Growing Table** objects by Clause 10.3.9, then the larger computed storage increase must be used instead of the 5% increase.

5.6.6.4   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:
    - **Data Growth** = *Data-Space-per-Trade-Result* * tpsV * **Business Day** duration in seconds

5.6.6.5   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:
    - **Log Growth** = *Log-Space-per-Trade-Result* * tpsV * **Business Day** duration in seconds

5.6.7   Continuous Operation Requirement

Within the **Measured Configuration**, there must be sufficient **On-Line** storage to support:

- The **Initial Database Size**.
- An additional **Business Day's Data Growth** and **Log Growth** at the **reported tpsV**. The methods to calculate the **Data Growth** and the **Log Growth** are described in Clauses 6.6.6.4 and 5.6.6.5.

Comment: 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 6.5 are **On-Line** and meet the **Sustainable** performance requirements of Clause 5.6.3.

5.6.8   Performance & Database Size

5.6.8.1   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. It is bound by the limits defined in Clause 6.7.8.5.

5.6.8.2   The **Measured Throughput** must be measured, rather than interpolated or extrapolated.

5.6.8.3

5.6.8.4   To keep throughput proportional to database size, each **Measured Throughput** must be within a certain range of performance based on the database size.

5.6.8.5   **Nominal Throughput** is defined to be 2.00 **Transactions**-Per-Second-V for every 1000 customer rows in the **Configured Customers**.

5.6.8.6   Another way of expressing the **Nominal Throughput** is by using a **Scale Factor**, which is defined as: The **Scale Factor** is the number of required customer rows per single **Transactions-Per-Second-V.** The **Scale Factor** for **Nominal Throughput** is 500.

5.6.8.7   The number of **Load Units** configured per **Group** must be equal to the number of **Load Units** actually accessed per **Group** during the **Test Run.**

5.7   Required Reporting

5.7.1   Reported Throughput

5.7.1.1   The **Performance Metric reported** by TPCx-V is the Reported Throughput. The **name** of the metric used for the **TPCx-V Reported** Throughput of **the** SUT is tpsV. **The** value of this metric is based on the **Measured** Throughput and is bound by **the limits** defined in Clause 5.7.1.2.

5.7.1.2   The **Measured Throughput** must be between 80% and 102% of the **Nominal Throughput**. If **Measured Throughput** exceeds the **Nominal Throughput**, but not by more than 2%, the measurement may be used, but the **Reported Throughput** must be set to the **Nominal Throughput**. Otherwise, the **Reported Throughput** equals the **Measured Throughput**. If the **Measured Throughput** is not within these bounds, then the measurement is invalid and may not be reported.

5.7.1.3   The **Measured Throughput** of each **Group** should be individually calculated and reported. If there are N **Tiles**, as per Clause 4.3.4.1, the contribution of each **Group** to the aggregate **Measured Throughput** should be between 98% and 102% of (**Measured Throughput** * (*Group %*))/N, with *Group %* set to 10%, 20%, 30%, and 40% for **Group** 1, 2, 3, and 4, respectively.

5.7.1.4   The **Reported Throughput** must be rounded down to the nearest two decimal places. For example, suppose 105.748 tpsV is measured during a **Measurement Interval**. Then the **Reported Throughput** is 105.74 tpsV rather than 105.75 or some interpolated value between 105.748 and 117.572.

5.7.2   Test Run Graph

A graph of the one-minute average tpsV 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 one-minute average throughput in tpsV (computed as the total number of Trade-Result Transactions that complete within each one-minute interval divided by 60). A plot interval size of 1 minute must be used. The Ramp-up, **Steady State**, **Measurement Interval**, and **Ramp-down** must be identified on the graph. The **Test Run Graph** must be reported in the **Report**.
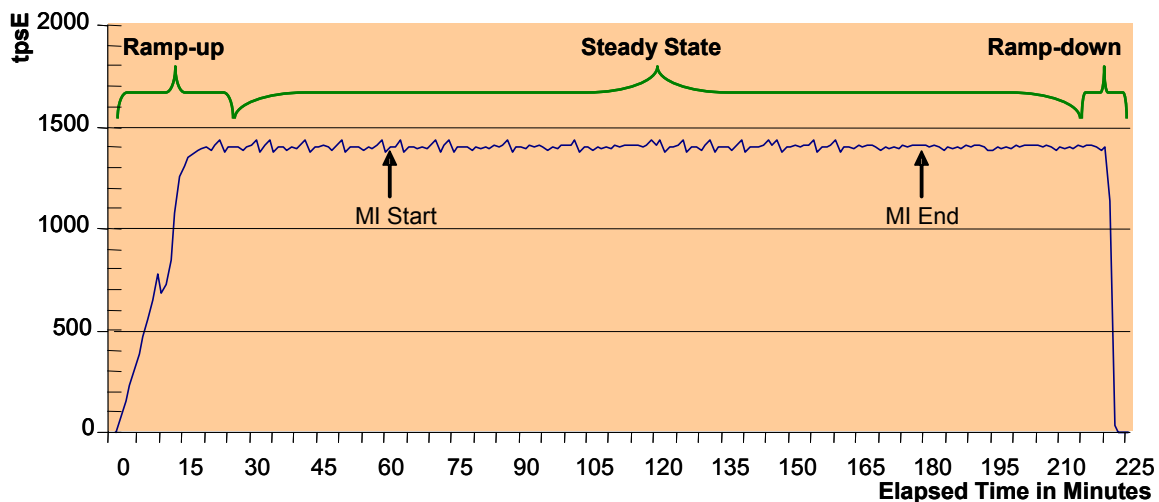
Figure 5c - Example of the Measured Throughput versus Elapsed Time Graph

5.7.3   Primary Metrics

5.7.3.1   To be compliant with the TPCx-V standard and the TPC's Fair Use Policies and Guidelines, all public references to TPCx-V **Results** for a configuration must include the following components which will be known as the Primary Metrics.

- The TPCx-V **Reported Throughput** as expressed in tpsV.
- The TPCx-V total 3-year pricing divided by the **Reported Throughput** is price/tpsV. This is also known as the **Price/Performance Metric** (See Clause 7 --).
- 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 8.2.1.1).

5.7.4   VGenValidate Results

5.7.4.1   Each **Load Unit** must do approximately the same number of Trade-Results during the **Measurement Interval**. This requirement must be demonstrated by passing the **VGenValidate** test and providing these results in the **Supporting Files**. **VGenValidate** is TPC provided code. A **Sponsor** must generate the **VGenValidate** binary executable for their environment.

5.7.4.2   When the **Sponsor** runs **VGenValidate** they must provide the following inputs:-

- number of **Configured Customers** from all **Groups**, summed up
- the standard deviation of the number of Trade-Results performed for each **Load Unit** of all VM3 databases (how?) during the **Measurement Interval** (see Clause 5.6.5.2)

$$\text{standard deviation} = \sqrt{\frac{\sum (\bar{n} - n_i)^2}{(\#LU - 1)}}$$

where n_i = the number of Trade-Results completed for Load Unit i during the Measurement Interval

The precision required for the standard deviation is three decimal digits to the right of the decimal point, rounded up if necessary.

- the number of seconds in the **Measurement Interval**
- the actual tpsV rate recorded during the **Measurement Interval**

5.7.4.3   **VGenValidate** does the following:-

1. Reads the input values.

2. Uses **VGen** code to simulate up to 10,000 runs on a TPCx-V 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**.

3. Prints out "Passed!" if the maximum calculated standard deviation across the 10,000 simulated runs is larger than or equal to the standard deviation for the **Measurement Interval**. Otherwise "Failed!" is printed.

CLAUSE 6 -- <u>TRANSACTION AND SYSTEM PROPERTIES (ACID)</u>

6.1   ACID Properties

6.1.1   The **ACID** (Atomicity, Consistency, Isolation, and Durability) properties of transaction processing systems must be supported by the **System Under Test** during the running of this benchmark.

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

6.1.3   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 **Test Run**. They are not intended to be an exhaustive quality assurance test.

6.1.4   The configuration needed to insure full **ACID** properties must be enabled during the **Test Run**. This applies to both the database (including TPCx-V tables and **User-Defined Objects**) and the **Database Session**(s) used to execute the **ACID** tests and the **Test Run**.

> **Comment 1:** The term "configuration" includes all database properties and characteristics that can be externally defined; this includes but is not limited to 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**.

6.1.5   Although the **ACID** tests do not exercise all **Transaction** types of this workload, the **ACID** properties must be satisfied for all **Transactions**.

6.1.6   Both databases in the **Tier B VMs** of each **Group** of each **Tile** must meet the **ACID** property requirements.

6.1.7   **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. All **FDRs** must identify the systems that were used to verify **ACID** requirements and full details of the **ACID** tests conducted and results obtained.

6.1.8   The TPCx-V Express **Benchmark Kit** performs the Atomicity, Consistency, and Isolation tests required by this Specification, and reports the results in the **Report**. The details of these tests are described in Clauses 6.2, 6.3, and 6.4. The Atomicity, Consistency, and Isolation tests are on all databases configured on the SUT. Only one VM is tested for Durability, as described in Clause 6.5.

6.2   Atomicity Requirements

6.2.1   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 ensure that no partially completed operations leave any effects on the data.

6.2.2 Atomicity Tests

Perform a market Trade-Order **Transaction** with the *roll_it_back* flag set to 0. 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 1. Verify that no rows associated with the rolled back Trade-Order have been added to the TRADE and TRADE_HISTORY tables.

6.3 Consistency Requirements

6.3.1 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.

6.3.1.1 A TPCx-V database when first populated by **VGenLoader** must meet these consistency conditions.

6.3.1.2 If data is replicated, as permitted under Clause 10.3.4, each copy must meet the consistency conditions defined in Clause 6.3.2.

6.3.2 Consistency Conditions

Three consistency conditions are defined in the following clauses. Explicit demonstration that the conditions are satisfied is required for all three conditions.

6.3.2.1 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').

6.3.2.2 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').

6.3.2.3 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).

6.3.3 Consistency Tests

The three consistency conditions must be tested after initial database population and after any **Business Recovery** tests.

6.4 Isolation Requirements

6.4.1 Isolation Property Definition

6.4.1.1 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.

6.4.1.2 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

6.4.1.3 During the **Test Run**, each TPCx-V **Transaction** must provide a level of isolation from **Arbitrary Transactions** that is at least as restrictive as the level defined in the table below:

TPCx-V Transaction	Isolation Level
Trade-Result	L3

Market-Feed Trade-Order Trade-Update	L2
Broker-Volume Customer-Position Data-Maintenance Market-Watch Security-Detail Trade-Lookup Trade-Status	L1

6.4.1.4   During the **Test Run** the **SUT** must allow concurrent execution of **Arbitrary Transactions**.

6.4.1.5   During the **Test Run**, the data read by each TPCx-V **Transaction** must be no older than the most recently **Committed** data at the time the **Transaction** started.

6.4.1.6   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 6.4.2.

6.4.1.7   Systems that implement **Transaction** isolation using techniques other 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.

6.4.2   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 6.4.1.3.

6.4.2.1   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**. The second Trade-Result **Transaction (Session** S4 below) plays the role of an **Arbitrary Transaction** that is inserting a row into a range of the HOLDING_SUMMARY table which has been accessed by the first Trade-Result **Transaction (Session** S3 below).

For the purpose of this test, the two Trade-Result **Transactions** must be instrumented to record *hs_qty* after returning from **Frame** 1. In addition, the Trade-Result **Transaction** executed by S3 must be able to repeat the execution of **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 an *acct_id*. Using an ad hoc read-only transaction, find a *symbol* that does not have a corresponding row in the HOLDING_SUMMARY table for the selected *acct_id* and perform a commit.

10. From S1, request and successfully complete a Trade-Order for the *acct_id* and *symbol* selected in step 1. Record the *trade_id* assigned to this new trade.

11. From S2, request and successfully complete another Trade-Order for the *acct_id* and *symbol* used in step 2. Record the *trade_id* assigned to this new trade.

12. From S3, request a Trade-Result for the *trade_id* from step 2. Pause between **Frame** 1 and **Frame** 2. Record *hs_qty* and verify that it is set to zero.

13. From S4, request a Trade-Result for the *trade_id* from step 3. Verify that it completes **Frame** 1 and starts execution of **Frame** 2. Record *hs_qty* and verify that it is set to zero.

   Case A, if S4 stalls in **Frame** 2, then rolls back, while S3 completes:

   6A. From S3, repeat the execution of **Frame** 1 and pause again between **Frame** 1 and **Frame** 2. Record *hs_qty* and verify that it set to zero.

   7A. Resume execution of S3 at **Frame** 2. Verify the successful completion of the remaining **Frames**.

   8A. Verify that S4 rolled back.

   Case B, if S4 completes (perhaps after stall) and S3 rolls back:

   6B. Verify that S4 completes the execution of **Frame** 2 and of the remaining **Frames**.

   7B. Verify that S3 rolled back.

   Case C, if S4 stalls in **Frame** 1 (Invalid):

   6C. If this case occurs, the test is invalid. To properly test protection against phantom reads, **Session** S4 must get to the point in Trade-Result **Frame** 2 where a row is inserted into HOLDING_SUMMARY. The Trade-Result **Transaction** used for S4 may need to be modified to prevent it from blocking in **Frame** 1. For example, it may be executed at the lower isolation level of an **Arbitrary Transaction**.

Comment 1: This P3 test is successful if either Case A or B is followed. It fails if Case C occurs. Other valid possibilities may exist (e.g., both S3 and S4 could fail), but if both S3 and S4 record *hs_qty* = 0 from execution of **Frame** 1, then at most one of these **Sessions** may complete normally and commit the **Transaction**. The intent of this test is to demonstrate that in all circumstances when S3 repeats the read on the HOLDING_SUMMARY table after S4 has inserted (or attempted to insert) a new row for the selected *acct_id* and *symbol*, there is still no qualifying row found by S3.

Comment 2: This isolation test creates one or more new holdings. Subsequently executing the P2 Test in Read-Write (see Clause 6.4.2.2) for the same selected *acct_id* and *symbol* can result in closing the positions created during the execution of this test.

6.4.2.2 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**. The second Trade-Result **Transaction** (**Session** S4 below) plays the role of an **Arbitrary Transaction** that is updating a row in the HOLDING_SUMMARY table which has been read by the first Trade-Result **Transaction** (**Session** S3 below).

For the purpose of this test, the two Trade-Result **Transactions** must be instrumented to record *hs_qty* after returning from **Frame** 1. In addition, the Trade-Result **Transaction** executed by S3 must be able to repeat the execution of **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 an *acct_id*. Using an ad hoc read-only transaction, find a *symbol* that has a corresponding row in the HOLDING_SUMMARY table for the selected *acct_id*, record the HS_QTY for that holding and perform a commit.

14. From S1, request and successfully complete a Trade-Order for the *acct_id* and *symbol* selected in step 1. Record the *trade_id* assigned to this new trade.

15. From S2, request and successfully complete another Trade-Order for the *acct_id* and *symbol* used in step 2. Record the *trade_id* assigned to this new trade.

16. From S3, request a Trade-Result for the *trade_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.

17. From S4, request a Trade-Result for the *trade_id* from step 3. Verify that it completes **Frame** 1 and starts execution of **Frame** 2. Record *hs_qty* and verify that it is equal to HS_QTY from step 1.

   Case A, if S4 stalls in **Frame** 2, then rolls back, while S3 completes:

6A. 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 equal to HS_QTY from step 1.

7A. Resume execution of S3 by invoking **Frame** 2. Verify the successful completion of the remaining **Frames**.

8A. Verify that S4 rolled back.

Case B, if S4 completes (perhaps after stall) and S3 rolls back:

6B. Verify that S4 completes the execution of **Frame** 2 and the remaining **Frames**.

7B. Verify that S3 rolled back.

Case C, if S4 stalls in **Frame** 1 (Invalid):

6C. If this case occurs, the test is invalid. To properly test protection against the Non-repeatable Read phenomenon P2, **Session** S4 must get to the point in Trade-Result **Frame** 2 where a row is updated in HOLDING_SUMMARY. The Trade-Result **Transaction** used for S4 may need to be modified to prevent it blocking in **Frame** 1. For example, it may be executed at the lower isolation level of an **Arbitrary Transaction**.

Comment: This test is successful if either Case A or B is followed. It fails if Case C occurs. Other valid possibilities may exist (e.g., both S3 and S4 could fail), but if both S3 and S4 record the same *hs_qty* value from execution of **Frame** 1, then at most one of these **Sessions** may complete normally and commit the **Transaction**. The intent of this test is to demonstrate that in all circumstances when S3 repeats the read on the HOLDING_SUMMARY table for the selected *acct_id* and *symbol*, the row found and value is the same as in Step 1.

6.4.2.3 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 *acct_id[]* and *cash_ball[]*.

2. From S1, request and successfully complete a Trade-Order from an *acct_id* selected from the set recorded in step 1, for a given *symbol* and with a *type_is_margin* set to 0. Record the *trade_id* assigned to this new trade.

3. From S1, request and successfully complete another Trade-Order for the same *acct_id* but a different *symbol* than that used in step 2, and with a *type_is_margin* set to 0. Record the *trade_id* assigned to this new trade.

18. From S2, request a Trade-Result for the *trade_id* from step 2. Before invoking **Frame** 6, record *se_amount*, then invoke **Frame** 6 and pause before committing.

19. From S3, request a Trade-Result for the *trade_id* from step 3. The **Transaction** may pause or fail or be temporarily blocked from fully executing. If it reaches the start of **Frame** 6, record *se_amount*, then invoke **Frame** 6. If it reaches the end of **Frame** 6, pause before committing.

20. From S2, proceed with committing and successfully completing the **Transaction**. Record the resulting *acct_bal*.

21. From S3, depending on how the **Transaction** behaved at the end of step 5:

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 *trade_id* input parameter. Verify that the Trade-Result executes in full, records *se_amount* at the start of **Frame** 6, commits at the end of **Frame** 6 and completes successfully.

22. From S3, record the resulting *acct_bal* and verify that it is equal to *cash_bal[]* from step 1 (for the *acct_id* chosen in step 2) plus the sum of the *se_amount* outputs for the two Trade-Results.

6.4.2.4   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 *acct_id[]* and *cash_bal[]*.

2. From S1, request and successfully complete a Trade-Order where the associated *acct_id* input matches one of the *acct_id[]* recorded in step 1 and *type_is_margin* is 0. Record the *trade_id* assigned to this new trade.

23. From S2, request a Trade-Result for the *trade_id* from step 2 and then pause in **Frame** 6 before committing.

24. From S3, request a Customer-Position for the *cust_id* selected in step 1. The **Transaction** may complete or fail or be temporarily blocked from fully executing.

25. From S2, proceed with committing and successfully completing the Trade-Result **Transaction**. Record the resulting *acct_bal*.

26. From S3, depending on how the Customer-Position **Transaction** behaved at the end of step 4:

    If it completed, record the set of resulting *acct_id[]* and *cash_bal[]* and verify that the *cash_bal* for the *acct_id* used in step 2 is unchanged from step 1.

    If it was blocked, verify that it has now completed, record the set of resulting *acct_id[]* and *cash_bal[]* and verify that the *cash_bal* for the *acct_id* used in step 2 matches the *acct_bal* from step 5.

    If it failed, proceed to the next step.

27. From S4, request a Customer-Position for the *cust_id* selected in step 1, complete the **Transaction**, record the set of resulting *acct_id[]* and *cash_bal[]* and verify that the *cash_bal* for the *acct_id* used in step 2 has changed from step 1 and reflects the amount of the trade completed in step 5 (by matching *acct_bal* from step 5).

6.5   Durability Requirements

No system provides complete data protection under all possible types and/or combinations of failures. However, data protection against any Single Point of Failure is commonly expected. Therefore, the intent of this clause is to ensure that the **SUT** has no unrecoverable **Single Points of Failure**. The required data protection is satisfied by the **SUT** persisting certain data across certain types of failures.

This clause provides details on:

- Which data must persist
- Which types of failures must be protected against
- Which steps to follow for the testing/demonstration
- Which results must be disclosed

Comment: The limited nature of the tests described in this clause must not be interpreted to allow other unrecoverable Single Points of Failure.

6.5.1 Definition of Commit

The concept of "commit" has to do with delineating the successful completion of an atomic unit of work. The following definition will be leveraged to focus the scope of which data must be persisted by the **SUT**.

Commit is a control operation that:

- Is initiated by a unit of work (a **Transaction**)
- Is implemented by the **DBMS**
- Signifies that the unit of work has completed successfully and all tentatively modified data are to persist (until modified by some other operation or unit of work)

Upon successful completion of this control operation both the **Transaction** and the data are said to be **Committed**.

-

6.5.2 Definition of Single Point(s) of Failure

This clause lists various types of failures that can occur within the **SUT**. This list will be leveraged to focus the scope of failures the **SUT** must protect against.

Any single item covered here is defined to be a **Single Point of Failure**; when two or more items are being discussed, the term **Single Points of Failure** is used.

At present only one type of **Single Point of Failure** is defined in Clause 6.5.2.1.

6.5.2.1 Loss of Processing

This failure covers an instantaneous interruption in processing **Commit** control operations to a **Virtual Machine** in a **Group** (e.g. system crash / system hang) that requires the **Virtual Machine** to be started from the file system image of the **Virtual Machine**. This implies an immediate abnormal system shutdown where the run-time state and the memory contents of the **VM** are lost, but the virtual disk contents are intact although possibly in an unknown state. A recovery requires starting the **Virtual Machine**, rebooting the **VM** operating system, recovering the file systems in the **VM**, and, if necessary, restoring the **DBMS** from a backup and roll forward from the **Undo/Redo Log**.

6.5.3 Definition of Durable / Durability

The **SUT** must provide **Durability** as defined in this clause.

In general, state that persists across failures is said to be **Durable** and an implementation that ensures state persists across failures is said to provide **Durability**. In the context of the benchmark, **Durability** is more tightly defined as the **SUT**'s ability to ensure all **Committed** data persist across any **Single Point of Failure**.

6.5.4 Durability Testing Rules and Guidelines

The intent of this clause is to cover specific rules and special-case guidelines.

6.5.4.1 Durability Throughput Requirements

All **Durability** tests must meet the following requirements:

- Be performed with the same number of **Configured Customers** and **Driver** load used for the **Measurement Interval.**
- Be in **Steady State.**
- Satisfy the **Response Time** constraints in Clause 5.5.1.2.
- Satisfy the **Transaction Mix** requirements listed in Clause 5.3.1.

- Be at or above 95% of the **Reported Throughput** with no errors.
- Match all **Driver** and **SUT** configuration settings used during the **Measurement Interval**.

6.5.4.2   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 6.5.2.1. Note that "checkpoints", "control points", "consistency points", etc. of the database taken during a run are not considered to be archives.

6.5.4.3   Instantaneous Failures

Single Points of Failure must be induced instantaneously without any foreknowledge given to the **SUT**.

Comment: Reactive actions initiated within the SUT as a result of an Instantaneous Failure are not considered foreknowledge.

6.5.4.4   Simulated Failures

A **Single Point of Failure** may be simulated if the effects on the SUT are identical to those of the actual occurrence of the **Single Point of Failure**. In particular, the loss of processing (e.g., Clause 6.5.2.1) may be simulated using a **VMMS** command that instantaneously shuts down the **VM**.

6.5.4.5   Multiple Identical Single Points of Failure

If the **SUT** contains multiple identical **Single Points of Failure** as defined in Clause 6.5.2 that perform identical benchmark functions, successful demonstration of **Durability** for one instance is sufficient; there is no requirement to repeat the demonstration for all the other instances unless directed to do so by the **Auditor**.

Example – Loss of Processing: In configurations where more than one instance of an **Operating System** performs an identical benchmark function, **Durability** for the failure in Clause 6.5.2.1 must be completed on at least one such instance.

6.5.5   Definition of Recovery Terms

6.5.5.1   Database Recovery

Database Recovery is the process of recovering the database from a **Single Point of Failure** system failure.

6.5.5.2   Database Recovery – Start Time

The start of **Database 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: 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 **Database Recovery**.

6.5.5.3   Database Recovery – End Time

The end of **Database Recovery** is the time at which database files have been recovered.

Comment: The database will usually report this time in its log files.

6.5.5.4   Database Recovery Time

Database Recovery Time is the duration from the start of **Database Recovery** to the point when database files complete recovery.

6.5.5.5   Application Recovery

Application Recovery **is the process of recovering the business application after a** Single Point of **Failure and reaching a point where the business meets certain operational criteria**.

6.5.5.6   Application Recovery – Start Time

The start of **Application Recovery** is the time when the first **Transaction** is submitted after the start of **Database Recovery**.

6.5.5.7   Application Recovery – End Time

The end of **Application Recovery** is the first time, T, after the start of **Application Recovery** at which the following conditions are met:

- The one-minute average tpsV (i.e. average tpsV over the interval from T to T + 1 minute) is greater than or equal to 95% of **Reported Throughput**
- The 20-minute average tpsV (i.e. average tpsV over the interval from T to T + 20 minutes) is greater than or equal to 95% of **Reported Throughput**.

Comment: When considering the 20-minute interval, the average tpsV for the first minute must be at or above 95% of **Reported Throughput** (as required by the first bullet above). However, some number of the subsequent 19 one-minute average tpsV values may drop below 95% of **Reported Throughput**. This is acceptable as long as the overall 20-minute average tpsV is not less than 95% of **Reported Throughput** (as required by the second bullet above).

6.5.5.8   Application Recovery Time

Application Recovery Time is the elapsed time between the start of **Application Recovery** and the end of **Application Recovery** (see Clause 6.5.5.5).

6.5.5.9   Business Recovery

Business Recovery is the process of recovering from a **Single Point of Failure** and reaching a point where the business meets certain operational criteria.

6.5.5.10   Business Recovery Time

Business Recovery Time is the elapsed period of time between start of **Business Recovery** and end of **Business Recovery** (see Clause 6.5.5.9).

Comment: **Single Points of Failure** can be very disruptive to business processing, therefore it is imperative for businesses to recover from these failures as quickly as possible. There are many database configuration parameters and practices that directly affect the performance of the **DBMS** and its recovery time from a **Single Point of Failure**. However, while it is recognized that boot times for systems vary greatly, boot parameters have little to no affect on the performance of the **DBMS**. For this reason, server boot times are not included as part of the **Business Recovery Time**.

6.5.6   Durability Test Procedure for Single Points of Failures

1. Determine the current number of completed trades in the database by running:

   select count(*) as count1 from SETTLEMENT.

2. Start **Test Run** 1 by submitting **Transactions** and ramp up to the **Durability Throughput Requirements** (as defined in Clause 6.5.4.1) and satisfy those requirements for at least 20 minutes.

3. Induce the **Single Points of Failure** failure, from Clause 6.5.2 to a **VM3 Virtual Machine**

4. If appropriate for the test configuration, stop submitting **Transactions**.

5. If necessary, restart the **SUT** (may necessitate a full reboot).

6. Note the time when **Database Recovery** starts (see Clause 6.5.5.2), either automatically or manually by an operator.

7. When **Database Recovery** ends, note the time. This may occur during the following steps (see Clause 6.5.5.3).

8. Start **Test Run** 2 or continue **Test Run** 1 submitting **Transactions** and note this time as the start of **Application Recovery** (see Clause 6.5.5.6). Ramp up to 95% of **Reported Throughput**.

    Comment: If there is a time gap between the end of **Database Recovery** and the start of **Application Recovery** and if **Drivers** and **Transactions** need to be started again (not just continued), then the Trade-Cleanup **Transaction** may be executed during this time gap.

9. Note the end of **Application Recovery** as defined in Clause 6.5.5.7.

10. Terminate the **Driver** gracefully.

11. Verify that no errors were reported by the **Driver** during steps 7 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:

    select count(*) as count2 from SETTLEMENT

13. Compare the number of completed 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 maximum number of **Transactions** which can be simultaneously in-flight from the **Driver** to the **SUT**. This number is specific to the implementation of the **Driver** and configuration settings at the time of the crash.

    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.

14. Verify consistency conditions as specified in Clause 6.3.3.

15. Calculate **Business Recovery Time** as the sum of **Application Recovery Time** and **Database Recovery Time**, if those times do not overlap. If **Application Recovery** begins before **Database Recovery** is complete, **Business Recovery Time** is the time elapsed between the beginning of **Database Recovery** and the end of **Application Recovery**.

6.5.7   Required Reporting for Durability

6.5.7.1   Business Recovery Time

The **Business Recovery Time** must be **reported** on the **Executive Summary Statement** and in the **Report**. All the **Business Recovery Times** for each test requiring **Business Recovery** must be **reported** in the **Report**.

6.5.7.2   Business Recovery Time Graph

A graph of the one-minute average tpsV 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 6.5.6 steps 2 and 8

- The y-axis represents the throughput in tpsV (computed as the total number of Trade-Result **Transactions** that complete within each one-minute interval divided by 60)
- 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 6.5.6 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 6.5.6 step 8, time 0 is defined as the point in time where **Database Recovery** begins
- For graphing purposes, the end of the run is defined as follows:
- For the run outlined in 6.5.6 step 2, the end of the run is the time at which the failure is induced (see 6.5.6 step 3)
- For the run outlined in 6.5.6 step 8, the end of the run is the time at which the **Application Recovery** has ended successfully (see 6.5.6 step 8)
- For the run outlined in 6.5.6 step 8, if any time elapses between the end of **Database Recovery** and the start of **Application Recovery**, this time should be ignored and the two periods should be presented adjacent on the graph.
- A horizontal line at 95% of the **Reported Throughput** must also be plotted across the graph

6.6   Data Accessibility Requirements

The **System Under Test** must be configured to satisfy the requirements for **Data Accessibility** detailed in this clause. **Date Accessibility** is the ability to maintain database operations with full data access after the permanent irrecoverable failure of any single **Durable Medium** containing database tables, recovery log data, or **Database Metadata**. **Data Accessibility** tests are conducted by inducing failures of **Durable Media** within the **SUT**. The failures of Clause 6.6.3 test the ability of the **SUT** to maintain access to the data. The specific set of single failures addressed in Clause 6.6.3 is defined sufficiently significant to justify demonstration of **Data Accessibility** across such failures. However, the limited nature of the tests listed must not be interpreted to allow other unrecoverable single points of failure.

6.6.1   Definition of Terms

6.6.1.1   **Date Accessibility** is the ability to maintain database operations with full data access after the permanent irrecoverable failure of any single **Durable Medium** containing database tables, recovery log data, or **Database Metadata**.

6.6.1.2   **Durable Medium is** a data storage medium that is inherently non-volatile such as a magnetic disk or tape. **Durable Media** is the plural of **Durable Medium**.

6.6.2   Data Accessibility Throughput Requirements

All **Data Accessibility** tests must meet the following requirements:

- Be performed with the same number of **Configured Customers** and **Driver** load used for the **Measurement Interval**
- Be in **Steady State**
- Satisfy the **Response Time** constraints in Clause 5.5.1.2.
- Satisfy the **Transaction Mix** requirements listed in Clause 5.3.1.
- Be at or above 95% of the **Reported Throughput** with no errors

- Match all **Driver** and **SUT** configuration settings used during the **Measurement Interval**

6.6.3 Failure of Durable Media

The failures detailed in this clause affect the access of data from **Durable Media**. The following requirements are also known as the **Data Accessibility** requirements.

6.6.3.1 The **SUT** must maintain database access to data on **Durable Media** during and after a permanent and irrecoverable failure of a 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**.

6.6.3.2 **Durable Media** are inherently non-volatile and are typically magnetic disks using replication (RAID-1 mirroring) or other form of protection (RAID-5, et.al.) to guarantee access to the data during a **Durable Medium** failure. Volatile media such as memory can also be used if the volatile media can 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: Memory can be considered a **Durable Medium** if it can preserve data long enough to satisfy the requirements stated 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). If main memory is used as a **Durable Medium**, then it must be considered as a potential single point of failure. A sample mechanism to survive single **Durable Medium** failure is 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.

6.6.3.3 The **Data Accessibility** tests (aka. **Non-catastrophic** failures) must meet the **Data Accessibility Throughput Requirements** of Clause 6.6.2.

6.6.3.4 **Redundancy Levels**

The redundancy levels refer to the level of guarantee for data access given a single failure among the data storage components. The **SUT** must implement one of the following Redundancy Levels:

- **Redundancy Level One (Durable Media Redundancy)** guarantees access to the data on **Durable Media** when a single **Durable Media** failure occurs.

  Comment: The intent of this redundancy level is to test the ability of the **Durable Media** environment to survive the failure of a single **Durable Medium** and continue processing requests from the **Operating System** and/or **DBMS**.

  Example: The **Sponsor** has implemented RAID-1 (mirroring) on the disks within an enclosure. The **Sponsor** must maintain access to the data on the remaining disks despite the induced failure of a single disk.

- **Redundancy Level Two (Durable Media Controller Redundancy)** includes **Redundancy Level One** and guarantees access to the data on **Durable Media** when a single failure occurs in the storage controller used to satisfy the redundancy level or in the communication media between the storage controller and the **Durable Media**.

  Comment: The intent of this redundancy level is to test the ability of the implementation to survive the failure of a storage controller responsible for implementing **Redundancy Level One**.

Example: If **Redundancy Level One** is satisfied by implementing RAID-5 protection within a disk enclosure, then **Redundancy Level Two** would be tested by failing the hardware used to implement the RAID-5 protection.

If the controller implementing the RAID-5 is contained within the disk enclosure (or similar externally attached device), then the **Sponsor** must demonstrate they can still access the data stored within the enclosure.

If the controller implementing the RAID-5 is separate from the enclosure containing the disks, and the controller is not being used as a **Durable Medium** (e.g. mirrored write caches), then it is sufficient to fail the communications between the controller and the enclosure.

- **Redundancy Level Three (Full Redundancy)** includes **Redundancy Level Two** and guarantees access to the data on **Durable Media** when a single failure occurs within the **Durable Media** system, including communications between **Tier B** and the **Durable Media** system.

  Comment 1: The **Durable Media** system includes all components necessary to meet the durability requirements defined above. This does not include the **Tier B** system or the system bus, but does include the adapter on the system bus and any and all components "downstream" from the adapter.

  Comment 2: The intent of this clause is to test the ability of the **Tier B** system to withstand component failures and continue processing of the **Transactions**.

Comment: The components being tested by this clause are those that are considered to be Field Replaceable Units (FRUs). It is not the intent of the clause to require **Sponsors** to test the durability of a backplane inside a **Durable Media** enclosure or similar non-replaceable components. However, testing the failover properties of storage controllers, including mirrored caches on a controller, and the corresponding software, is within the intent of this clause.

6.6.3.5 Test Procedure for Data Accessibility

1. Determine the current number of completed trades in the database by running:
   select count(*) as count1 from SETTLEMENT

2. Start submitting **Transactions** and ramp up to the **Data Accessibility Throughput Requirements** (as defined in Clause 6.6.2) and satisfy those requirements for at least 5 minutes.
   Comment: Once the **Data Accessibility Throughput Requirements** are met

   - no **Driver** configuration changes are permitted until the conclusion of step 5

   - no **SUT** configuration changes are permitted except those needed to satisfy steps 3 and 4

3. Induce the failure described for the redundancy level being demonstrated.

4. Begin the necessary recovery process.

5. Continue running the **Driver** for 20 minutes.

6. Terminate the run gracefully from the **Driver**.

7. Retrieve the new number of completed trades in the database by running:
   select count(*) as count2 from SETTLEMENT

8. 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.

9. Allow recovery process to complete as needed.

6.6.3.6 Requirement for Combinations of Durable Media Technologies

At least one of each combination of durable media technology, bus type, and redundancy level, (e.g. SSD/RAID-10, SATA/RAID-5, FC/RAID-5) must be tested independently as specified in clause 6.6.3.5.

6.6.4   Required Reporting for Data Accessibility

6.6.4.1   Redundancy Level

The **Test Sponsor** must report the Redundancy Level and describe the test(s) used to demonstrate compliance in the **Report**. A list of all combinations of **Durable Media** technologies tested in Clause 6.6.3.5 must be reported in the **Report**

6.6.4.2   Data Accessibility Time Graph

A graph of the Trade-Results per second averaged over one-minute 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 6.6.3.5, steps 2 through 6

- The y-axis represents the throughput in tpsV **(**computed as the total number of Trade-Result **Transactions** that complete within each one-minute interval divided by 60**)**

- A plot interval size of 1 minute must be used

- A horizontal line at 95% of the **Reported Throughput** must also be plotted across the graph

Comment: The intent is to show how throughput is affected during recovery.

CLAUSE 7 -- <u>PRICING</u>

Rules for pricing the **Priced Configuration** and associated software and maintenance are included in the TPC Pricing Specification, located at *www.tpc.org*.

The following requirements are intended to supplement the TPC Pricing Specification:

7.1   Priced Configuration

The system to be priced is the aggregation of the **SUT** 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 as defined in Clause 10.1.2.
- Price of any additional storage and associated infrastructure required by the **On-Line** Storage Requirement in Clause 7.2.
- 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 function defined in the TPCx-V specification it must be priced regardless of where is draws its resources.

7.2   On-line Storage Requirement

7.2.1   A storage device is considered **On-Line** if it is capable of providing an access time to data, for random read or update, of one second or less by the **Operating System.**

Comment: Examples of **On-Line** storage may include magnetic disks, optical disks, solid-state storage, or any combination of these, provided that the above mentioned access criteria is met.

7.2.2   **On-Line** storage must be priced for sufficient space to store and maintain the data and **User-Defined Objects** generated during a period of one **Business Day** at the **Reported Throughput**.

7.2.3   The first **Business Day** of storage must satisfy the requirements of the Clause 6.6.7 Continuous Operations Requirement.

7.2.4   **Archive Operation Requirement**

TPCx-V has no requirements for pricing additional archive storage.

7.2.5   **Back-up Storage Requirements**

TPCx-V has no requirements for on-line back-up data capabilities in the **Priced Configuration**.

7.3   TPCx-V Specific Pricing Requirements

7.3.1   **Additional Operational Components**

7.3.1.1   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.

7.3.1.2   Copies of the software, on appropriate media, and a software load device, if required for initial load or maintenance updates, must be included.

7.3.1.3   Clause 6.6.3.2The price of all components, including cables, used to interconnect components of the **SUT** must be included.

7.3.2   Additional Software

7.3.2.1   All software licenses must be priced for a number of users at least equal to one user for each tpsV of **Nominal Throughput**. Any usage pricing for this number of users must be based on the pricing policy of the company supplying the priced component.

7.3.2.2   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.

7.3.2.3   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.

7.4   Component Substitution

7.4.1   **Substitution** is defined as a deliberate act to replace components of the **Priced Configuration** by the **Test Sponsor** as a result of failing the availability requirements of the TPC Pricing Specification or when the **Part Number** for a component changes.

Comment: Corrections or "fixes" to components of the **Priced Configuration** are often required during the life of products. These changes are not considered **Substitutions** so long as the **Part Number** of the priced component does not change. Suppliers of hardware and software may update the components of the **Priced Configuration**, but these updates must not impact the **Reported Throughput**. The following are not considered **Substitutions**:

- software patches to resolve a security vulnerability
- silicon revision to correct errors
- new supplier of functionally equivalent components (i.e. memory chips, disk drives, ...)

7.4.2   Some hardware components of the **Priced Configuration** may be substituted after the **Test Sponsor** has demonstrated to the **Auditor's** satisfaction that the substituting components do not negatively impact the **Reported Throughput**. All **Substitutions** must be **reported** in the **Report** and noted in the **Auditor's Attestation Letter**. The following hardware components may be substituted:

- Durable Medium
- Durable Medium Enclosure
- Network interface card
- Router
- Bridge
- Repeater

7.5   Required Reporting

7.5.1   Two metrics will be **reported** with regard to pricing. The first is the total 3-year pricing as described in the effective version of the TPC Pricing specification. The second is the total 3-year pricing divided by the **Reported Throughput** (tpsV), as defined in Clause 5.7.1.

7.5.2   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 tpsV, then the 3-year pricing is $ 5,734,418 USD and the price/performance is $ 54,700 USD per tpsV (5,734,418/105).

CLAUSE 8 -- <u>FULL DISCLOSURE REPORT</u>

8.1 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,
- 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.

8.1.1 General Items

8.1.1.1 The order and titles of sections in the **Report** and **Supporting Files** must correspond with the order and titles of sections from the TPCx-V 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**.

8.1.1.2 The **FDR** must follow all reporting rules specified in the effective version of the TPC Pricing Specification, located at *www.tpc.org*. For clarity and readability the TPC Pricing Specification requirements may be repeated in the TPCx-V Specification.

8.1.1.3 The directory structure of the **FDR** has three folders:

- *ExecutiveSummaryStatement* - contains the **Executive Summary Statement**
- *Report* - contains the **Report**,
- *SupportingFiles* - contains the **Supporting Files**.

8.1.1.4 The reporting requirements of Clause 8 require descriptions, scripts and step-by-step GUI instructions that are necessary to reproduce the benchmark **Result**. The **Test Sponsor** can only provide descriptions, scripts and GUI instructions for the measured **SUT** as no knowledge is available at the time of publication of future changes in hardware or software. To meet the Clause 8.1 reproducibility requirement, the **Test Sponsor** must provide upon request any and all updated descriptions, scripts and step-by-step GUI instructions required to reproduce the benchmark **Result**.

8.2 Executive Summary Statement

The TPC Executive Summary Statement must be included near the beginning of the Report. An example of the Executive Summary Statement is presented in Appendix A. The Executive Summary Statement generated by the **Benchmark Kit** must be used.

8.2.1 First Page of the Executive Summary Statement

8.2.1.1 The first page of the **Executive Summary Statement** must include the following:

- **Sponsor's** name
- Measured server's name
- TPCx-V 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** in tpsV (see Clause 5.7.1)
- **Price/Performance Metric** (see TPC Pricing Specification)
- **Availability Date** (see TPC Pricing Specification)
- Total System Cost (see TPC Pricing Specification)
- **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 (see TPC Policies located at *www.tpc.org*)
- Memory in GB configured on the **Database Server**
- A diagram (see Clause 8.3.1.2) describing the components of the **Priced Configuration** (see TPC Pricing Specification)
- **Initial Database Size** in GB
- Redundancy Level and Redundancy Level implementation details
- Priced number of **Durable Media** (disks) for the database

8.2.2   Additional Pages of Executive Summary Statement

8.2.2.1   The Price Spreadsheet must be included in the **Executive Summary Statement** as specified by the TPC Pricing Specification.

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)

8.2.2.2   The name of the **Auditor** who certified the result must be included after the Price Spreadsheet.

8.2.2.3   The numerical quantities listed below must be included in the **Executive Summary Statement** after the Price Spreadsheet:

- **Reported Throughput** in tpsV (see Clause 5.7.1)
- **Configured Customers** (see Clause 2.4)
- **Measurement Interval** in hh:mm:ss (hours, minutes, seconds) (see Clause 5.6.1.5),
- **Ramp-up** time in hh:mm:ss (see Clause 5.6.1.2),
- **Business Recovery Time** in hh:mm:ss (see Clause6.5.7.1),
- The number of **Transactions** in the **Transaction Mix** completed within the **Measurement Interval**, (report the total, and the number per **Transaction** type) (see Clause 5.3.1)
- The number of each **Transaction** type (including Data-Maintenance) completed within the **Measurement Interval**
- Percentage of **Transaction Mix** for each **Transaction** type completed within the **Measurement Interval** (see Clause 5.3.1).

- Ninetieth percentile, minimum, maximum and average **Response Times** must be **reported** for all **Transactions** of the **Transaction Mix** completed within the **Measurement Interval** (see Clause 5.5.1).
- Maximum, minimum and average **Response Times** must be **reported** for Data-Maintenance.

8.3 Report Disclosure Requirements

8.3.1 Report Introduction

8.3.1.1   A statement identifying the benchmark **Sponsor**(s) and other participating companies must be **reported** in the **Report**.

8.3.1.2   Diagrams of both **Measured** and **Priced Configuration**s 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. **VMMS , 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.

8.3.1.3   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 TPCx-V benchmark measurement.
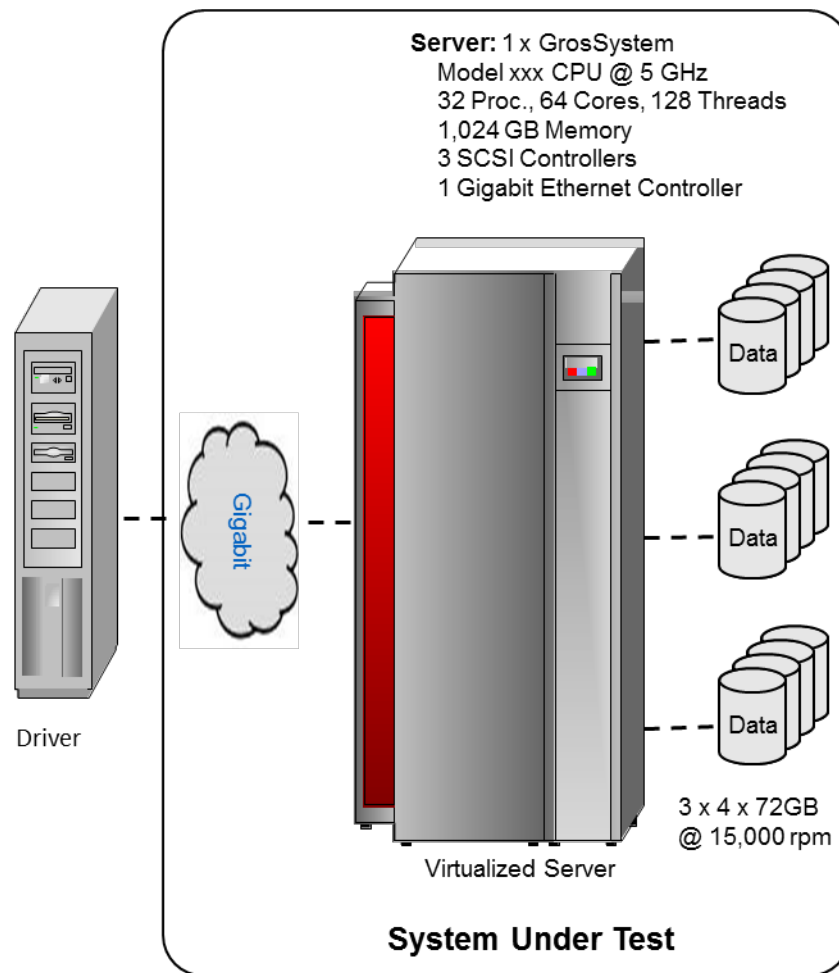
Figure 8a - Example of Measured Benchmark Configuration

8.3.1.4   A description of the steps taken to configure all of the hardware must be **reported** in the **Report**. Any and all configuration scripts or step-by-step GUI instructions are **reported** in the **Supporting Files** (see Clause 8.4.1.1). The description, scripts and GUI instructions must be sufficient such that a reader knowledgeable of computer systems and the TPCx-V specification could recreate the hardware environment. This includes, but is not limited to:

- A description of any firmware updates or patches to the hardware.

- A description of any GUI configuration used to configure the system hardware.

- A description of exactly how the hardware is combined to create the complete system. For example, if the **SUT** description lists a base chassis with 1 processor, a processor update package of 3 processors, a NIC controller and 3 disk controllers, a description of where and how the processors, NIC and disk controllers are placed within the base chassis must be **reported** in the **Report**.

- A description of how the hardware components are connected. The description can assume the reader is knowledgeable of computer systems and the TPCx-V specification. For example, only a description that Controller 1 in slot A is connected to Disk Tower 5 is required. The reader is assumed to be knowledgeable enough to determine what type of cable is required based upon the component descriptions and how to plug the cable into the components.

8.3.1.5   A description of the steps taken to configure all software must be **reported** in the **Report**. Any and all configuration scripts or step-by-step GUI instructions are **reported** in the **Supporting Files** (see Clause 8.4.1.2). The description, scripts and GUI instructions must be sufficient such that a reader knowledgeable of computer systems and the TPCx-V specification could recreate the software environment. This includes, but is not limited to:

- A description of any updates or patches to the software.
- A description of any changes to the software.
- A description of any GUI configurations used to configure the software.

8.3.2 Clause 2 Database Design, Scaling & Population Related Items

8.3.2.1 A description of the steps taken to create the database for the **Reported Throughput** must be **reported** in the **Report**. No changes may be made to the database schema as created by the DDL and DML in the TPCx-V **Benchmark Kit**. The output of the setup.sh script must be captured and included in the supporting files. The distribution of tables, partitions and logs across all media must be explicitly depicted for the **Measured** and **Priced Configurations**.

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	2 X 36.4GB EEENNN Enclosure RAID 10	/	20.00GB	Root file system
2	2	4	6 X 36.4GB EEENNN Enclosure RAID 10	/pg_xlog	60.00GB	DB Log
3	2	4	14 X 74.8GB EEENNN Enclosure RAID 10	/dbstore	400.00GB	DB data tablespace
4	3	5	8 X 74.8GB EEENNN Enclosure RAID 10	/dbstore/tpcv-index	200.00GB	DB index tablespace

8.3.2.2 The methodology used to load the database must be **reported** in the **Report**.

8.3.3 Clause 3 Transaction Related Items

8.3.4 Clause 3 SUT, Driver, and Network Related Items

8.3.4.1 The **Network** configurations of both the **Measured** and **Priced Configurations** must be described and **reported** in the **Report**. This includes the mandatory **Network** between the **Driver** and **Tier A** (see Clause 10.1.2.2) and any optional **Database Server** interface networks (see Clause 10.1.1.3.15).

8.3.5 Benchmark Kit **Related Items**

8.3.5.1 The version of **Benchmark Kit** used in the benchmark must be **reported** in the **Report** (see Clause 10.7.3.1).

8.3.5.2 A statement that the required TPC-provided **Benchmark Kit** was used in the benchmark must be **reported** in the **Report**.

8.3.5.3 If the **Test Sponsor** modified **the Benchmark Kit**, a statement **that Benchmark Kit** has been modified must be **reported** in the **Report**. All formal waivers from the TPC documenting the allowed changes to **Benchmark Kit** must also be **reported** in the **Report** (see Clause 1.5.)

8.3.5.4 If the **Test Sponsor** extended **VGenLoader** (as described in Appendix 10.7.10.6), the use of the extended **VGenLoader** and the audit of the extension code by an **Auditor** must be **reported** in the **Report** (see Clause 10.7.6.4).

8.3.6   **Clause 5 Performance Metrics and Response Time Related Items**

8.3.6.1   The number of **VGenDriverMEE** and **VGenDriverCE** instances used in the benchmark must be **reported** in the **Report** (see Clause 10.2.3).

8.3.6.2   The **Measured Throughput** must be **reported** in the **Report** (see Clause 5.7.1.2).

8.3.6.3   The **Measured Throughput** of each **Group** must be reported, and be within 2% of its expected contribution to the aggregate **Measured Throughput** (see Clause 5.7.1.3).

8.3.6.4   A **Test Run Graph** of throughput versus elapsed wall clock time must be **reported** in the **Report** for the Trade-Result Transaction (see Clause 5.7.2).

8.3.6.5   The method used to determine that the **SUT** had reached a **Steady State** prior to commencing the **Measurement Interval** must be **reported** in the **Report**.

8.3.6.6   A description of how the work normally performed during a **Test Run**, actually occurred during the **Measurement Interval** must be **reported** in the **Report** (for example checkpointing, writing **Undo/Redo Log** records, etc.).

8.3.6.7   The recorded averages over the **Measurement Interval** for each of the **Transaction** input parameters specified by clause 5.4.1 must be **reported** in the **Report**.

8.3.7   **Clause 6 Transaction and System Properties Related Items**

8.3.7.1   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.

8.3.7.2   The **Test Sponsor** must **report** in the **Report** the Redundancy Level (see Clause 6.6.4.1) and describe the **Data Accessibility** test(s) used to demonstrate compliance. A list of all combinations of **Durable Media** technologies tested in Clause 6.6.3.5 must be reported in the **Report.**

8.3.7.3   A **Data Accessibility** Graph for each run demonstrating a Redundancy Level must be **reported** in the **Report** (see Clause 6.6.4.2).

8.3.7.4   The **Test Sponsor** must describe in the **Report** the test(s) used to demonstrate **Business Recovery**.

8.3.7.5   The **Business Recovery Time** Graph (see Clause 6.5.7.2) must be **reported** in the **Report** for all **Business Recovery** tests.

8.3.8   **Clause 7 Pricing Related Items**

8.3.8.1   Details of the **1 Business Day Space** computations along with proof that the database is configured to sustain a **Business Day** of growth (see Clause 5.6.6.1) must be **reported** in the **Report**.

8.3.8.2   The **Auditor's Attestation Letter,** which indicates compliance, must be included in the **Report**.

8.3.9   **Supporting Files Index Table**

An index for all files required by Clause 8.4 **Supporting Files** must be provided in the **Report**. The **Supporting Files** index is presented in a tabular format where the columns specify the following:
- The first column denotes the clause in the TPC Specification
- The second column provides a short description of the file contents

- The third column contains the path name for the file starting at the SupportingFiles directory.

If there are no **Supporting Files** provided then the description column must indicate that there is no supporting file and the path name column must be left blank.

Comment: This may be the common case for Clause 8.4.4 where **Benchmark Kit** modifications are required in the **Supporting Files**.

8.3.9.1   The following table is an example of the **Supporting Files** Index Table that must be **reported** in the **Report**.

Clause	Description	Pathname
Introduction	Database Tunable Parameters	SupportingFiles/Introduction/DBtune.txt
	OS Tunable Parameters	SupportingFiles/Introduction/OStune.txt
Clause 2	SupportingFiles/Clause2/setup.out	
Clause 3	There are no files required to be included for Clause 3.	
Clause 5	VGenValidate Output	SupportingFiles/Clause5/Validate.out
Clause 6	ACID Scripts	SupportingFiles/Clause5/runACID.sh
	Output of ACID tests	SupportingFiles/Clause6/ACID.out
Clause 7	1 Business Day Space Calculations	SupportingFiles/Clause7/space.xls
Clause 10	No **Benchmark Kit** modifications	
	No VGenLoader extensions	
	VGenDriver Configuration	SupportingFiles/Clause10/DriverConfig.txt
	VGenLoader Parameters	SupportingFiles/Clause10/LoaderParams.txt
	CCE 1 VGenLogger Output	SupportingFiles/ Clause10/CCE1.out
	CCE 2 VGenLogger Output	SupportingFiles/ Clause10/CCE2.out
	CMEE VGenLogger Output	SupportingFiles/ Clause10/CMEE.out
Clause 10.6	Broker-Volume Frames	SupportingFiles/Clause3/BrokerVolume.txt
	Trade-Order Frames	SupportingFiles/Clause3/TradeOrder.txt
	Trade-Result Frames	SupportingFiles/Clause3/TradeResult.txt

8.4   Supporting Files

The **Supporting Files** contain human readable and machine executable (i.e., able to be performed by the appropriate program without modification) scripts that are required to recreate the benchmark **Result**. If there is a choice of using a GUI or a script, then the machine executable script must be provided in the **Supporting Files**. If no corresponding script is available for a GUI, then the **Supporting Files** must contain a detailed step-by-step description of how to manipulate the GUI.

The directory structure under SupportingFiles must follow the clause numbering from the TPCx-V Standard Specification (i.e., this document). The directory name is specified by the 8.4 third level Clauses immediately preceding the fourth level **Supporting Files** reporting requirements. If there is more than one instance of one type of file, subfolders may be used for each instance

File names should be chosen to indicate to the casual reader what is contained within the file. For example, if the requirement is to provide the scripts for all table definition statements and all other statements used to set-up the database, file names of 1, 2, 3, 4 or 5 are unacceptable. File names that include the text "tables", "index" or "frames" should be used to convey to the reader what is being created by the script.

8.4.1 SupportingFiles/Introduction Directory

8.4.1.1 All scripts required to configure the hardware must be **reported** in the **Supporting Files**.

8.4.1.2 All scripts required to configure the software must be **reported** in the **Supporting Files**. This includes any **Tunable Parameters** and options which have been changed from the defaults in commercially available 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.

Comment: This requirement can be satisfied by providing a full list of all parameters and options.

8.4.2 SupportingFiles/Clause2 Directory

8.4.2.1 Outputs of the setup.sh script on all **VMs** of all **Groups** of all **Tiles** must be **reported** in the **Supporting Files**.

8.4.3 SupportingFiles/Clause3 Directory

8.4.3.1 No requirements

8.4.4 SupportingFiles/Clause4 Directory

8.4.4.1 If the **Test Sponsor** modified **Benchmark Kit**, the changes must be **reported** in the **Supporting Files**.

8.4.4.2 If the **Test Sponsor** extended **VGenLoader** (as described in Appendix 10.7.10.6), the extension code must be **reported** in the **Supporting Files**.

8.4.4.3 The **VGenLoader** parameters used must be **reported** in the **Supporting Files**.

8.4.4.4 The **VGenLogger** output for each CCE object, CMEE object and CDM object must be **reported** in the **Supporting Files** (see Clause 10.7.7.1).

8.4.5 SupportingFiles/Clause5 Directory

8.4.5.1 The output from **VGenValidate** must be **reported** in the **Supporting Files** (see Clause 5.7.4).

8.4.6 SupportingFiles/Clause6 Directory

8.4.6.1 The output of the **ACID** tests must be **reported** in the **Supporting Files**.

8.4.7 SupportingFiles/Clause7 Directory

8.4.7.1 A spreadsheet detailing the **1 Business Day Space** calculations must be reported in the **Supporting Files**.

CLAUSE 9 -- <u>INDEPENDENT AUDIT</u>

9.1   General Rules

9.1.1   Prior to its publication, a TPCx-V **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.)

9.1.2   All audit requirements specified in the version of the TPC Pricing Specification, located at www.tpc.org must be followed. For clarity and readability the TPC Pricing Specification requirements may be repeated in the TPCx-V Specification.

9.1.3   A generic audit checklist is provided as part of this specification. The **Auditor** may choose to provide the **Sponsor** with additional details on the TPCx-V audit process.

9.1.4   The generic audit checklist specifies the TPCx-V requirements that should be checked to ensure a TPCx-V **Result** is compliant with the TPCx-V Specification. The TPCx-V requirements may also be required to be **reported** in the **FDR**. Not only should the TPCx-V requirement be checked for accuracy but also the **Auditor** must ensure that the **FDR** accurately reflects the audited **Result**. For example, if the audit checklist indicates to "verify that a **Business Recovery Time** Graph is generated as specified", the graph must be verified to be accurate and verified to be the same graph that is **reported** in the **FDR** as specified by Clause 8.3.7.5.

9.1.5   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**.

9.1.6   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.

9.1.7   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 new **Result** must have the same hardware and software architecture and configuration as the referenced **Result**. 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.)

- 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: 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.

Comment 2: 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 3: 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.

9.2 Self-validation, Self-audit, and the role of the Auditor

Some of the requirement in this Clause, e.g. Clause 9.4, can be satisfied by verifying that the **Test Sponsor** has used the mandatory, TPC-supplied TPCx-V **Benchmark Kit** without any modifications.

The TPCx-V **Benchmark Kit** includes **Audit Tools** that perform many of the mechanical database audit tasks that are typically performed by an **Auditor**. The **Benchmark Kit** also automatically validates many of the numerical quantities that need to be checked after a **Test Run**, e.g., the **Transaction Mix**, **Transaction** input value mix requirements, **Transaction Response Times**, distribution of load among **Tiles** and **Groups**, etc.

It is expected that the numerical validation reports and the output of **Audit Tools** will greatly facilitate the work of an **Auditor**, and result in a faster, simpler, less costly audit process. Nonetheless, the tools are meant to assist the **Auditor** and simplify the audit process, not replace the need for an independent audit. The opinion of the **Auditor,** not the outputs of numerical validation or **Audit Tools**, ultimately determines whether a TPCx-V **Result** is compliant with the TPCx-V Specification.

9.2.1 Numerical validation by the Benchmark Kit

At the conclusion of a **Test Run**, the **Benchmark Kit** produces a number of files that contain various, detailed results from the run. The file audit_check.log file contains the results of checking of the following numerical quantities:

- Input Value Mix percentages (see Clause 5.4.1)

- The **Transaction Mix** (see Clause 5.3)

- **Response Time** requirements (see Clause 5.5) for each **Transaction** type in each **Phase** in each **Group** in each **Tile**.

- The reported Trade-Result throughput in each **Phase** in each **Group** in each **Tile**.

The benchmark kit tests every one of these conditions, and produces a PASSED or FAILED outcome to be used by the auditor in validating the Test Run. It is expected that a valid run will not have any FAILED results.

9.2.2   Audit Tools

At the conclusion of a **Test Run**, the test sponsor must use the xVAudit application of the **Benchmark Kit** to run the supplied database audit tests. These tests provide much of the data that the Auditor needs for verifying the requirements laid out in Clauses 9.3, 9.4, 9.7, and 9.8. Below is the list of xVAudit commands, and their primary use cases.

- The commands xVAudit.Atomicity.AtomicityAudit, xVAudit.Consistency.ConsistencyAudit, xVAudit.Isolation.Iso1Audit, xVAudit.Isolation.Iso2Audit, xVAudit.Isolation.Iso3Audit, and xVAudit.Isolation.Iso4Audit test the Atomicity, Consistency, and Isolation properties of the databases.

- The command xVAudit.Cardinality.TestBedCardinalityAudit audit TPCx-V table cardinalities at all the **Tiles** in the **SUT**.

- The command xVAudit.Schema.DatabaseStructureAudit produces a dump of the database schemas for verifying the requirements of Clause 9.3.1

- The command xVAudit.StoredProcs.StoredProcAudit produces a dump of the stored procedures for verifying the requirements of Clause 9.4.

- The commands xVAudit.Tables.DuplicatePrimaryKeyAudit, xVAudit.RI.RIAudit, and xVAudit.Tables.RangeMaxValueAudit are used to verify the requirements of Clause 9.3.1.7.

9.3   Auditing the Database

The **Auditor** must verify that the implementation of the measured database meets the TPCx-V Specification requirements. The **Auditor** may require the review of any and all source code and associated scripts or programs used to create and populate the database. The **Auditor** can require additional database verification not specified in the TPCx-V Specification to ensure the validity of the database.

9.3.1   Schema Related Items

9.3.1.1   Verify that the data types used to implement the columns of the TPCx-V required tables meet the requirements from Clause 2.2.1.

9.3.1.2   Verify that the data Meta-types used to implement the columns of the TPCx-V required tables meet the requirements of Clause 2.2.2.

9.3.1.3   Verify that the 9 tables in the Customer set have all of the required properties (see Clause 2.2.4).

9.3.1.4   Verify that the 9 tables in the Broker set have all of the required properties (see Clause 2.2.5).

9.3.1.5   Verify that the 11 tables in the Market set have all of the required properties (see Clause 2.2.6).

9.3.1.6   Verify that the 4 tables in the Dimension set have all of the required properties (see Clause 2.2.7).

9.3.1.7   Verify that all **Primary Key**s, all **Foreign Key**s, and all check constraints specified are maintained by the database (see Clause 2.2.3).

9.3.1.8   Verify that **Primary Key**s are not a direct representation of the physical disk addresses of the row (see Clause 10.3.8).

9.3.1.9   Verify that the implementation of the database satisfies the integrity rules (see Clause 10.4).

Comment: A check for the condition in clause 10.4.2 is not required, but the requirement still exists.

9.3.1.10   Verify that the implementation of the database satisfies the data access transparency requirements (see Clause 10.5).

9.3.2 Population Related Items

9.3.2.1   Verify that the version of **VGenLoader** used is compliant with the current version of the TPCx-V specification (see Clause 10.7.6.1).

9.3.2.2   Verify that none of the **VGenLogger** output contains "NO". A "NO" indicates that the associated **VGenDriver** or **VGenLoader** configuration parameter is not compliant with the current TPCx-V Specification (see Clause 10.7.2.8).

9.3.2.3   Verify that the database is populated using data generated by **VGenLoader** (see Clause 2.4.1.1).

9.3.2.4   Verify that the database is populated with an integral number of **Load Units** (see Clause 2.4.1.2).

9.3.2.5   Verify that the number of **Load Units** in each VM is compliant with the requirements in Clauses 2.4.1.2, 2.4.1.3, and 4.3.4.2.

9.3.2.6   Verify that the initial database population consists of a number of **Business Days** equal to **ITD** (see Clause 2.4.1.6).

9.3.2.7   Verify that the cardinality of the TPCx-V required tables in the initially populated database meets the requirements of Clause 2.4.1.

9.3.2.8   Verify that each non-**Growing Table** can grow by a number of rows equal to at least 5% of the table cardinality (see Clause 10.3.9).

9.4   Auditing the Transactions

The **Auditor** must verify that the implementation of the **Transactions** meets the TPCx-V Specification requirements. The **Auditor** may require the review of any and all source code and associated scripts or programs for the **Transactions**. The **Auditor** can require additional **Transaction** verification not specified in the TPCx-V Specification to ensure the validity of the **Transactions**.

9.4.1   Verify that the implementation of each **Transaction** specified in Clause 3.3 is compliant with its respective input parameters, output parameters, **Database Footprint** and **Frame Implementation** requirements. More specifically verify that the stored procedures and the **Frame Implementation** in the TPCx-V **Benchmark Kit** have not been modified.

9.5   Auditing the SUT, Driver and Networks

The **Auditor** must verify that the implementation of the test environment meets the TPCx-V Specification requirements. The **Auditor** may require the review of any and all source code implementing the various components involved and associated scripts or programs. The **Auditor** can require additional verification not specified in the TPCx-V Specification to ensure the validity of the test environment.

9.5.1   Verify the presence and use of a **Network** to communicate between the **Driver** and **Tier A** (see Clause 10.1.3.1.6).

9.5.2   Verify that the restrictions on operator interventions are met (see Clause 4.3.3).

9.6 Auditing Benchmark Kit

9.6.1 Verify that the version of **Benchmark Kit** used is compliant with the version of the TPCx-V specification used for publication (see Clause 10.7.3).

- Verify that the **VGenSourceFiles** used have not been modified (see Clause 10.7.5).
- If the **Test Sponsor** modified **Benchmark Kit** in response to a formal waiver issued by the TPC, verify that the changes fall under the scope of the waiver (see Clause 1.5.7).
- If the **Test Sponsor** modified **Benchmark Kit** 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 **Benchmark Kit** (see Clause 1.5.6).

9.7 Auditing the Execution Rules and Metrics

The **Auditor** must verify that all TPCx-V execution rules have been followed by the **Test Sponsor**. The **Auditor** may require the review of any and all output of the benchmark environment. The **Auditor** can require additional verification not specified in the TPCx-V Specification to ensure the validity of the Benchmark Execution Rules and the resulting **Reported Throughput**.

9.7.1 Pre-run Configuration Items

9.7.1.1 Verify that the contents of the database meet the requirements of Clause 5.6.2.1 and Clause 5.6.2.3.

9.7.1.2 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) (see Clause 5.6.2.2).

9.7.1.3 Verify that no executions of the Trade-Cleanup **Transaction** occur during the **Test Run** (see Clause 5.6.1.1).

9.7.1.4 Verify that the system clocks are synchronized as required by Clause **Error! Reference source not found.**.

9.7.2 Runtime Configuration Items

9.7.2.1 Verify that, for specific global inputs, each instance of the **CE**, **DM** and the **MEE** is using the same values as those used by the **VGenLoader** instances during the initial database population (see Clause 10.7.7.4). This requirement 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.**

9.7.2.2 Verify that none of the **VGenLogger** output contains "NO". A "NO" indicates that the associated **VGenDriver** or **VGenLoader** configuration parameter is not compliant with the current TPCx-V Specification (see Clause 10.7.2.8).

9.7.3 Runtime Data Generation Items

9.7.3.1 Verify that the **reported Transaction Mix** over the **Measurement Interval** only counts **Valid Transactions** (see Clause5.3).

9.7.3.2   Verify that the **reported Transaction Mix** over the **Measurement Interval** excludes the Data-Maintenance **Transactions** (see Clause 5.3.1).

9.7.3.3   Verify that the specified mix of **Transactions** over the **Measurement Interval** meets the requirements (see Clause 5.3.1).

9.7.3.4   Verify that the **reported Transaction Mix** over the **Measurement Interval** is computed and **reported** with the required precision and rounding (see Clause 5.3.2).

9.7.3.5   Verify that the **CE Driver** generated input data with a random variability that stays within the specified ranges (see Clause 5.4.1).

9.7.3.6   Verify that the number of **Load Units** configured for the database is equal to the number of **Load Units** actually accessed during the **Test Run** (see Clauses 2.4.1.7 and 5.6.8.7).

9.7.4   Response Time Items

9.7.4.1   Verify that the **Transaction Response Time**s meet the requirements of Clause 5.5.1.2.

9.7.4.2   Verify for each type of Transaction that its average **Response Times** does not exceed its 90th percentile **Response Time** (see Clause 5.5.1.4).

9.7.5   Throughput Items

9.7.5.1   Verify that each **Measured Throughput** is between 80% and 102% of the corresponding **Nominal Throughput** (see Clause 5.7.1.2).

9.7.5.2   Verify that the **Reported Throughput** is not greater than the **Nominal Throughput** (see Clause 5.7.1).

9.7.6   Data-Maintenance Items

9.7.6.1   Verify that one, and only one, Data-Maintenance **Transaction** generator is used during the **Test Run** (see Clause 10.7.7.7.2).

9.7.6.2   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 5.3.3).

9.7.6.3   Verify that the Data-Maintenance **Transaction** modified the rows specified in Clause 10.6.11.

9.7.7   Steady State Items

9.7.7.1   Verify that the **Steady State** meets the requirements of **Sustainable** performance as specified by Clause 5.6.3.

9.7.7.2   Verify that all events performed at regular intervals during **Steady State** are present before and during the **Measurement Interval** as required (see Clause 5.6.4.1) and that the duration of **Steady State** meets all the requirements listed in Clause 5.6.4.2.

9.7.7.3   Verify that the **Measurement Interval** meets all the requirements of Clause 5.6.5.

9.7.8   VGenValidate Items

9.7.8.1   Verify that output from **VGenValidate** indicates "Passed!" (see Clause 5.7.4).

9.7.9 Space Calculation Items

9.7.9.1 Verify that the **Data Growth** is computed as specified and that sufficient space to accommodate it is available on-line (see Clause 5.6.6).

9.8 Auditing the ACID Tests

The **Auditor** must verify that the implementation of the **ACID** tests sufficiently demonstrates compliance with the TPCx-V **ACID** requirements. The **Auditor** may require the review the source code implementing these tests and any associated scripts or programs. The **Auditor** can require additional verification not specified in the TPCx-V Specification to ensure the validity of the **ACID** tests.

9.8.1 Atomicity Items

9.8.1.1 Verify that the atomicity test is implemented as specified in Clause 6.2.2.

9.8.1.2 Verify that the atomicity test correctly demonstrates the atomicity property (see Clause 6.2.1).

9.8.2 Consistency Items

9.8.2.1 Verify that the consistency tests are implemented as specified in Clause 6.3.3.

9.8.2.2 Verify that the consistency conditions are successfully demonstrated by the tests (see Clause 6.3.2)

9.8.3 Isolation Items

9.8.3.1 Verify that the isolation tests are implemented as specified in Clause 6.4.2.

9.8.3.2 Verify that the isolation tests correctly demonstrate the isolation requirements (see Clause 6.4.1.3).

9.8.4 Data Accessibility Items

9.8.4.1 Verify that the **Durability** tests for **Data Accessibility** are implemented as specified (see Clause 6.6.3.5).

9.8.4.2 Verify that the Redundancy Level chosen by the **Sponsor** is successfully demonstrated by the **Data Accessibility** test (see Clause 6.6.3.5).

9.8.4.3 Verify that the Redundancy Level chosen by the **Sponsor** is correctly **reported** in the **Report** (see Clause 6.6.3.4).

9.8.4.4 Verify that a **Data Accessibility** Graph is generated as specified in Clause 6.6.4.2.

9.8.4.5 Verify that all components of **Durable Media** technologies tested in Clause 6.6.3.5 are correctly reported in the **Report**

9.8.5 Business Recovery Items

9.8.5.1 Verify that the **Durability** tests for **Business Recovery** are implemented as specified (see Clause 6.5.7).

9.8.5.2 Verify that recovery from each required single failure scenario is successfully demonstrated by one or more **Business Recovery** tests (see Clause 6.5.7).

9.8.5.3 Verify that the **Business Recovery Time** correctly measures the time between the start of **Business Recovery** and the end of **Business Recovery** (see Clause 6.5.5.10).

9.8.5.4   Verify that a **Business Recovery** Graph is generated as specified in Clause 6.5.7.2.

9.9   Auditing the Pricing

9.9.1   Rules for auditing Pricing information are specified in the effective version of the TPC Pricing Specification, located at www.tpc.org.

9.9.2   Verify that the greater of the **1 Business Day Space** or the data storage configured during the measurement is included in the **Priced Configuration** (see Clause 7.2).

9.9.3   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 7.3.1 and Clause 7.3.2).

9.9.4   Verify that all component **Substitutions** are compliant with the TPC Pricing Specification and with the TPCx-V specific restrictions (see Clause 7.4).

9.10   Auditing the FDR

For the Audit requirements specified in Clauses 9.6 through 9.9, the **Auditor** must ensure that if required by Clause 8 -- , the items, requirements or values are correctly **reported** in the **FDR**.

For those items, requirements or values that are **reported** in the **FDR** and not required to be audited, the **Auditor** need only ensure that they are in the **FDR** and appear to be reasonable. For example, the **Auditor** cannot be held responsible for accuracy of the **Availability Date** but can ensure that it is **reported** in the **FDR** and does not fall outside the 6-month availability window starting from the publication date.

9.10.1   Verify that table partitioning, if used, meets the requirements from Clause 10.3.3.

9.10.2   Verify that the **reported Transaction Mix** over the **Measurement Interval** is computed and **reported** with the required precision and rounding (see Clause 5.3.2).

9.10.3   Verify that the **Reported Test Run Graph** meets the requirements (see Clause 5.7.2).

9.10.4   Verify that the **Executive Summary Statement** is accurate and complies with the reporting requirements as specified in Clause 8.2.

9.10.5   For those items that are required by Clause 8.3 to be **reported** in the **Report** and are also required by Clauses 9.6 through 9.9 to be verified by the **Auditor**, verify that the items are accurately **reported** in the **Report**. For those items that are required to be **reported** by Clause 8.3 but are not required to be verified by the **Auditor**, ensure that the items are **reported** in the **Report** and appear to be reasonable.

9.10.6   Verify that the **Supporting Files** specified by Clause 8.4 exist and appear to be reasonable.

9.10.7   Verify that the following sections of the FDR are accurate:

- Verify that the diagram illustrating the **Measured Configuration** is accurate (see Clause 8.3.1.2)
- Verify that the diagram illustrating the **Priced Configuration** is accurate (see Clause 8.3.1.2)
- Verify that the textual descriptions required by Clause 8.3.2 are accurate.
- Verify that any **Benchmark Kit** changes made by the **Sponsor comply with the requirements listed in Clause 1.5**, and are **reported** in detail in the **FDR** (see Clause 8.3.5.3).

9.10.8   A complete review of the **Report** by the **Auditor,** beyond the sections listed above, can be requested by the **Sponsor,** but is not required.

CLAUSE 10 -- TPCX-V BENCHMARK KIT DESIGN DOCUMENT

10.1 Description of SUT, Driver, and Network

10.1.1 Overview

TPCx-V is a distillation of an abstraction of multiple virtualized "real-world" OLTP environment. In order to understand what TPCx-V tests and, as a consequence, what TPCx-V does not test, it is necessary to understand the base "real-world" environment (Clause 10.1.1.1 Description of Real-World OLTP Environment), the abstraction of that base environment (Clause 10.1.1.2 Functional Component Abstraction of the Real-World OLTP Environment) and the distillation of that abstraction (Clause 10.1.1.3 Distillation of Functional Components into the TPCx-V Environment).

10.1.1.1 Description of the Real-World OLTP Environment

The figure below shows the "real-world" environment upon which TPCx-V 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).
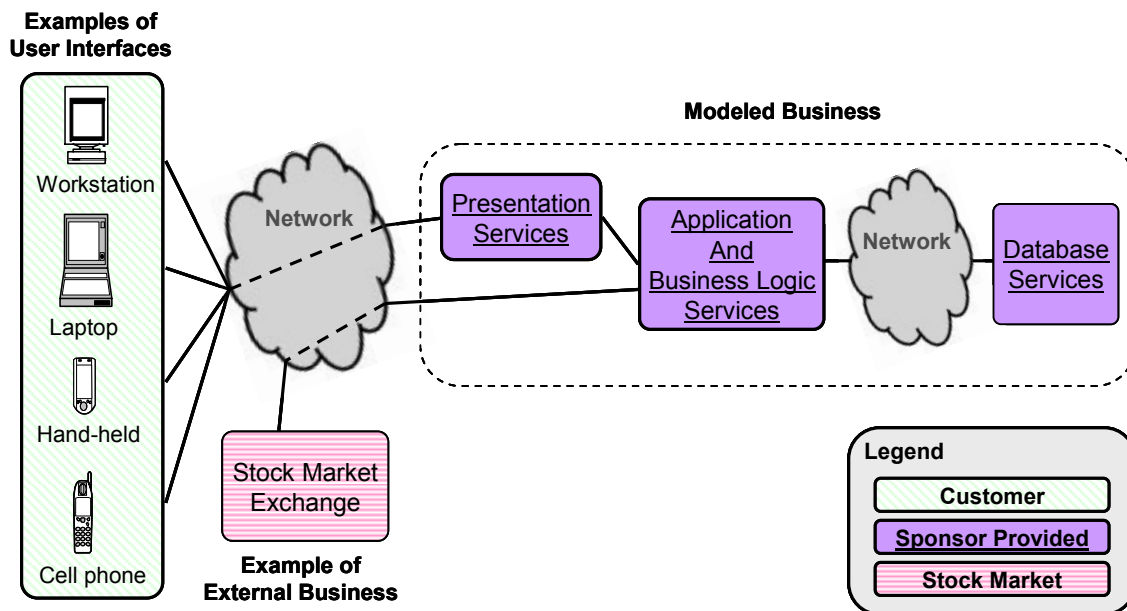
Figure 10.a - Diagram of the Real-World OLTP Environment

10.1.1.2 Functional Component Abstraction of the Real-World OLTP Environment

From the diagram of the real-world OLTP environment, the following diagram of the key functional components can be abstracted.
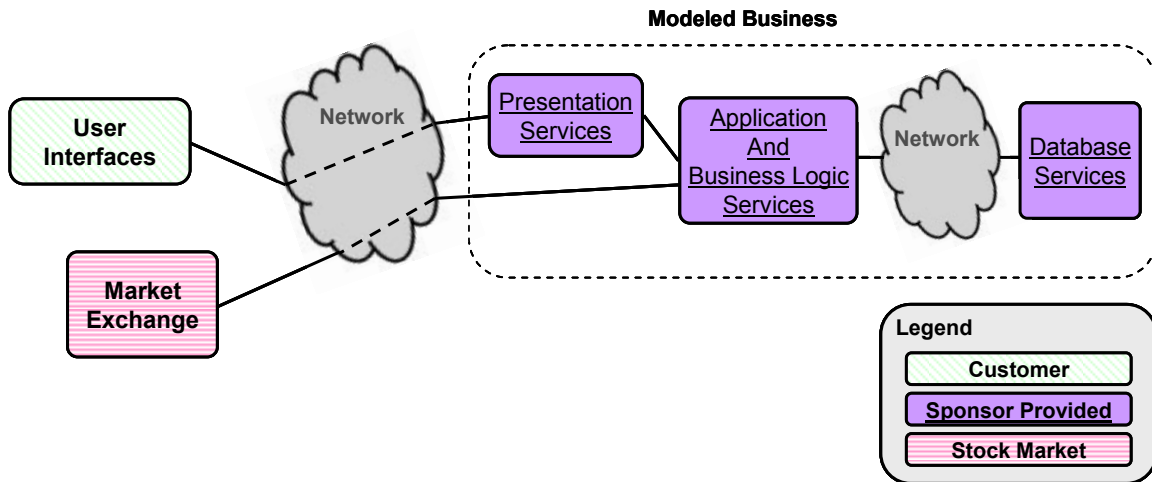
Figure 10.b - Abstraction of the Functional Components in an OLTP Environment

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.

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.

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.

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.

10.1.1.3   Distillation of Functional Components into the TPCx-V Environment

By design, TPCx-V virtualized business model 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 TPCx-V 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.

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 TPCx-V 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.

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, TPCx-V does not define or require 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).

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.
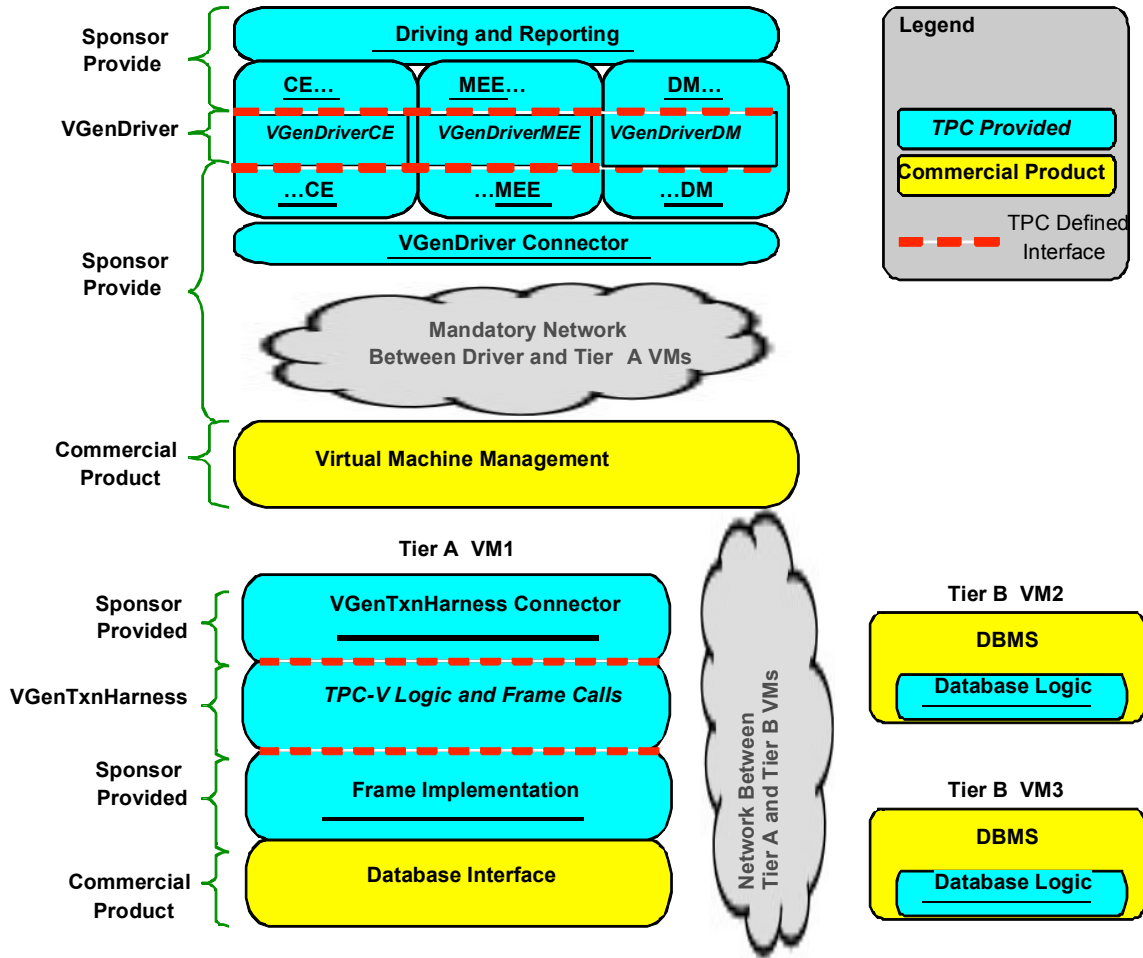
Figure 10.c - Functional Components of the Test Configuration

10.1.1.3.1 **Driving & Reporting** – The **TPC** provided **Benchmark Kit** includes functionality to set up, administer and execute a **Test Run**, collect data and generate summary reports. The **TPC** provided kit invokes **VGenDriver** to generate input parameter for transactions according to this specification. The **Benchmark Kit** also performs validation of the generated results.

10.1.1.3.2 **CE** – **TPC** provided functionality to set up, administer and execute the **Customer Emulator**. The **TPC** written kit invokes **VGenDriverCE**.

10.1.1.3.3 **MEE** – **TPC** provided functionality to set up, administer and execute the **Market-Exchange Emulator**. The **TPC** written kit invokes **VGenDriverMEE**.

10.1.1.3.4 **DM** – **TPC** provided functionality to set up, administer and execute the Data-Maintenance **Transaction** once a minute. The **TPC** written kit invokes **VGenDriverDM**. The **Benchmark Kit** also provides functionality to call the Trade-Cleanup **Transaction** once prior to the start of the run (see description of **VGenDriverDM** below).

10.1.1.3.5 A **TPC Defined Interface** is a C++ class member that is designed to exchange data (and transfer execution control) between various components of the TPC provided **Benchmark Kit**. The table in appendix A.14 lists the **TPC Defined Interfaces** and the associated C++ classes and member functions.

10.1.1.3.6 **VGenDriver** – TPC provided C++ source code that implements essential functionality during a **Test Run**. The use of **VGenDriver** is mandatory. The following are parts of **VGenDriver**.

- **VGenDriverCE** – **Customer Emulator** that provides the required **Transaction Mix** and user input data generation
- **VGenDriverMEE** – **Market Exchange Emulator** that provides the stock market functionality and data generation
- **VGenDriverDM** – Data-maintenance functionalities that generates data for and invokes the Data-Maintenance **Transaction**. Also, supplies an interface that can be used by the **Benchmark Kit** to invoke the Trade-Cleanup **Transaction**.

10.1.1.3.7 **VGenDriver Connector** – **TPC** provided functionality that complies with a **TPC Defined Interface**. The **VGenDriver Connector** is invoked from inside **VGenDriver** through the interface. The **VGenDriver Connector** is responsible for sending the **VGenDriver** generated data to, and receiving the corresponding resultant data back from, the **VGenTxnHarness Connector** via the **Network**. An example of the hardware and software needed to implement the Connector is:

- **TPC provided** 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**).

10.1.1.3.8 **A** Network **is** defined as 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.

10.1.1.3.9 TPC/IP over a local area network is an example of an acceptable Network implementation.

10.1.1.3.10 **Virtual Machine Management Software (VMMS) –** Commonly referred to as a Hypervisor, a commercially available framework or methodology of dividing the resources of a system into multiple computing environments. Each of these computing environments allows a completely isolated software stack including an operating system to run in complete isolation from anything else running on the system. The **VMMS** allows for the creation of multiple computing environments on the same system.

Comment: The term VMMS is not meant to include the static partitioning of a system that occurs at boot time or any dynamic partitioning that may take place through operator intervention.

10.1.1.3.11 **Virtual Machine (VM)** – A self-contained operating environment, managed by the VMMS, that behaves as if it were a separate computer.

10.1.1.3.12 **VGenTxnHarness Connector** – **TPC** provided functionality responsible for receiving the data sent from, and sending the appropriate resultant data back to, the **VGenDriver Connector** via the **Network**. The **VGenTxnHarness Connector** provides the data to, and accepts the resultant data from, **VGenTxnHarness** by invoking a **TPC Defined Interface**. The **VGenDriver Connector** example implementation above applies here as well.

10.1.1.3.13 **VGenTxnHarness** – TPC provided C++ source code that implements essential functionality during a **Test Run**. **VGenTxnHarness** invokes the **TPC's** implementations of the **Transaction Frames**, providing the necessary inputs and accepting the necessary outputs through a **TPC Defined Interface**. The use of **VGenTxnHarness** is mandatory.

10.1.1.3.14 **Frame Implementation** is **TPC** provided functionality that accepts inputs from, and provides outputs to, **VGenTxnHarness** 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).

10.1.1.3.15 **Database Interface** is a 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.

10.1.1.3.16 **A Database** Server is a commercially available product(**s**). TPC 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

10.1.1.3.17 **Database Logic** is **TPC** provided **Frame implementation** logic (e.g. stored SQL procedure)

Comment: VGenDriver Connector and **VGenTxnHarness 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.

10.1.2   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**.
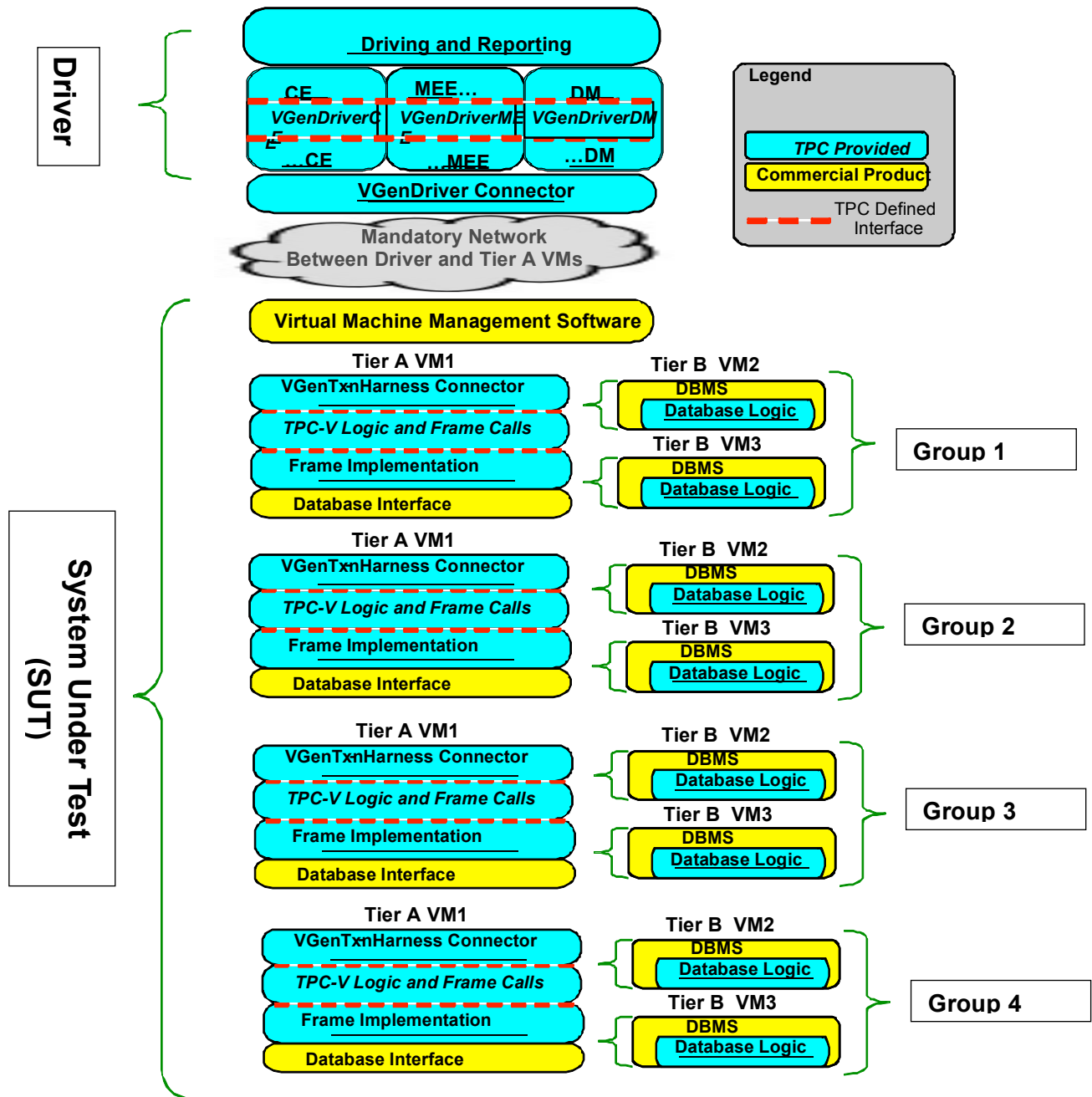
Figure 10.d - Defined Components of the Test Configuration

The clauses below define some terms used in this specification. A TPCx-V configuration has a single instance of some components, e.g. the driver, and multiple of others, e.g., Tier B.

10.1.2.1   **The Driver** – is defined to be all hardware and software needed to implement the Driving & Reporting, **VGenDriver** and up-stream Connector functional components.

10.1.2.2   The use of a **Network** (as defined in Clause 10.1.1.3) between the **Driver** and **Tier A** is mandatory.

10.1.2.3 The use of commercially available **Virtual Machine Management Software (VMMS)** product (as defined in Clause 10.1.1.3) is mandatory.

10.1.2.4 **Virtual Machine** (**VM**) is defined as: A **Virtual Machine (VM)** is a self-contained operating environment, managed by the **VMMS**, and that behaves as if it were a separate computer (as defined in Clause 10.1.1.3). TPCx-V requires that there shall be three **VM**s per **Group**: one **Tier A VM** and two transactional specific **Tier B VM**s.

10.1.2.5 **Tier A** is defined as: **Tier A** consists of all hardware and software needed to implement the down-stream Connector, **VGenTxnHarness**, **Frame Implementation** and **Database Interface** functional components.

10.1.2.6 **Tier B** is defined as: **Tier B** consists of all hardware and software needed to implement the **Database Server** functional components, encapsulated within two transaction-specific **Virtual Machines**, contained within the same **Group**. This includes data storage media sufficient to satisfy the initial database population requirements of Clause 2.4.1 and the **Business Day** growth requirements of Clause 5.6.6.4 and Clause 5.6.6.5.

10.1.2.7 **Tile** is defined as: **Tile** is the unit of replication of TPCx-V configuration and load distribution. Each **Tile** consists of 4 **Groups**. A valid TPCx-V configuration may have between 1 and 6 **Tiles**, with all **Tiles** contributing identical proportions of the total load. The number of **Tiles** and the number of **Load Units** configured in the initial populations of the databases in each **Group** are dependent on the **Nominal Throughput**, and are determined by a formula defined in Clause 4.3.4.

10.1.2.8 **Group** is defined as: Each **Tile** has four **Groups**, with **Groups** 1, 2, 3, and 4 contributing an average of 10%, 20%, 30%, and 40% of the total throughput of the **Tile**, respectively. Each **Group** consists of one **Tier A Virtual Machine** and two transaction-specific **Tier B Virtual Machines**.

10.1.2.9 **System Under Test** is defined as: **System Under Test (SUT)** is the total collection of all hardware and software components in all **Tiles**, to include their **Tier A** and **Tier B** Virtual Machines.

10.1.2.10 **Measured Configuration** - See **System Under Test.**

10.1.3 **Further Requirements for SUT and Driver Implementations**

10.1.3.1 **Restrictions on the Driver**

The purpose of this section is to limit the knowledge (or use of the knowledge by the **Driver**) of the **SUT**, the contents of the databases and the transactions.

10.1.3.1.1 During the **Test Run** the **TPC provided** code to implement the **Driver** must not:

- make decisions based upon the contents of the databases (including **VGenInputFiles**)
- provide information to the **SUT or any of the VMs** that results in a performance advantage

10.1.3.1.2 The **no-peeking-in-the-packet** rule: Data predicated routing (based on the content of the packet) in **VGenDriver Connector** or **VGenTxnHarness Connector** is not allowed. Data predicated routing (based on the Transaction type of the packet only) in VGenTxnHarness Connector is allowed for Transaction routing of Trade-Lookup and Trade-Update to VM2 and all other Transactions to VM3. No other packet data access usage is allowed in VGenTxnHarness Connector.

10.1.3.1.3 The **TPC provided** code executed between **VGenDriver** (i.e. the following APIs: CESUTInterface, MEESUTInterface, DMSUTInterface) and the mandatory Network may not make any decision related to routing, timing, reordering or pacing of that **Transaction** or any other **Transaction** based on that **Transaction's** type or input values.

> **Comment:** These restrictions include direct knowledge (e.g., obtained by peeking in the packet) or implied knowledge (e.g., obtained by card counting, message size, etc.).

10.1.3.1.4 Any **TPC provided** code that sends a market request from the **SUT** to the **Driver** (i.e. SendToMarketInterface) may not make any decisions related to routing, timing, reordering, or pacing of that request or any other request based on that request's input values.

> **Comment:** These restrictions include direct knowledge or implied knowledge.

10.1.3.1.5 The TPCx-V model allows the **Frame Implementation** within **Tier A** to select VM2 or VM3 as the destination of a transaction based on the transaction types described in Clause 5.3.1. Otherwise, if routing is done within a **Frame Implementation**, a transaction monitor must perform the routing (see Clause 3.2.1.9). The **Sponsor's** implementation of *SendToMarketFromFrame* interface is not governed by this clause but the implementation still must conform to Clause 10.1.3.1.5

10.2 Driver Implementation Architectures

The driver architecture has an impact on understanding and interpreting the benchmark execution rules. Therefore, this section provides an overview of key architectural modules. These models are examples only and do not represent an exhaustive list. For simplicity, the focus will be on the **CE**, but the same principles apply to the **MEE** as well.

10.2.1 The Simple CE

In its simplest form, the **CE** has:

- A single thread of execution
- A single instance of the CCE class (i.e. a **VGenDriverCE** of size 1)
- A single blocking **Network** connection to the **SUT**

During the **Test Run**, the **CE** cycles through a process of calling from **Sponsor** provided code into **VGenDriverCE** code to generate the next **Transaction** type and the necessary input data, calling from the **VGenDriverCE** code into **Sponsor** provided code to record the **Transaction's** start time, send the input data to the **SUT**, wait for the **Transaction** to execute, receive in the output data from the **SUT**, record the **Transaction's** end time, and then finally return from the **Sponsor** code back through the **VGenDriverCE** code back to the initial **Sponsor** code. The following diagram captures this pictorially.

Customer Emulator | **VGenDriverCE** | **CESutInterface/ VGenDriver Connector**

DoTxn()

DoTxn()
  –Generate Txn Type
  –Generate Txn Inputs
  –CESUTInterface::TxnType

TxnType()
  –Record Start Time sT_n
  –Send data to SUT
  –Wait for Response
  –Receive data from SUT
  –Record End Time eT_n

Legend

TPC Provided
Commercial Product
TPC Defined Interface
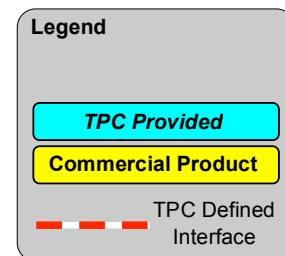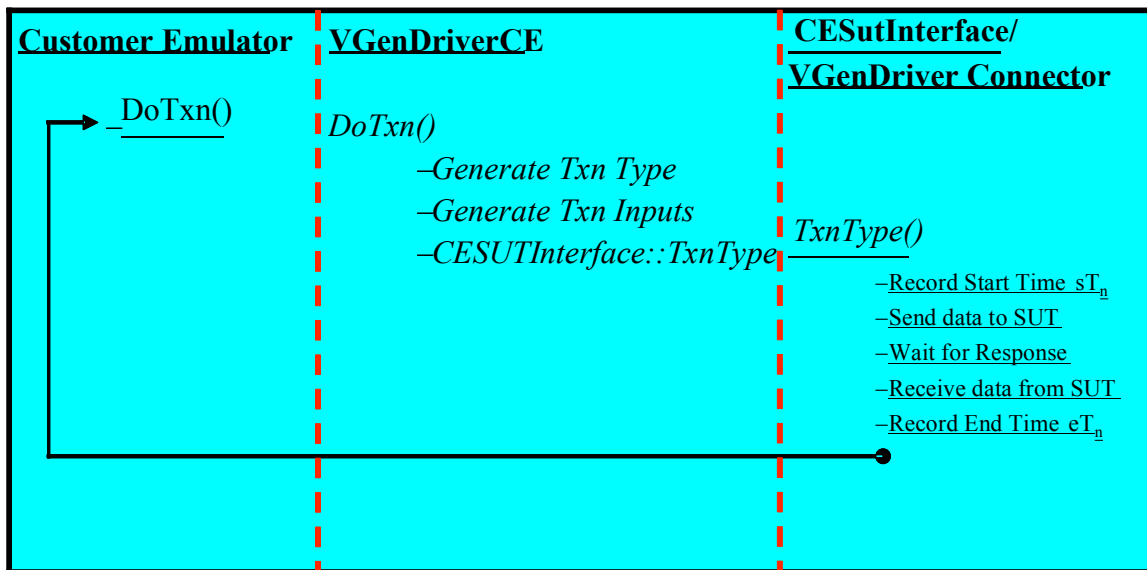
Figure 10.e - The Simple CE

10.2.2   The Replicated CE

There are limits to the amount of throughput the Simple **CE** can generate. So replication of the Simple **CE** is permitted. This allows multiple copies of the Simple **CE** to generate the necessary **Nominal Throughput** for any size database. Since there will be multiple instances of the CCE class, this is equivalent to a **VGenDriverCE** of size N (where N is the number of CCE instances).

The mandatory use of **VGenDriverCE's** auto-RNG seeding (see Clause 10.7.7.2) means that these will not be exactly identical copies of the Simple **CE**. Each copy will start off at a different point in the RNG stream. The following diagram shows the Replicated **CE**.

VGenDriverCE

| Customer Emulator | VGenDriverCE | CESutInterface / |
| Customer Emulator | VGenDriverCE | CESutInterface / |

Customer Emulator   **VGenDriverCE**   **CESutInterface /**
VGenDriver Connector

–DoTxn ()

Txn()
  –Generate Txn Type
  –Generate Txn Inputs
  –CESUTInterface::TxnType

TxnType ()
  –Record Start TimesT_n
  –Send data to SUT
  –Wait for Response
  –Receive data from SUT
  –Record End TimeeT_n

Legend

TPC Provided

Commercial Product

- - - - TPC Defined
Interface

Figure 10.f The Replicated CE

10.2.3 Driver Reporting Requirements

The TPCx-V Express **Benchmark Kit reports** the number of **VGenDriverMEE** and **VGenDriverCE** instances used in the benchmark in the **Report**.

10.3 Implementation Rules

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

10.3.2 All TPCx-V required tables must have the properly scaled number of rows as defined by the database population requirements in Clause 2.4.

10.3.3 Table Partitioning

10.3.3.1 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**.

10.3.3.2   Vertical partitioning of tables is allowed. Groups of columns of one table may be assigned to files, disks, or areas different from those storing the other columns of that table. If implemented, the details of such partitioning must be **reported** in the **Report** (see Clause 10.5 for limitations).

10.3.3.3   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 column boundaries), is not considered partitioning. For example, distribution or striping 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 software without knowledge of the logical structure stored in the physical file.

10.3.4   Replication is allowed for all tables. All copies of TPCx-V tables that are replicated must meet all requirements for atomicity, consistency, and isolation as defined in Clauses 6.2, 6.3 and 6.4. If implemented, the details of such replication must be **reported** in the **Report**.

Comment: Only one copy of a replicated TPCx-V table needs to meet the **Durability** requirements defined in Clause 6.5.

10.3.5   Columns may be added and/or duplicated from one TPCx-V table to another as long as these changes do not improve performance.

10.3.6   Each TPCx-V column, as described by the table definitions in Clause 2.2, must be logically discrete and independently accessible by the **DBMS**. For example, ADDRESS.AD_LINE1 and ADDRESS.AD_LINE2 are not allowed to be implemented as two sub-parts of a single column ADDRESS.AD_LINE.

10.3.7   Each TPCx-V column, as described by the table definitions in Clause 2.2, must be accessible by the **DBMS** as a single column. For example, NEWS_ITEMS.NI_ITEM is not allowed to be implemented as two separate columns NEWS_ITEMS.NI_ITEM1 and NEWS_ITEMS.NI_ITEM2.

10.3.8   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 column(s) specified in the transaction **Profile** and no other columns. Initial access includes insertion, deletion, retrieval, and update of any row.

10.3.9   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 (and corresponding growth in associated **User-Defined Objects**, such as indices) be configured for the **Test Run** and priced (as **Fixed Space** per Clause 5.6.6.2) 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.

10.3.10   The implementation of the BLOB 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 BLOBs, to the same point in time.
- The object, and any associated references to it, must be treated as a unit with respect to atomicity.

Comment: The implementation of BLOB in the NEWS_ITEM table may be implemented either by specific inclusion of the BLOB in the table or by use of a reference to a BLOB object stored elsewhere on the **System Under Test**.

10.3.11   User-Defined Objects

Any object defined in the database is considered a **User-Defined Object**, except for the following:
- a TPCx-V Table (see clause 2.2.3)
- a required **Primary Key** (see clause 2.2.3.1)
- a required **Foreign Key** (see clause 2.2.3.2)
- a required constraint (see clause 2.2.3.3)
- **Database Metadata**

10.3.11.1 There are no restrictions on **User-Defined Objects**, provided that:

- all **Transaction** and **Frame** implementation rules from clause 3.2 are met
- all **ACID** requirements in clause 7 are met

10.4   Integrity Rules

10.4.1   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.

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

10.4.3   **Referential Integrity** (RI) must be enforced by the database for all **Foreign Key** (FK) and **Primary Key** (PK) relations defined between TPCx-V tables.

Comment: Referential Integrity preserves the relationship of data between tables, by restricting actions performed on **Primary Keys** 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 without also removing the rows with corresponding/referencing **Foreign Keys**. **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 without also modifying the corresponding/referencing **Foreign Keys** to be equal to the new **Primary Key**.

10.5   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 tests can prove that the system supports complete data access transparency. The requirements below describe the minimum capabilities needed to establish that the system provides transparent data access.

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.

10.5.1   Each of the tables described in Clause 2.2 (and any additional tables used in the implementation of the **Transactions**) 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.

10.5.2   The system must prevent any data manipulation operation performed using the names described in Clause 10.5.1 that would result in a violation of the integrity rules (see Clause 10.4). For example: the system must prevent a non-TPCx-V application from committing the insertion of a row in a vertically partitioned table unless all partitions of that row have been inserted.

10.5.3   Using the names which satisfy Clause 10.5.1, any arbitrary non-TPCx-V 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 10.5.1 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 TPCx-V **Application Program** uses general-purpose mechanisms to manipulate data in the database.

10.6   The Transactions

10.6.1   The Broker-Volume Transaction

The Broker-Volume **Transaction** is designed to emulate a brokerage house's "up-to-the-minute" internal business processing. An example of a Broker-Volume **Transaction** would be a manager generating a report on the current performance potential of various brokers.

Broker-Volume is invoked by **VGenDriverCE**. It consists of a single **Frame**. The **Transaction** searches the pending limit orders to find orders that are associated with a given list of brokers responsible for stocks of a given sector. The value of each order is calculated based upon bid price and quantity of shares and added to the running total volume for the appropriate broker. The list of brokers with their associated total volume sorted in descending volume order is returned.

10.6.1.1   Broker-Volume Transaction Parameters

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

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:

Parameter	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.
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.
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.

10.6.1.2   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

10.6.1.3   Broker Volume Transaction Frame 1 of 1

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

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

```
{
    invoke (Broker–Volume_Frame–1)
    if (list_len < 0) or (list_len > max_broker_list_len) then
    {
        status = –111
    }
}
```

Broker-Volume Frame 1 of 1 Parameters:

Parameter	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.
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 Frame.
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
    where
        TR_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

    // row_count will frequently be zero near the start of a Test Run when
    // TRADE_REQUEST table is mostly empty.
    list_len = row_count
    commit transaction
```

```
Broker-Volume_Frame-1 Pseudo-code: Broker Volume
```

```
}
```

10.6.2   The Customer-Position Transaction

The Customer-Position **Transaction** is designed to emulate the process of retrieving the customer's profile and summarizing their overall standing based on current market values for all assets. This is representative of the work performed when a customer asks the question "What am I worth today?"

Customer-Position is invoked by **VGenDriverCE**. It consists of three **Frames**, (**Frame** 2 and 3 are mutually exclusive). The customer is specified either by a customer ID or a customer tax ID. If the customer ID passed into the **Transaction** is 0, then the customer tax ID is used to look up the customer ID. Detailed information about the customer's profile is retrieved. In addition, for each of the customer's accounts, the cash balance of the account and the total current market value of all holdings in the account are returned.

If a history of trading activity has been requested, information is retrieved on the ten most recent trades for a randomly chosen account among the customer's accounts.

10.6.2.1   Customer-Position Transaction Parameters

The inputs to the Customer Position **Transaction** are generated by the **VGenDriverCE** code in CETxnInputGenerator.cpp and the data structures defined in TxnHarnessStructs.h must be used to communicate the input and output parameters.

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:

Parameter	Direction	Description
acct_id_idx	IN	Index to one of the customer's 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 1 if Frame 2 is to be invoked or 0 if not.
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_ctry_1	OUT	Country code for customer's first phone number.
c_ctry_2	OUT	Country code for customer's second phone number.
c_ctry_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.
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.
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 for each event from history.

10.6.2.2 Customer-Position Transaction 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		

Table	Column			
	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

10.6.2.3   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 **VGenTxnHarness** controls the execution of **Frame** 1 as follows:

```
{
    invoke (Customer-Position_Frame-1)
    if (acct_len < 1) or (acct_len > max_acct_len) then
    {
        status = -211
    )
}
```

Customer-Position Frame 1 of 3 Parameters:

Parameter	Direction	Description
cust_id	IN/OUT	Customer id or 0, selected by the driver. When cust_id is not 0, the rules for determining the range of available customer identifiers are described in clause 3.2.2.1.
tax_id	IN	Customer tax id or empty string selected by the driver. When tax_id is not the empty string, the rules for determining the range of available customer tax identifiers are described in clause 3.2.2.1.
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_ctry_1	OUT	Country code for customer's first phone number.
c_ctry_2	OUT	Country code for customer's second phone number.
c_ctry_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	Code indicating the execution status for this Frame.

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,
      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

}
```

10.6.2.4   **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 account 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 account. 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 **VGenTxnHarness** 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)


        if (hist_len < 10) or (hist_len > max_hist_len) then
        {
            status = -221

        }
        exit

    }
}
```

Customer-Position Frame 2 of 3 Parameters:

Parameter	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	Code indicating the execution status for this Frame.
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.

TPC Express Benchmark™ V - Specification, Revision 1.0.1 **- Page 151 of 267**

trade_status[max_hist_len]	OUT	Trade Status for each event from history.

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

```
{
   // Should return 10 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
      (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

}
```

10.6.2.5   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 **VGenTxnHarness** controls the execution of **Frame** 3 as follows:

```
    {
        if (get_history != 1)
        {
            invoke (Customer-Position_Frame-3)
        }
    }
```

Customer-Position Frame 3 of 3 Parameters:

Parameter	Direction	Description
status	OUT	Frame Status.

Customer-Position_Frame-3: End database transaction

```
{
    commit transaction

}
```

10.6.3   The Market-Feed Transaction

The Market-Feed **Transaction** is designed to emulate the process of tracking the current market activity. This is representative of the brokerage house processing the "ticker-tape" from the market exchange.

Market-Feed is invoked by **VGenDriverMEE**. It consists of a single **Frame**. The **Transaction** receives the latest trade activity information (symbol, price, quantity, etc.) from the market exchange. As a result of processing the ticker feed, the prices for securities will increase or decrease. These changes in price may trigger pending limit orders.

Each Market-Feed ticker consists of 20 entries (max_feed_len constant in TxnHarnessStructs.h). These entries are generated by the MEE to simulate the reporting of trades from other brokerage houses. The Market-Feed **Transaction** is allowed to process any number of ticker elements (from one to all) per **Database Transaction**.

10.6.3.1   Market-Feed Transaction Parameters

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

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:

Parameter	Direction	Description
price_quote[ ]	IN	A list of numeric prices the Market Exchange 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.
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 Exchange 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.
status	OUT	Code indicating the execution status for this transaction.

10.6.3.2 Market-Feed Transaction 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
Transaction Control		Start Commit

10.6.3.3 Market-Feed Transaction Frame 1 of 1

Using the entries in the ticker list, the **Frame** is responsible for:

- modifying the rows in the LAST_TRADE table with the new prices, the new daily volumes and the new last trade dates
- identifying any pending limit orders that should be triggered by these ticker prices, processing them, and submitting them to the **MEE**

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

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

```
{

invoke (Market-Feed_Frame-1)

}
```

Market-Feed Frame 1 of 1 Parameters:

Parameter	Direction	Description
price_quote[ ]	IN	A list of numeric prices the Market Exchange 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.
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 Exchange 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.
status	OUT	Code indicating the execution status for this Frame.

Market-Feed_Frame-1 Pseudo-code: Record the stock price and update the volume and datetime for securities contained in the ticker feed.

```
{
    declare now_dts DATETIME
    declare rows_updated int

    get_current_dts(now_dts)
    rows_updated = 0

        start transaction


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

        rows_updated = row_count

        commit transaction



    if (rows_updated != max_feed_len) then
    {
        status = -311
    }

}
```

10.6.4   The Market-Watch Transaction

The Market-Watch **Transaction** is designed to emulate the process of monitoring the overall performance of the market by allowing a customer to track the current daily trend (up or down) of a collection of securities. The collection of securities being monitored may be based upon a customer's current holdings, a customer's watch list of prospective securities, or a particular industry.

Market-Watch is invoked by **VGenDriverCE**. It consists of a single **Frame**. This **Transaction** calculates the percentage change in value of the market capitalization of a collection of securities at a chosen day's closing prices compared to the current market prices. The chosen day is non-uniformly selected from the 1305 days of market data that was loaded during initial population of the database. The calculation is done by looking at the chosen day's closing price for each security in the list and multiplying that by the number of outstanding shares for that security. This product is added to a running total for the chosen day's closing market capitalization. In addition, the current price for each security in the list is multiplied by the number of outstanding shares for that security. This product is added to a running sum for the current market capitalization. The difference between the total market capitalization for the chosen day's closing and the current total, expressed as a percentage, is returned.

The **Transaction** supports 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 rules for determining the range of available customers, and thereby watch lists, are described in clause 3.2.2.1.

- **Industry-Watch** - The collection of securities is chosen using all the securities in an industry belonging to companies within a specified range. 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. The rules for determining the range of available customers are described in clause 10.6.1.1. The customer account identifier is chosen at random from all the possible accounts for that customer using a uniform distribution.

10.6.4.1   Market-Watch Transaction Parameters

The inputs to the Market-Watch **Transaction** are generated by the **VGenDriverCE** code in CETxnInputGenerator.cpp. The data structures defined in TxnHarnessStructs.h must be used to communicate the input and output parameters.

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:

Parameter	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. The rules for determining the range of available customers are described in clause 3.2.2.1. 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. 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. (The rules for determining the range of possible customer identifiers are described in clause 3.2.2.1.) 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.
start_date	IN	A date non-uniformly selected from the 1305 days in the DAILY_MARKET table. The closing price of securities on this date is used in the market capitalization calculations.
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 chosen day'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.

10.6.4.2   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

10.6.4.3   Market-Watch Transaction Frame 1 of 1

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

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

```
{
    if (acct_id != 0) or (cust_id != 0) or (industry_name != "") then
    {
        invoke (Market-Watch_Frame-1)
    }
    else
    {
        status = -411
    }
}
```

Market-Watch Frame 1 of 1 Parameters:

Parameter	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. The rules for determining the range of available customers are described in clause 3.2.2.1. 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. 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. (The rules for determining the range of possible customer identifiers are described in clause 3.2.2.1.) 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.

start_date	IN	A date non-uniformly selected from the 1305 days in the DAILY_MARKET table. The closing price of securities on this date is used in the market capitalization calculations.
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 chosen day'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 of this Frame.

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
            WI_S_SYMB
         from
            WATCH_ITEM,
            WATCH_LIST
         where
            WI_WL_ID = WL_ID and
            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
```

```
            HS_CA_ID = acct_id
}
old_mkt_cap = 0.0
new_mkt_cap = 0.0
pct_change = 0.0
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

    // Closing price for this security on the chosen day.
    select
        old_price = DM_CLOSE
    from
        DAILY_MARKET
    where
        DM_S_SYMB = symbol and
        DM_DATE = start_date

    old_mkt_cap += s_num_out * old_price
    new_mkt_cap += s_num_out * new_price
}
if (old_mkt_cap != 0) then
{
    // value of 0.00 for pct_change is valid
    pct_change = 100 * (new_mkt_cap / old_mkt_cap - 1)
}
else
{
    // no rows found, this can happen rarely when an account has no holdings
```

Market-Watch_Frame-1 Pseudo-code: Build list of securities and compute percentage

```
    pct_change = 0.0


  }
  close stock_list
  commit transaction

}
```

10.6.5 The Security-Detail Transaction

The Security-Detail **Transaction** is designed to emulate the process of accessing detailed information on a particular security. This is representative of a customer doing research on a security prior to making a decision about whether or not to execute a trade.

Security-Detail is invoked by **VGenDriverCE**. It consists of a single **Frame**. For a given security, the **Transaction** will return detailed security and company information, a list of the company's competitors, current and historical financial data, and recent news items about the company.

10.6.5.1 Security-Detail Transaction Parameters

The inputs to the Security-Detail **Transaction** are generated by the **VGenDriverCE** code in CETxnInputGenerator.cpp and the data structures defined in TxnHarnessStructs.h must be used to communicate the input and output parameters.

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:

Parameter	Direction	Description
access_lob_flag	IN	If 1, access the complete news articles for the company. If 0, 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.

10.6.5.2 Security-Detail Transaction Database Footprint

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

Table	Column	Frame 1
Security-Detail Database Footprint		
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
COMPANY_COMPETITOR	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

10.6.5.3 Security Detail Transaction Frame 1 of 1

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

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

```
{

    invoke (Security-Detail_Frame-1)

    if (day_len < min_day_len) or (day_len > max_day_len) then

    {

        status = -511

    }
    else if (fin_len != max_fin_len) then

    {

        status = -512

    }
    else if (news_len != max_news_len) then

    {

        status = -513

    }

}
```

Security-Detail Frame 1 of 1 Parameters:

Parameter	Direction	Description
access_lob_flag	IN	If 1, access the complete news articles for the company. If 0, 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_ctry	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. VGen 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. VGen 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 VGen 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 VGen 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 Frame.
yield	OUT	Number containing yield for the security

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

```
{
    Declare co_id    IDENT_T
    start transaction
```

```
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,
    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 (20) rows
   select first max_fin_len 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
        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     = "",
         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

}
```

10.6.6   The Trade-Lookup Transaction

The Trade-Lookup **Transaction** is designed to emulate information retrieval by either a customer or a broker to satisfy their questions regarding a set of trades. The various sets of trades are chosen such that the work is representative of:

- performing general market analysis

- reviewing trades for a period of time prior to the most recent account statement

- analyzing past performance of a particular security

- analyzing the history of a particular customer holding

Trade-Lookup is invoked by **VGenDriverCE**. It consists of four mutually exclusive **Frames**. Each **Frame** employs a different technique for looking up historical trade data.

Frame 1 accepts a list of trade IDs. Information for each of the trades in the list is returned.

Frame 2 accepts a customer account ID, a start timestamp, end timestamp and a number of trades (N) as inputs. It returns information for the first N trades for the specified customer account between the start and end timestamps (inclusive).

Frame 3 accepts a security symbol, a start timestamp, end timestamp and a number of trades (N) as inputs. It returns information for the first N trades for the given security between the start and end timestamps (inclusive).

Frame 4 accepts a customer account ID and a timestamp as inputs. The first trade for this customer account at or after the specified timestamp is identified. Then a maximum of 20 historical holding changes for this trade ID are returned. The historical holding changes report on changes made by this trade to holdings created by prior trades, and report on changes made by subsequent trades to any holding created by this trade.

10.6.6.1   Trade-Lookup Transaction Parameters

The inputs to the Trade-Lookup **Transaction** are generated by the **VGenDriverCE** code in CETxnInputGenerator.cpp. The data structures defined in TxnHarnessStructs.h must be used to communicate the input and output parameters.

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:

Parameter	Direction	Description
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 is 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 is 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.
trade_list[ ]	OUT	List of trade IDs found in Frames 2 and 3.

10.6.6.2 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	Return	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_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

10.6.6.3 Trade-Lookup Transaction Frame 1 of 4

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

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

```
{

    if( frame_to_execute == 1 )

    {

        invoke (Trade-Lookup_Frame-1)

        if (num_found != max_trades) then

        {

            status = -611

        }

        frame_executed = 1

    }

[...]
```

Trade-Lookup Frame 1 of 4 Parameters:

Parameter	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; should be 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 = 0
```

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

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

   num_found = num_found + row_count


   // 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
```

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

_____

```
       select first 3 rows
          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

}
```

10.6.6.4   Trade-Lookup Transaction Frame 2 of 4

The second **Frame** returns information for the first N trades executed for the specified customer account between a specified start time and end time. If the specified start time is too close to the specified end time, then it is possible that fewer than N trades may be returned.

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

```
[...]
   else if( frame_to_execute == 2 )
   {
      invoke (Trade-Lookup_Frame-2)
      if (num_found < 0) or (num_found > max_trades) then
      {
         status = -621
      }
      else if (num_found == 0) then
      {
         // Can happen rarely in large databases when an account has no trades
         // in the last 4 days
         status = +621
      }
      frame_executed = 2
   }
[...]
```

Trade-Lookup Frame 2 of 4 Parameters:

Parameter	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. The rules for determining the

		range of available customers are described in clause 3.2.2.1. 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.
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_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
      where
         CT_T_ID = trade_list[i]
   }


   // read trade_history for the trades
   // Should return 2 to 3 rows per trade
   select first 3 rows
      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
```

```
   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.
```

```
}
```

10.6.6.5   Trade-Lookup Transaction Frame 3 of 4

The third **Frame** returns information for the first N trades for a given security between a specified start time and end time. If the specified start time is too close to the specified end time, then it is possible that fewer than N trades may be returned.

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

```
[...]
   else if( frame_to_execute == 3 )
   {
       invoke (Trade-Lookup_Frame-3)
       if (num_found < 0) or (num_found > max_trades) then
       {
           status = -631
       }
       else if (num_found == 0) then
       {
           // Can happen rarely in large databases
           status = +631
       }
       frame_executed = 3
   }
}
```

Trade-Lookup Frame 3 of 4 Parameters:

Parameter	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_DTS >= start_trade_dts and
      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.
```

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

```
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 first 3 rows
      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

}
```

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 **VGenTxnHarness** controls the execution of **Frame** 4 as follows:

```
[...]

    else if( frame_to_execute == 4 )

    {

        invoke (Trade-Lookup_Frame-4)

        if (num_trades_found <> 1) then

        {

            status = -641

        }


        if (num_found == 0) then

        {

            status = +643

        }


        if (num_found < 0) or (num_found > 20) then

        {

            status = -642

        }


        frame_executed = 4

    }
[...]
```

Trade-Lookup Frame 4 of 4 Parameters:

Parameter	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. The rules for determining the range of available customers are described in clause 3.2.2.1. 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.
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 (may be zero).
num_trades_found	OUT	Number of TRADE rows found.
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 account 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.

```
{
    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

    if (row_count == 0) then
    {
        status = +641
    }
    // 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 (capped) 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

}
```

10.6.7   The Trade-Order Transaction

The Trade Order **Transaction** is designed to emulate the process of buying or selling a security by a Customer, Broker, or authorized third-party. 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 at a requested price. 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 before either submitting or canceling the trade.

The Trade-Order **Transaction** is invoked by **VGenDriverCE**. It consists of six **Frames**. 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.

Next, the **Transaction** conditionally validates that the person executing the trade is authorized to perform such actions on the specified account. 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. Estimation 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). If a profit would be realized as a result of this trade, the capital gains taxes are calculated. Administrative fees and the broker's commission for handling the trade are calculated. If the trade is being submitted on margin, the customer's total assets for the account are assessed. All the above information is used for recording the order.

After all the above processing has completed, a small percentage of the Trade-Order **Transactions** are selected to emulate either the canceling the order or an error condition by rolling back all modifications. All other Trade-Order **Transactions** are **Committed**. After a successfully **Committed** market order, the **VGenTxnHarness** sends the order for the trade to the appropriate **MEE**.

10.6.7.1   Trade-Order Transaction Parameters

The inputs to the Trade-Order **Transaction** are generated by the **VGenDriverCE** code in CETxnInputGenerator.cpp. The data structures defined in TxnHarnessStructs.h must be used to communicate the input and output parameters.

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:

Parameter	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. The rules for determining the range of available customers are described in clause 3.2.2.1. 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 are an empty string (i.e. ""). Otherwise the company name and the issue are both specified and the symbol is an empty string (i.e. ""). For more information, see Clause 5.4.1.
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 5.4.1.
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 5.4.1.
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 5.4.1.
is_lifo	IN	If this flag is set to 1 then this trade will process against existing holdings from newest to oldest (LIFO order). If this flag is set to 0, then this trade will process against existing holdings from oldest to newest (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 are an empty string (i.e. ""). Otherwise the company name and the issue are both specified and the symbol is an empty string (i.e. ""). For more information, see Clause 5.4.1.
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 is set to the current market price for the given security inside Frame 3.
roll_it_back	IN	If this flag is 1 then an intentional rollback (Frame 5) is executed. If 0, then a commit (Frame 6) is executed. See Clause 5.4.1 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 are an empty string (i.e. ""). Otherwise the company name and the issue are both specified and the symbol is an empty string (i.e. ""). For more information, see Clause 5.4.1.
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 5.4.1.

type_is_margin	IN	If this flag is set to 1, then the order will be done on margin. If the flag is set to 0, 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.
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.

10.6.7.2   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_B_ID	Return					
	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

10.6.7.3 Trade-Order Transaction Frame 1 of 6

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

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

```
{
    invoke (Trade-Order_Frame-1)
    if (num_found <> 1) then
    {
        status = -711
    }


}
```

Trade-Order Frame 1 of 6 Parameters:

Parameter	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_id	OUT	Identifier of the broker associated with the specified 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.
num_found	OUT	Number of CUSTOMER_ACCOUNT rows found.
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

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

```
{
   start transation

   // Get account, customer, and broker information
   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

   if (row_count == 0) then
   {
      status = -711
   }

   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

}
```

10.6.7.4   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[0] == '\0') then

        {

            status = -721;

        }

    }

}
```

Trade-Order Frame 2 of 6 Parameters:

Parameter	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
        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
        status = -721
    }

}
```

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 **VGenTxnHarness** controls the execution of **Frame** 3 as follows:

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

    if ((sell_value > buy_value) and

        ((tax_status == 1) or (tax_status == 2)) and

        (tax_amount == 0.00)) then

    {

        status = -731

    }
    else if (comm_rate <= 0.0000) then

    {

        status = -732

    }
    else if (charge_amount <= 0.00) then

    {

        status = -733

    }


}
```

Trade-Order Frame 3 of 6 Parameters:

Parameter	Direction	Description
acct_id	IN	Identifier of the customer account involved in the transaction.
cust_id	IN	Unique identifier of the customer who owns the specified account.
cust_tier	IN	The brokerage house service level tier this customer belongs to.
is_lifo	IN	If this flag is set to 1 then this trade will process against existing holdings from newest to oldest (LIFO order). If this flag is set to 0, then this trade will process against existing holdings from oldest to newest (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 1, then the order will be done on margin. If the flag is set to 0, 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 1 for market orders and to 0 for limit orders.
type_is_sell	OUT	Flag set to 1 for sell orders and to 0 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,
         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 the frame,
  // but if this a market-order, then the requested_price needs to be set 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
// 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  = acct_id and
   HS_S_SYMB = symbol

if (hs_qty is NULL) then     // No prior holdings exist — no rows returned
   hs_qty = 0

if (type_is_sell) then {
   // This is a sell transaction, so estimate the impact to any currently held
   // long postions 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
        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 at different prices for this security (representing
    // multiple purchases different times).
    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 creating a new short position for the remaining balance of
// this 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 postions will be covered before opening a long postion 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
                 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 postions currently held for this security. The customer
           // may have multiple holdings at different prices for this security
           // (representing multiple purchases at different times).
           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
            }
        }
        close hold_list
    }
    // NOTE: If needed_qty is still greater than 0 at this point, then the
    // customer would cover all current short positions (if any) for this security,
    // 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 {
    //
    // Customers may be subject to more than one tax at different rates.
    // Therefore, get the sum of the tax rates that apply to the customer
    // and estimate the 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, commision 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
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
    }

}
```

10.6.7.6   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 commision 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:

Parameter	Direction	Description
acct_id	IN	Identifier of the customer account involved in the transaction.
broker_id	IN	Identifier of the broker associated with 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 1, then this trade will be done with cash. If this flag is set to 0, then this trade will be done on margin.
is_lifo	IN	If this flag is set to 1 then this trade will process against existing holdings from newest to oldest (LIFO order). If this flag is set to 0, then this trade will process against existing holdings from oldest to newest (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 1 for market orders and to 0 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,
      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_B_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
         broker_id         // TR_B_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
 )

}
```

10.6.7.7 Trade-Order Transaction Frame 5 of 6

The fifth **Frame** is conditionally executed when the parameter roll_it_back is set to 1. 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 **VGenTxnHarness** controls the execution of **Frame** 5 as follows:

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

Trade-Order Frame 5 of 6 Parameters:

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

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

}
```

10.6.7.8   Trade-Order Transaction Frame 6 of 6

The sixth **Frame** is conditionally executed when parameter roll_it_back is set to 0. 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 **VGenTxnHarness** 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)
    SendToMarketFromHarness (

        requested_price,

        symbol,

        trade_id,

        trade_qty,

        trade_type_id,

        eAction

    )

}
```

Trade-Order Frame 6 of 6 Parameters:

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

Trade–Order Frame 6 Pseudo–code: Commit database transaction

```
{

    commit transaction

}
```

10.6.8   The Trade-Result Transaction

The Trade-Result **Transaction** is designed to emulate the process of completing a stock market trade. This is representative of a brokerage house receiving from the market exchange the final confirmation and price for the trade. The customer's holdings are updated to reflect that the trade has completed. Estimates generated when the trade was ordered for the broker commission and other similar quantities are replaced with the actual numbers and historical information about the trade is recorded for later reference.

Trade-Result is invoked by **VGenDriverMEE**. It consists of seven **Frames**. The **Transaction** starts by using the trade ID passed into the **Transaction** to obtain information about the trade. The information gathered includes the account ID of the customer account, which is used to lookup additional account information.

Next the customer's holdings are updated to reflect the completion of the trade. The particular work done depends on the type of trade (buy or sell), the number of shares involved and the customer's current position (long or short) with respect to the security. When selling shares, current holdings are liquidated to cover the sale. If the customer does not have enough shares to cover the sale, any currently held shares are liquidated and a short position is taken for the balance of shares. If the customer already has a short position and more shares are sold, then the short position is simply extended. An analogous situation exists when purchasing shares. Any shares bought will first be used to cover any existing short position. After that, any shares bought will be used to create or extend a long position.

If, when reconciling the trade with the customer's current holdings, any shares are sold for a profit and the profit is taxable, the amount of tax due on the profit is calculated.

Next the broker's commission is calculated and then all information with respect to the trade is recorded.

Finally, settlement records are entered for the trade and if the trade is not on margin, the customer's account balance is update accordingly.

The seventh frame is independent of the prior six and is a separate database transaction. It is invoked only when the separate transaction "trigger_id" input parameter is non-zero. When that condition occurs, the seventh frame performs the actions required to submit the previously pending limit order that has now reached its trigger (bid or ask) price.

10.6.8.1   Trade-Result Transaction Parameters

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

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
Frame 7 Input/Output Structure	TTradeResultFrame7Input TTradeResultFrame7Output

Trade-Result Transaction Parameters:

Parameter	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.
trigger_id	IN	The Trade ID for the pending trade that has triggered and needs to be to be submitted to the MEE. Trade ID is the primary key of the TRADE table.
acct_bal	OUT	Customer account's cash balance after the trade was completed.
acct_id	OUT	Customer account ID of the customer account involved in Trade-Result transaction.
load_unit	OUT	Load Unit number for the customer account involved in the Trade-Result transaction.
status	OUT	Code indicating the execution status for this transaction.

10.6.8.2 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**	**7**
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*					

TPC Express Benchmark™ V - Specification, Revision 1.0.1 - Page 201 of 267

	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_DTS					Modify		Modify*
	T_IS_CASH	Return						
	T_LIFO	Return						
	T_QTY	Return						
	T_S_SYMB	Return						
	T_ST_ID					Modify		Modify*
	T_TAX			Modify				
	T_TRADE_PRICE					Modify		
	T_TT_ID	Return						
TRADE_HISTORY	1 row					Add		Add *
TRADE_REQUEST	TR_BID_PRICE							Return*
	TR_QTY							Return*
	TR_T_ID							Return*
	TR_TT_ID							Return*
	Row(s)							Remove*
TRADE_TYPE	TT_IS_MRKT	Return						
	TT_IS_SELL	Return						
	TT_NAME	Return						
Transaction Control		Start					Commit	Start, Commit

10.6.8.3 Trade-Result Transaction Frame 1 of 7

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 **VGenTxnHarness** controls the execution of **Frame** 1 as follows:

```
{

    invoke (Trade-Result_Frame-1)

    if (num_found <> 1) then

    {

        status = -811

    }


}
```

Trade-Result Frame 1 of 7 Parameters:

Parameter	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 flag is set to 1, then this trade will process against existing holdings from newest to oldest (LIFO order). If this flag is set to 0, then this trade will process against existing holdings from oldest to newest (FIFO order).
num_found	OUT	Number of TRADE rows found.
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 (1) or on margin (0).
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 trade (1) or limit trade (0).
type_is_sell	OUT	Boolean indicating if this is a sell trade (1) or a buy trade (0).
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

num_found = row_count

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

}
```

10.6.8.4   Trade-Result Transaction Frame 2 of 7

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 **VGenTxnHarness** controls the execution of **Frame** 2 as follows:

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

}
```

Trade-Result Frame 2 of 7 Parameters:

Parameter	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 flag is set to 1, then this trade will process against existing holdings from newest to oldest (LIFO order). If this flag is set to 0, then this trade will process against holdings from oldest to newest (FIFO order).
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 (1) or a buy trade (0).
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
```

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

```
// 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,
            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.
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
                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_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
             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
       }
```

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

```
// 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
         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,
                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
   }

}
```

10.6.8.5   Trade-Result Transaction Frame 3 of 7

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 **VGenTxnHarness** 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 **VGenTxnHarness** controls the execution of **Frame** 3 as follows:

```
{
   tax_amount = 0.00

   if ((tax_status == 1 or tax_status == 2)

      and (sell_value > buy_value)) then

   {

      invoke (Trade-Result_Frame-3)

      if (tax_amount <= 0.00) then

      {

         status = -831

      }

   }

}
```

Trade-Result Frame 3 of 7 Parameters:

Parameter	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 VGen 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
    T_ID = trade_id

}
```

10.6.8.6   Trade-Result Transaction Frame 4 of 7

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 **VGenTxnHarness** controls the execution of **Frame** 4 as follows:

```
{
    invoke (Trade-Result_Frame-4)
    if (comm_rate <= 0.00) then
    {
        status = -841
    }
}
```

Trade-Result Frame 4 of 7 Parameters:

Parameter	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 Pseudo-code: 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
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

}
```

10.6.8.7   Trade-Result Transaction Frame 5 of 7

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 **VGenTxnHarness** 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 7 Parameters:

Parameter	Direction	Description
broker_id	IN	Broker ID, which was obtained in Frame 1.
comm_amount	IN	The broker commission amount, computed by the VGen 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.

```
{
  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

}
```

10.6.8.8   Trade-Result Transaction Frame 6 of 7

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

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

The **VGenTxnHarness** 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 7 Parameters:

Parameter	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 (1) or on margin (0).
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

}
```

10.6.8.9 Trade-Result Transaction Frame 7 of 7

The seventh **Frame** is responsible for submitting a pending limit order that has been triggered. It is therefore independent of the prior six frames and performs as a separate database transaction.

The database access methods used in **Frame** 7 are a mixture **Add, Modify, Remove and Return**.

The **VGenTxnHarness** controls the execution of **Frame** 7 as follows:

```
{
    if (trigger_id != 0) then
    {
        invoke (Trade-Result_Frame-7)

        eAction = eMEEProcessOrder

        // Send the trade to the Market Exchange Emulator (MEE)
        SendToMarketFromHarness (
            bid_price,
            symbol,
            trade_id,
            trade_qty,
            trade_type_id,
            eAction
    }
}
```

Parameter	Direction	Description
status_submitted	IN	The string ID value for the STATUS_TYPE Submitted status.
trigger_id	IN	The Trade ID for the pending trade that has triggered and needs to be to be submitted to the MEE. Trade ID is the primary key of the TRADE table.
bid_price	OUT	Requested bid/ask price for triggered limit trade.
num_found	OUT	Number of TRADE rows found to trigger.
status	OUT	Code indicating the execution status for this frame.
symbol	OUT	Security symbol for triggered limit trade.
trade_id	OUT	Trade ID of triggered limit trade.
trade_qty	OUT	Requested share quantity.
trade_type_id	OUT	Trade type of triggered limit trade.

Trade-Result_Frame-7 Pseudo-code: Submit the triggered limit trade

```
{
    declare now_dts DATETIME

    start transaction

    get_current_dts(now_dts)

    select    TR_T_ID,
              TR_BID_PRICE,
              TR_S_SYMB,
              TR_TT_ID,
              TR_QTY
    from
              TRADE_REQUEST
    where
              TR_T_ID = trigger_id



    num_found = row_count

    delete    TRADE_REQUEST
    where     TR_T_ID = trigger_id

    update    TRADE
    set       T_DTS   = now_dts,
              T_ST_ID = status_submitted
    where     T_ID    = trigger_id

    insert    TRADE_HISTORY (TH_T_ID, TH_DTS, TH_ST_ID)
                    values (trigger_id, now_dts, status_submitted)

    commit transaction
```

10.6.9   The Trade-Status Transaction

The Trade-Status **Transaction** is designed to emulate the process of providing an update on the status of a particular set of trades. It is representative of a customer reviewing a summary of the recent trading activity for one of their accounts.

Trade-Status is invoked by **VGenDriverCE**. It consists of a single **Frame**. For the given account ID, Trade-Status returns the trade ID and status of the 50 most recent trades.

10.6.9.1   Trade-Status Transaction Parameters

The inputs to the Trade-Status **Transaction** are generated by the **VGenDriverCE** code in CETxnInputGenerator.cpp and the data structures defined in TxnHarnessStructs.h must be used to communicate the input and output parameters.

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:

Parameter	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. The rules for determining the range of available customers are described in clause 3.2.2.1. 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.
status	OUT	Code indicating the execution status for this transaction.
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.

10.6.9.2 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

10.6.9.3 Trade-Status Transaction Frame 1 of 1

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

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

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

    if (num_found <> max_trade_status_len) then
    {
        status = -911
    }


}
```

Trade-Status Frame 1 of 1 Parameters:

Parameter	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. The rules for determining the range of available customers are described in clause 3.2.2.1. 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).
num_found	OUT	Number of TRADE rows found.
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 frame.
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.

```
{
    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
    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

    num_found = row_count

    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
```

```
}
```

10.6.10 The Trade-Update Transaction

The Trade-Update **Transaction** is designed to emulate the process of making minor corrections or updates to a set of trades. This is analogous to a customer or broker reviewing a set of trades, and discovering that some minor editorial corrections are required. The various sets of trades are chosen such that the work is representative of:

- reviewing general market trends
- reviewing trades for a period of time prior to the most recent account statement
- reviewing past performance of a particular security

Trade-Update is invoked by **VGenDriverCE**. It consists of three mutually exclusive **Frames**. Each **Frame** employs a different technique for looking up historical trade data. Minor corrections are made to the retrieved data.

Frame 1 accepts a list of trade IDs. Information for each of the trades in the list is returned. The executor's name for each trade is modified.

Frame 2 accepts a customer account ID, a start timestamp, end timestamp and a number of trades (N) as inputs. The **Frame** returns information for the first N trades for the specified customer account between the start and end timestamps (inclusive). The settlement cash type for each trade is modified.

Frame 3 accepts a security symbol, a start timestamp, end timestamp and a number of trades (N) as inputs. The **Frame** returns information for the first N trades for the given security between the start and end timestamps (inclusive). For cash trades the description of the **Transaction** is modified.

10.6.10.1 Trade-Update Transaction Parameters

The inputs to the Trade-Update **Transaction** are generated by the **VGenDriverCE** code in CETxnInputGenerator.cpp. The data structures defined in TxnHarnessStructs.h must be used to communicate the input and output parameters.

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:

Parameter	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.
trade_list[ ]	OUT	List of trade IDs found in Frames 2 and 3.

10.6.10.2 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

TPC Express Benchmark™ V - Specification, Revision 1.0.1 - Page 226 of 267

	T_QTY			Return
	T_S_SYMB			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

10.6.10.3 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 **VGenTxnHarness** controls the execution of **Frame** 1 as follows:

```
{
    if( frame_to_execute == 1 )
    {
        invoke (Trade-Update_Frame-1)
        if (num_found != max_trades) then
        {
            status = -1011
        }
        if (num_updated != max_updates) then
        {
            status = -1012
        }
        frame_executed = 1
    }
[...]
```

Trade-Update Frame 1 of 3 Parameters:

Parameter	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. Must be <= max_trades.
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; should be the same as max_trades.
num_updated	OUT	Number of TRADE rows that were modified; should be the same as max_updates.
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(49)
   start transaction

   num_found = 0
   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]

           num_found = num_found + row_count

           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
```

```
        CT_T_ID = trade_id[i]

    }
    // read trade_history for the trades
    // Will return 2 or 3 rows per trade
    select first 3 rows
        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

}
```

10.6.10.4 Trade-Update Transaction Frame 2 of 3

The second **Frame** returns information for the first N trades executed for the specified customer account between a specified start time and end time and modifies the SETTLEMENT row for each trade returned. If the specified start time is too close to the specified end time, then it is possible that fewer than N trades may be returned and SETTLEMENT rows modified.

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

```
[...]

    else if( frame_to_execute == 2 )

    {

        invoke (Trade-Update_Frame-2)

        if (num_updated != num_found) then

        {

            status = -1021

        }

    if (num_updated < 0) then

        {

            status = -1022

        }

    if (num_found > max_trades) then

        {

            status = -1022

        }

        if (num_updated == 0) then

        {

            status = +1021

        }

        frame_executed = 2

    }

[...]
```

Trade-Update Frame 2 of 3 Parameters:

Parameter	Direction	Description
acct_id	IN	A single customer is chosen non-uniformly by customer tier, from the range of available customers. The rules for determining the range of available customers are described in clause 3.2.2.1. 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.
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
        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
    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 first 3 rows
        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

}
```

10.6.10.5   Trade-Update Transaction Frame 3 of 3

The third **Frame** returns information for the first N trades for a given security between a specified start time and end time and modifies the related CASH_TRANSACTION row for each trade returned. If the specified start time is too close to the specified end time, then it is possible that fewer than N trades may be returned and CASH_TRANSACTION rows modified.

.

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

```
[...]

   else if( frame_to_execute == 3 )

   {

       invoke (Trade-Update_Frame-3)

       if (num_found == 0) then

       {

           status = +1031

       }

       if (num_found > max_trades) then

       {

           status = +1032

       }

       frame_executed = 3

   }

}
```

Trade-Update Frame 3 of 3 Parameters:

Parameter	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_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
```

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.

```
// 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
    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
```

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
            CT_T_ID = trade_list[i]

    }


    // read trade_history for the trades
    // Will return 2 or 3 rows per trade
    select first 3 rows
        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


}
```

10.6.11   The Data-Maintenance Transaction

The Data-Maintenance **Transaction** is designed to emulate the periodic modifications to data that is mainly static and used for reference. This is analogous to updating

Data-Maintenance is invoked by **VGenDriverDM**. It consists of one **Frame**. 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** alters the next table in the list. This means that each table in the list will only get altered once every twelve minutes.

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

The Data-Maintenance **Transaction** consists of a single **Frame**.

The intent of the **Transaction** is to alter data tables that would not otherwise be written to by the benchmark. The **VGenTxnHarness** will pick the next table in the list to alter, each time this **Transaction** is run.

Below is a description of what kind of alteration is done to each table when that table is selected:

1. ACCOUNT_PERMISSION - The **VGenTxnHarness** 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 (The **Sponsor** may decide which row is first). 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 **VGenTxnHarness** will pass a customer identifier to the Data-Maintenance **Transaction**. The other 33% of the time **VGenTxnHarness** 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 **VGenTxnHarness** 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 **VGenTxnHarness** 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 **VGenTxnHarness** will pass a customer identifier to the Data-Maintenance **Transaction**. The country tax rate will be modified cyclically to the next rate in the set {"US1", "US2", "US3", "US4", "US5"} or in the set {"CN1", "CN2", "CN3", "CN4"}, depending on the customer's country.

6. DAILY_MARKET – The **VGenTxnHarness** will pass a security symbol, a day of the month, and a random number (positive or negative) to the Data-Maintenance **Transaction**. All rows in DAILY_MARKET with matching symbol and day of the month will be updated by adding the random number to DM_VOL.

7. EXCHANGE – The **VGenTxnHarness** 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 **VGenTxnHarness** 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 **VGenTxnHarness** 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 **VGenTxnHarness** 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 so that a substring will be toggled between "Tax" and "tax".

12. WATCH_ITEM – The **EGenTxnHarness** will pass in a customer identifier to the Data-Maintenance **Transaction**. The middle security in the customer's WATCH_ITEM list will be selected. It will be modified to be the next symbol in the SECURITY table that is not already in the customer's WATCH_ITEM list.

10.6.11.1   Transaction Parameters

The inputs to the Data-Maintenance **Transaction** are generated by the **VGenDriverDM** in DM.cpp. The data structures defined in TxnHarnessStructs.h must be used to communicate the input and output parameters.

Data-Maintenance Interfaces	Module/Data Structure
Input generation	GenerateDataMaintenanceInput()
Transaction Input/Output Structure	TDataMaintenanceTxnInput TDataMaintenanceTxnOutput
Frame 1 Input/Output Structure	TDataMaintenanceFrame1Input TDataMaintenanceFrame1Output

Data-Maintenance Transaction Parameters:

Parameter	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. 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", or "CUSTOMER_TAXRATE". 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 to 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 altered. Valid values are "ACCOUNT_PERMISSION", "ADDRESS", "COMPANY", "CUSTOMER", "CUSTOMER_TAXRATE", "DAILY_MARKET", "EXCHANGE", "FINANCIAL", "NEWS_ITEM", "SECURITY". This input is always used.
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	Code indicating the execution status of this transaction.

10.6.11.2 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_RATE	Reference * Modify (1 row)*
CUSTOMER	C_AD_ID	Reference *
	C_EMAIL_2	Reference * Modify (1 row)*
	C_ID	Reference *
CUSTOMER_TAXRATE	CX_C_ID	Reference *
	CX_TX_ID	Reference* Modify (1 row)*
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 *
TAXRATE	TX_ID	Reference *
	TX_NAME	Reference * Modify *
WATCH_ITEM	WI_S_SYMB	Reference * Modify *
	WI_WL_ID	Reference *
Transaction Control		Start Commit

10.6.11.3 Data-Maintenance Transaction Frame 1 of 1

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

```
{
    invoke (Data-Maintenance_Frame-1)
}
```

Data-Maintenance Frame 1 of 1 Parameters:

Parameter	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. 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", or "CUSTOMER_TAXRATE". 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 altered. Valid values are "ACCOUNT_PERMISSION", "ADDRESS", "COMPANY", "CUSTOMER", "CUSTOMER_TAXRATE", "DAILY_MARKET", "EXCHANGE", "FINANCIAL", "SECURITY". This input is always used.
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	Code indicating the execution status of this Frame.

Data-Maintenance Frame 1 Pseudo-code: 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
    order by
        AP_ACL DESC

    if (acl != "1111") then {
        update
            ACCOUNT_PERMISSION
        set
            AP_ACL="1111"
        where
            AP_CA_ID = acct_id and
            AP_ACL = acl
    } else { /*ACL is "1111" change it to "0011" */
        update
            ACCOUNT_PERMISSION
        set
            AP_ACL = "0011"
        where
            AP_CA_ID = acct_id and
            AP_ACL = acl
    }
} 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".
       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
             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
      // Find the customer's current country tax rate code.
      // Calculate cyclically the next tax rate code for the customer's country.
```

```
    // Update to the new country tax rate code.
    declare old_tax_rate   char(3),
            new_tax_rate   char(3),
            tax_num        int


    select
        old_tax_rate = CX_TX_ID
    from
        CUSTOMER_TAXRATE
    where
        CX_C_ID = c_id and
        (CX_TX_ID like "US%" or CX_TX_ID like "CN%")


    if (left(old_tax_rate,2) = "US") {
        if (old_tax_rate = "US5") {
            new_tax_rate = "US1"
        }
        else {      // Change string US<n> to US<n+1> for n=1, 2, 3, 4
            tax_num = CODE(right(old_tax_rate,1)) – CODE("0") + 1
            new_tax_rate = "US" + CHAR(tax_num + CODE("0"))
        }
    }
    else {
        if (old_tax_rate = "CN4") {
            new_tax_rate = "CN1"
        }
        else {    // Change string CN<n> to CN<n+1> for n=1, 2, 3
            tax_num = CODE(right(old_tax_rate,1)) – CODE("0") + 1
            new_tax_rate = "CN" + CHAR(tax_num + CODE("0"))
        }
    }


    update
        CUSTOMER_TAXRATE
    set
        CX_TX_ID = new_tax_rate
    where
        CX_C_ID = c_id and
        CX_TX_ID = old_tax_rate

} 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

   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
        }
} 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
                 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
  }
  commit transaction

}
```

10.6.12 The Trade-Cleanup Transaction

The Trade-Cleanup **Transaction** is used to cancel any pending or submitted trades from the database. The **Sponsor** may use **VGenTxnHarness** 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**.

10.6.12.1 Trade-Cleanup Transaction Parameters

The inputs to the Trade-Cleanup **Transaction** are supplied by the **Sponsor**. The data structures defined in TxnHarnessStructs.h must be used to communicate the input and output parameters.

Trade-Cleanup Interfaces	Module/Data Structure
Transaction Input/Output Structure	TTradesCleanupTxnInput TTradesCleanupTxnOutput
Frame 1 Input/Output Structure	TTradesCleanupFrame1Input TTradesCleanupFrame1Output

Trade-Cleanup Transaction Parameters:

Parameter	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.

10.6.12.2   Trade-Cleanup Transaction Database Footprint

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

Table	Column	Frame 1
Trade-Cleanup Database Footprint		
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

10.6.12.3   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 **VGenTxnHarness** 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:

Parameter	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 frame.

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 )

  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) {
  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 (
```

```
Trade-Cleanup_Frame-1 Pseudo-code: cancel pending and submitted trades

        t_id,            // TH_T_ID
        now_dts,         // TH_DTS
        st_canceled_id   // TH_ST_ID
    )


  } //end of submit_list


  commit transaction

}
```

10.7   VGen

10.7.1   Overview

VGen is one of the modules of the **Benchmark Kit**, and is a TPC provided software package designed to facilitate the implementation of TPCx-V. **VGen** provides:

- consistent data generation independent of the underlying environment
- **Transaction** generation and **Frame** flow control management
- project build and makefile templates

This clause covers the constraints and regulations governing the use of **VGen**. For detailed information on **VGen**, what features and functionality it provides and how the TPCx-V **Benchmark Kit** uses those features and functionality see Clause 10 -- .

10.7.2   VGen Terms

10.7.2.1   **VGen** is a TPC provided software environment that is used in the TPC provided **Benchmark Kit** implementation of the TPCx-V benchmark. The software environment is logically divided into three packages: **VGenProjectFiles**, **VGenInputFiles**, and **VGenSourceFiles**. The software packages provide functionality to use: **VGenLoader** to generate the data used to populate the database, **VGenDriver** to generate transactional data and **VGenTxnHarness** to control frame invocation.

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

10.7.2.3   **VGenInputFiles** is a set of TPC provided text files containing rows of tab-separated data, which are used by various **VGen** packages as "raw" material for data generation.

10.7.2.4   **VGenSourceFiles** is the collection of TPC provided C++ source and header files.

10.7.2.5 **VGenLoader** is a binary executable, generated by using the methods described in **VGenProjectFiles** with source code from **VGenSourceFiles**, including any extensions by a **Test Sponsor** (see Clause 10.7.6.4). When executed, **VGenLoader** uses **VGenInputFiles** to produce a set of data that represents the initial state of the TPCx-V database.

VGenDriver comprises the following parts:

- **VGenDriverCE** provides the core functionality necessary to implement a **Customer Emulator**.
- **VGenDriverMEE** provides the core functionality necessary to implement a **Market Exchange Emulator**.
- **VGenDriverDM** provides the core functionality necessary to implement the **Data-Maintenance Generator**.

10.7.2.6 **VGenDriver** provides core transactional functionality (e.g. **Transaction Mix** and input generation) necessary to implement a **Driver**. **VGenTxnHarness** defines a set of interfaces that are used to control the execution of, and communication of inputs and outputs, of **Transactions** and **Frames**.

10.7.2.7 **VGenValidate** is a binary executable, generated by using methods described in **VGenProjectFiles** with source code from **VGenSourceFiles**. The **Benchmark Kit** uses **VGenValidate** to validate that the **Sponsor's Measurement Interval** had compliant Trade-Results per **Load Unit**.

10.7.2.8 **VGenLogger** logs the initial configuration and any re-configuration of **VGenDriver** and **VGenLoader**, and compares current configuration with the TPCx-V prescribed defaults.

10.7.3 Compliant VGen Versions

10.7.3.1 The TPC Policies Clause 5.3.1 requires that the version of the specification and VGen must match. The **VGen** version can be determined by calling the GetVGenVersion function provided in VGen/src/VGenVersion.cpp file.

10.7.3.2 **VGen** is intended to produce correct data. The TPCx-V **Benchmark Kit** ensures that the random distribution of all data values, inputs and **Transaction Mix** frequencies produced by **VGen** is compliant with all constraints documented in the specification (e.g. **Transaction Mix**, execution rules, population constraints, etc.).

10.7.3.3 Any existing errors in a compliant version of **VGen**, 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.

10.7.3.4 **VGen** 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 **VGen** because of the compiler's non-conformance with the above standards does not constitute a bug or error in **VGen**.

10.7.4 VGenInputFiles

Modification of **VGenInputFiles** provided by the TPC is not permitted.

10.7.5 VGenSourceFiles

Modification of **VGenSourceFiles** provided by the TPC is not allowed, except as permitted by clause 10.7.3.

10.7.6   VGenLoader

10.7.6.1   The data for a compliant TPCx-V database must be generated by **VGenLoader**. The version of **VGenLoader** used must be compliant with the version of the specification the **Result** is being published under, as listed in clause 10.7.3.

10.7.6.2   It is presumed that **VGenLoader** produces the correct number of rows for each TPCx-V table. However due to the random nature of the data generated by **VGenLoader**, the data may not be compliant with Clause 2 of this specification. In that event the test database is considered invalid.

10.7.6.3   If **VGenLoader** generates an empty string, an empty string should be loaded in the database.

10.7.6.4   If the **Test Sponsor** extends the loading interface of **VGenLoader** (as described in Appendix 10.7.10.6.6), 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 **VGenLoader** 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 **VGenLoader** to specify how the data gets loaded into the database.

10.7.7   VGenDriver

10.7.7.1   All **VGenLogger** output must be **reported** in the **Supporting Files**. If any **VGenLogger** output contains "NO", indicating the correct default values were not used, the benchmark **Result** is not compliant.

10.7.7.2   **Sponsors** must use a constructor for each object class (CCE, CMEE, or CDM) that does not have RNGSEED parameter(s).

10.7.7.3   **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.

10.7.7.4   The **Transaction** inputs are generated by the **VGenDriverCE**, **VGenDriverMEE** and **VGenDriverDM** 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 **VGenLoader** instances during the initial data generation.

The contents of **VGenInputFiles** used by all **VGenLoader** instances (when building the database) and by all **CE**, **MEE** and **DM** instances (when running against the database) must be the **VGenInputFiles** for the version of TPCx-V that is used in the benchmark publication.

10.7.7.5   VGenDriverCE

10.7.7.5.1 A compliant **CE** implementation must use **VGenDriverCE**.

10.7.7.6   VGenDriverMEE

10.7.7.6.1 A compliant **MEE** implementation must use **VGenDriverMEE**.

10.7.7.7   VGenDriverDM

10.7.7.7.1 A compliant **Data-Maintenance Generator** must use **VGenDriverDM**.

10.7.7.7.2 One, and only one, instance of the **Data-Maintenance Generator** is required and allowed during a **Test Run**.

10.7.8   VGenTxnHarness

10.7.8.1   A compliant TPCx-V implementation must use **VGenTxnHarness**.

10.7.9   VGenValidate

10.7.9.1   A compliant TPCx-V implementation must use **VGenValidate** to verify a **Measurement Interval** has compliant Trade-Results per **Load Unit**.

10.7.10   VGen User's Guide

10.7.10.1   Overview

VGen is a TPC provided software package. It is designed to facilitate the implementation of TPCx-V. This appendix provides information on how a **Test Sponsor** is to use the features and functionality of **VGen**. The definitions, descriptions, constraints and regulations governing the use of **VGen** are captured in Clause 1.5.

Comment: Some of the following sections assume the reader has a good understanding of object-oriented design and programming techniques using ANSI C++.

10.7.10.2   VGen Directory

10.7.10.2.1 **VGen** is distributed in a single directory hierarchy. The following diagram shows the overall **VGen** directory hierarchy.
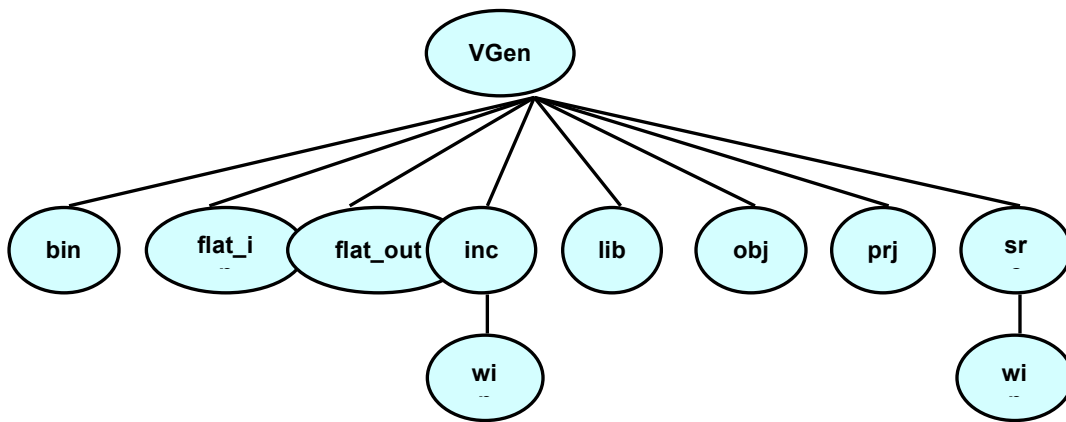
Figure A.a - Hierarchy of VGen Directory

- 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

10.7.10.3   VGenProjectFiles

10.7.10.3.1 **VGenProjectFiles** are located in the VGen/prj directory. These files can be used to facilitate building **VGen** components in various environments.

- Windows

    A set of Visual Studio 2003 files are provided. VGen.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 **VGen** components using a make utility. The makefile is known to work with GNU make, but other flavors of make may require some editing of the makefile.

10.7.10.4   VGenInputFiles

10.7.10.4.1 **VGenInputFiles** are located in the VGen/flat_in directory. These files are text files containing rows of tab-separated data. The files are used by **VGenLoader** to create the data to populate the database and by **VGenDriver** components to generate valid input for **Transactions**. The generated data is based on knowing the contents of the input files ("raw" material) and the overall scaling factors (**Scale Factor**, **Configured Customers**, **Initial Trade Days**).

10.7.10.5   VGenSourceFiles

10.7.10.5.1 **VGenSourceFiles** are located in VGen/inc, VGen/src and their associated sub-directories.

10.7.10.5.2 **VGenSourceFiles** contain TPC-provided ANSI C++ code to be used in a compliant TPCx-V implementation. Functionality is provided to facilitate:

- population of a TPCx-V compliant database
- implementation of a TPCx-V compliant environment

This functionality is described in subsequent sections.

10.7.10.6   VGenLoader

10.7.10.6.1 The task of populating a compliant TPCx-V database can be broken into two parts:

- generating compliant data records
- loading the records into the database

Comment: The **Sponsor** is responsible for coming up with scripts to create the database and tables and to apply the required constraints.

10.7.10.6.2 Data generation is a **DBMS**-neutral task, whereas database population is obviously very **DBMS**-specific. Therefore, **VGenLoader** is architected honoring this separation as follows. **VGenSourceFiles** contain class definitions that provide abstractions of the TPCx-V tables. These table classes are known collectively as **VGenTables** and they encapsulate the functionality needed to generate the data for each of the TPCx-V tables. Many of the classes in **VGenTables** are dependent on **VGenInputFiles** for "raw" material used in data record generation. **VGenLoader** therefore makes **VGenInputFiles** available to **VGenTables**, and uses **VGenTables** to generate TPCx-V compliant data records.

10.7.10.6.3 In order to support the **DBMS**-specific nature of loading the generated data, **VGenLoader** makes use of a virtual base class CBaseLoader to "load" the data. This provides a controlled interface from the **DBMS**-neutral data generation portion of **VGenLoader** to the **DBMS**-specific data loading portion of **VGenLoader**. **DBMS**-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 TPCx-V table being loaded.) **VGenLoader** provides two alternative implementations of CBaseLoader.

10.7.10.6.4 The first loader functionality provided by **VGenLoader** doesn't actually load a database directly, but rather produces output flat files. One text file is produced for each TPCx-V table. These files contain rows of data values, where the data values are separated by "|". To use this functionality, define the compile-time variable COMPILE_FLAT_FILE_LOAD when building **VGenLoader** and use the "-l FLAT" switch when running **VGenLoader**.

This mode of loader functionality is designed to work with bulk-loader tools which populate a database with the contents of a set of flat files. Due to variations in the expected format of certain data types, it is possible to configure **VGenLoader** via compile-time variables to change the format of certain data types in the output flat files. The data types, compile-time variables and possible values are listed in the following table:

Data Type	Compile-Time #define	Possible Values
DATETIME	DATETIME_FORMAT	See CDateTime::ToStr() in src/DateTime.cpp
DATE	DATE_FORMAT	See CDateTime::ToStr() in src/DateTime.cpp
TIME	TIME_FORMAT	See CDateTime::ToStr() in src/DateTime.cpp
BOOLEAN	BOOLEAN_TRUE	Any string constant representing a TRUE Boolean value. String constants must be quoted.
BOOLEAN	BOOLEAN_FALSE	Any string constant representing a FALSE Boolean value. String constants must be quoted.

10.7.10.6.5 The second loader functionality provided by **VGenLoader** 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 **VGenLoader** and use the "-l ODBC" switch when running **VGenLoader**.

10.7.10.6.6 **VGenLoader** 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 **VGenLoader**, and use the "-p" option to pass parameters to the custom loader.

10.7.10.6.7 A full listing of **VGenLoader** switches can be seen by building **VGenLoader** using **VGenProjectFiles** and then running **VGenLoader** with the "-?" switch.

10.7.10.7   VGenDriver

10.7.10.7.1 A TPCx-V **Test Sponsor** is responsible for implementing a compliant TPCx-V **Driver** (Clause 4 --). The TPC provides **VGenDriver** to facilitate implementation of a compliant **Driver** and to standardize certain key platform-independent parts of the **Driver**.

10.7.10.7.2 **VGenDriver** comprises the following three parts.

- **VGenDriverCE** is any and/or all instantiations of the CCE class (see **VGenSourceFiles** CE.h and CE.cpp).**VGenDriverMEE** is any and/or all instantiations of the CMEE class (see **VGenSourceFiles** MEE.h and MEE.cpp).

- **VGenDriverDM** is the single instantiation of the CDM class (see **VGenSourceFiles** DM.h and DM.cpp).

10.7.10.7.3 **VGenDriver,** like **VGenLoader,** makes use of **VGenInputFiles** and **VGenTables** in data generation. This provides data generation coherency between database population time and Test Run time.

10.7.10.7.4 The **Sponsor** is responsible for providing a suitable implementation of the Trade-Cleanup **Transaction** (see Clause 10.6.12). Trade-Cleanup may be implemented as a separate, standalone procedure or as part of **VGenDriverDM**.

10.7.10.8   VGenLogger

10.7.10.8.1 **VGenLogger** is used by **VGenDriver** and **VGenLoader** 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 **VGenLogger's** output. For details see VGen/inc/VGenLogger.h.

10.7.10.9   Implementing a CE using VGenDriverCE

10.7.10.9.1 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, **VGenDriverCE** encapsulates this functionality and provides a standardized implementation for it across all TPCx-V implementations.

10.7.10.10   Implementing a MEE using VGenDriverMEE

10.7.10.10.1   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, **VGenDriverMEE** encapsulates this functionality and provides a standardized implementation for it across all TPCx-V implementations.

Comment: A proper **MEE** implementation must to 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.

10.7.10.11   Implementing a Data-Maintenance Generator using VGenDriverDM

10.7.10.11.1   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, **VGenDriverDM** encapsulates this functionality and provides a standardized implementation for it across all TPCx-V implementations.

10.7.10.12 VGenTxnHarness

VGenTxnHarness comprises any and/or all instantiations of:

- CBrokerVolume class excluding the **Sponsor** provided implementation of CBrokerVolumeDBInterface (see **VGenSourceFile** TxnHarnessBrokerVolume.h)
- CCustomerPosition class excluding the **Sponsor** provided implementation of CCustomerPositionDBInterface (see **VGenSourceFile** TxnHarnessCustomerPosition.h)
- CDataMaintenance class excluding the **Sponsor** provided implementation of CDataMaintenanceDBInterface (see **VGenSourceFile** TxnHarnessDataMaintenance.h)
- CMarketFeed class excluding the **Sponsor** provided implementation of CMarketFeedDBInterface (see **VGenSourceFile** TxnHarnessMarketFeed.h)
- CMarketWatch class excluding the **Sponsor** provided implementation of CMarketWatchDBInterface (see **VGenSourceFile** TxnHarnessMarketWatch.h)
- CSecurityDetail class excluding the **Sponsor** provided implementation of CSecurityDetailDBInterface (see **VGenSourceFile** TxnHarnessSecurityDetail.h)
- CTradeCleanup class excluding the **Sponsor** provided implementation of CTradeCleanupDBInterface (see **VGenSourceFile** TxnHarnessTradeCleanup.h)
- CTradeLookup class excluding the **Sponsor** provided implementation of CTradeLookupDBInterface (see **VGenSourceFile** TxnHarnessTradeLookup.h)
- CTradeOrder class excluding the **Sponsor** provided implementation of CTradeOrderDBInterface (see **VGenSourceFile** TxnHarnessTradeOrder.h)
- CTradeResult class excluding the **Sponsor** provided implementation of CTradeResultDBInterface (see **VGenSourceFile** TxnHarnessTradeResult.h)
- CTradeStatus class excluding the **Sponsor** provided implementation of CTradeStatusDBInterface (see **VGenSourceFile** TxnHarnessTradeStatus.h)
- CTradeUpdate class excluding the **Sponsor** provided implementation of CTradeUpdateDBInterface (see **VGenSourceFile** TxnHarnessTradeUpdate.h)

10.7.10.13 Functional Implementation

The following diagram gives a high level overview of a sample implementation of the TPCx-V environment. A number of details have been omitted for clarity.
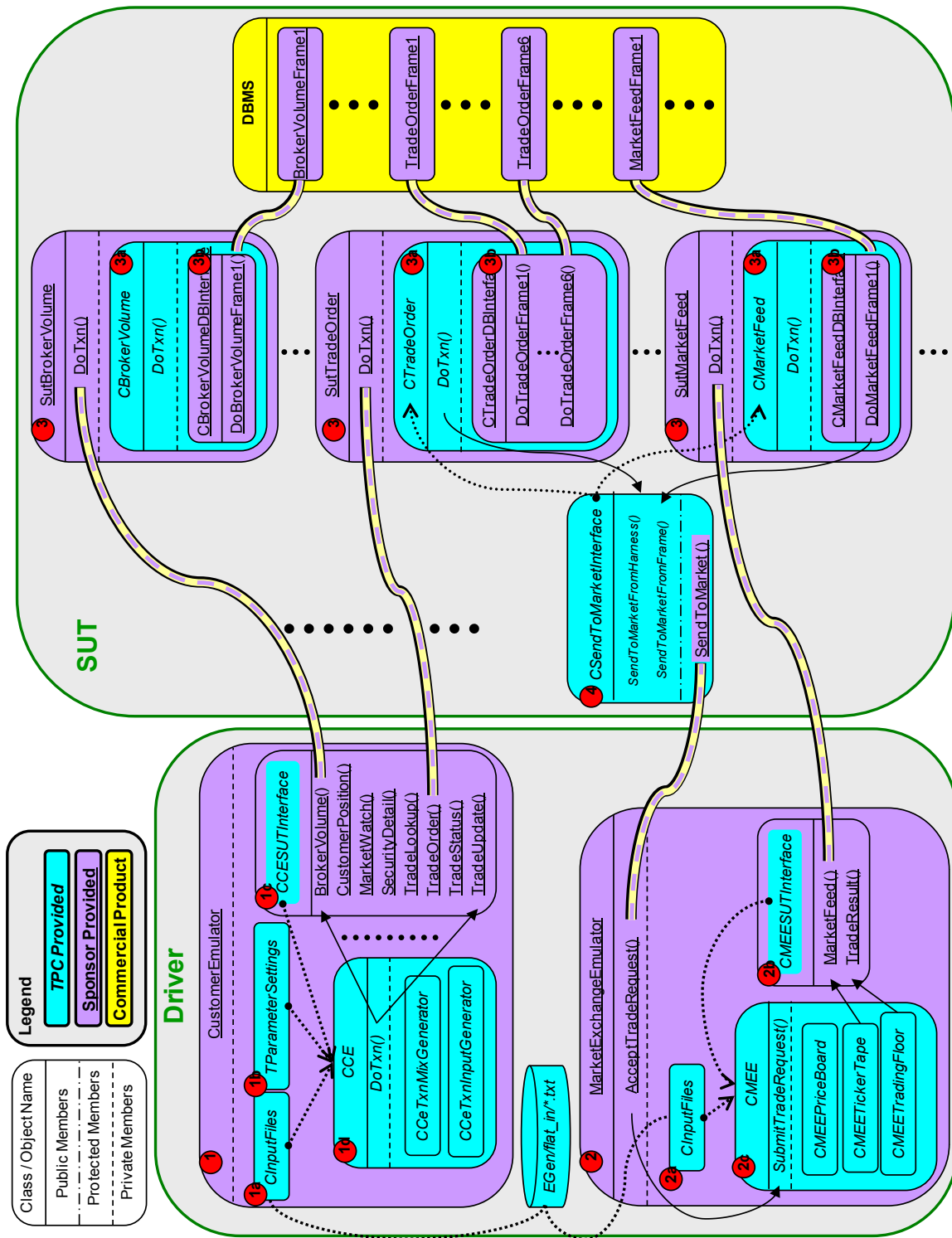
Figure A.b - High Level Overview of a Sample Implementation

In the diagram above,

- dotted "lines" with arrows between TPC Provided objects represent input parameters
- dotted "lines" without arrows between TPC Provided objects represent input files from **VGenInputFiles**
- Solid "lines" with arrows are calls

1. The **Test Sponsor** is responsible for implementing a **Customer Emulator** per Clauses 10.7.7.5 and 10.7.10.8.

   a. CInputFiles is a class provided as part of **VGen** used for loading into memory the **VGenInputFiles** used by other classes in **VGen**. The **Test Sponsor** is responsible for instantiating a CInputFile object correctly and passing a pointer to it into the CCE constructor. See VGen/inc/InputFlatFilesStructure.h.

   b. TParameterSettings is a TPC provided structure that can be used to alter the behavior of **VGenDriver**. Use of this structure for a compliant run is not required; it is provided to facilitate prototyping and engineering work. See VGen/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, including the **Tile** and the **Group** of the transaction. A pointer to the **Sponsor's** implementation of the CCESUTInterface must be passed into the CCE constructor. See VGen/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 VGen/inc/CE.h.

13. The Test Sponsor is responsible for implementing a Market Exchange Emulator per Clauses 10.7.7.6 and 10.7.10.10.

   a. CSecurityFile is a class provided as part of **VGen** used for loading VGen/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 VGen/inc/SecurityFile.h.

   b. CMEESUTInterface is a TPC provided pure virtual class that defines an interface used by the CMEE class. It is the **Sponsor's** responsibility to subclass CMEESUTInterface 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, including the **Tile** and the **Group** of the transaction. A pointer to the **Sponsor's** implementation of the CMEESUTInterface must be passed into the CMEE constructor. See VGen/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 CMEESUTInterface 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 VGen/inc/MEE.h.

14. 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 **VGenTxnHarness** 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 VGen/inc/TxnHarnessBrokerVolume.h – TxnHarnessTradeUpdate.h.

The **Sponsor** is responsible for providing implementations for the following classes used by **VGenTxnHarness**.

- CBrokerVolumeDBInterface

- CCustomerPositionDBInterface

- CMarketFeedDBInterface

- CMarketWatchDBInterface

- CSecurityDetailDBInterface

- CTradeLookupDBInterface

- CTradeOrderDBInterface

- CTradeResultDBInterface

- CTradeStatusDBInterface

- CTradeUpdateDBInterface

- These classes are responsible for implementing the **Frames** invoked by **VGenTxnHarness**.

15. 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 **VGenTxnHarness** objects CTradeOrder and CMarketFeed.

10.7.10.14 TPC Defined Interfaces

Connector	Attachment Point	Interface (Class::Method)
VGenDriver	Driving and Reporting	CCE::DoTxn() CMEE::SubmitTradeRequest() CDM::DoTxn() CDM::DoCleanupTxn()
VGenDriver	VGenDriver Connector	CCESUTInterface::BrokerVolume() CCESUTInterface::CustomerPosition() CMEESUTInterface::MarketFeed() CCESUTInterface::MarketeWatch() CCESUTInterface::SecurityDetail() CCESUTInterface::TradeLookup() CCESUTInterface::TradeOrder() CMEESUTInterface::TradeResult() CCESUTInterface::TradeStatus() CCESUTInterface::TradeUpdate() CDMSUTInterface::DataMaintenance() CDMSUTInterface::TradeCleanup()
VGenTxnHarness	VGenTxnHarness Connector	CBrokerVolume::DoTxn() CCustomerPosition::DoTxn() CMarketFeed::DoTxn() CMarketWatch::DoTxn() CSecurityDetail::DoTxn() CTradeLookup::DoTxn() CTradeOrder::DoTxn() CTradeResult::DoTxn() CTradeStatus::DoTxn() CTradeUpdate::DoTxn() CDataMaintenance::DoTxn() CTradeCleanup::DoTxn()
VGenTxnHarness	Frame Implementation	CBrokerVolumeDBInterface::DoBrokerVolumeFrame1() CCustomerPositionDBInterface::DoCustomerPositionFrame1/2/3() CMarketFeedDBInterface ::DoMarketFeedFrame1() CMarketWatchDBInterface::DoMarketWatchFrame1/2/3() CSecurityDetailDBInterface::DoSecurityDetailFrame1() CTradeLookupDBInterface::DoTradeLookupFrame1/2/3/4() CTradeOrderDBInterface::DoTradeOrderFrame1/2/3/4/5/6() CTradeResultDBInterface::DoTradeResultFrame1/2/3/4/5/6() CTradeStatusDBInterface::DoTradeStatusFrame1 CTradeUpdateDBInterface::DoTradeUpdateFrame1/2/3/4() CTradeResultDBInterface::DoTradeResultFrame1/2/3/4/5/6() CDataMaintenanceDBInterface::DoDataMaintenanceFrame1() CTradeCleanupDBInterface::DoTradeCleanupFrame1()

[A TPCx-V Executive Summary sample will be provided by the TPCx-V subcommittee.]

A.1 **Sample Layouts**

A.2 **Sample Executive Summary Statement**

	ABC Hyperia XP100 Virtuosa Nirvana 10.1	TPCx-V 1.0
		TPC Pricing 0.0.0
		Report Date: Apr 2, 2015

Availability Date	**TPCx-V Throughput**	**Price/Performance**	**Total System Cost**
Dec 32, 2099	425.60 tpsV	0 USD/tpsV	$0 USD

System Under Test Configuration Overview			
Virtualization Software	Database Software	Processor Description	Memory Size
Virtuosa Nirvana 10.1	DEF MoreSQL 1.0	XYZ HyperFast 2121 3.99GHz, 64MB L3 2/64/64 (proc/core/thr)	512 GB

Server
Model: ABC Hyperia XP100

- 2x XYZ HyperFast 2121 Processor 3.99GHz (2/64/64)
- 16x 32GB DDR3 1866 MHz DIMMs
- 4x ABC Storage Array P123/4GB, one per DB VM
- 2x 128GB SFF SAS 15K dual-port HDD (boot)
- 2x 10Gb Ethernet (onboard)

Clients

- 4x ABC WS123 Workstation
- 2x XYZ KindaFast 1010 Processor 1.99GHz (2/16/16)
- 2x 8GB PC3-8500 DIMMs
- 2x 128GB SFF SAS 15K HDD
- 2x 1Gb Ethernet (onboard)

Storage:

- 4x ABC D9000 Disk Enclosure (one per DB VM)
- 32x ABC 512GB SFF SLC SATA 2.5-inch SSD, 8 per enclosure
- Priced: 24x 512GB 15K SFF HDD

Network:

- ABC LinkUp E9000 24-port 1/10 Network Switch
- 8x 1Gb ports used, 2x 10Gb ports used

	TPCx-V	1.0
	TPC-Pricing	0.0.0
	Report Date	Apr 2, 2015

ABC Hyperia XP100
Virtuosa Nirvana 10.1

Availability Date | Dec 32, 2099

Description	Part Number	Vend	Unit Price	Qty.	Extended Price	3-Yr Maint Price
Server Hardware						
				Subtotal		
Server Software						
				Subtotal		
Storage						
				Subtotal		
Client Hardware						
				Subtotal		
Client Software						
				Subtotal		
Infrastructure						
				Subtotal		
Discounts						
				Subtotal		

Total Price	$0	$0
Total Discounts	$0	$0
Grand Total	$0	$0
3-yr Total Cost of Ownership		$0
tpsV		425.60
$ USD/tpsV		$0

Vendor Legend:

Notes:
1: Prices and descriptions included couldn't be more random

		TPCx-V 1.0
		TPC Pricing 0.0.0
ABC Hyperia XP100 **Virtuosa Nirvana 10.1**		Report Date: Apr 2, 2015

Database VM Details

Database Manager	Database OS	Memory (GB)	vCPUs	DB Initial Size	Customers Configured
DEF MoreSQL 1.0	Nirvana OS-V 1.0	32 GB	4	555,345,678,901	125,000

Transaction Response Times (in seconds)

Transaction Type	Min	Avg	90th%	Max
Trade-Order	0.003	0.035	0.077	1.008
Trade-Result	0.005	0.054	0.124	1.010
Trade-Lookup	0.001	0.107	0.191	1.033
Trade-Update	0.010	0.129	0.209	1.031
Trade-Status	0.002	0.011	0.019	1.060
Customer-Position	0.002	0.016	0.028	1.017
Broker-Volume	0.001	0.006	0.010	1.155
Security-Detail	0.005	0.014	0.019	1.151
Market-Feed	0.003	0.006	0.008	0.269
Market-Watch	0.000	0.011	0.020	1.114
Data-Maintenance	null	null	null	null

Transaction Mix

Transaction Type	Transaction Count	Mix Percentage
Trade-Order	784,334	10.054
Trade-Result	766,085	9.820
Trade-Lookup	699,250	8.963
Trade-Update	77,690	0.996
Trade-Status	1,398,378	17.925
Customer-Position	1,165,549	14.940
Broker-Volume	303,092	3.885
Security-Detail	1,242,939	15.932
Market-Feed	43,199	0.554
Market-Watch	1,320,821	16.931
Data-Maintenance	0	0.000

Total Transactions		7,801,337
Measurement Interval		00:30:00
Business Recovery Time		12:34:56
Redundancy Level Details	All storage was configured with redundancy level 1	
Auditor		